

MAIN MENU

Indice

Introducción

Resumen RAPID

Characterísticas Básicas RAPID

Principios de Movimiento y de E/S

Tipos de Datos

Instrucciones

Funciones

Datos y Programas Predefinidos

Programación Off-line

ArcWare

SpotWare

GlueWare

Referencia Rápida

Funciones especiales de este robot

Indice, Glosario

INDICE

	Página
1 Indice	1-1
2 Introducción.....	2-1
1 Otros Manuales	2-3
2 Cómo utilizar este Manual.....	2-3
2.1 Convenciones tipográficas	2-4
2.2 Reglas de sintaxis.....	2-5
2.3 Sintaxis formal	2-6
3 Sumário RAPID	
1 A Estrutura da Linguagem	3-5
2 Control del Flujo del Programa	3-6
2.1 Principios de programación	3-6
2.2 Llamada a otra rutina	3-6
2.3 Control del programa dentro de la rutina	3-7
2.4 Paro de la ejecución del programa	3-7
2.5 Paro del ciclo actual	3-7
3 Instrucciones varias	3-8
3.1 Asignar un valor a un dato	3-8
3.2 Condición de Espera	3-8
3.3 Comentarios	3-8
3.4 Carga de módulos de programa	3-9
3.5 Funciones varias.....	3-9
3.6 Datos básicos.....	3-9
3.7 Función de conversión	3-9
4 Características de movimiento	3-10
4.1 Principios de programación	3-10
4.2 Definición de la velocidad	3-10
4.3 Definición de la aceleración.....	3-11
4.4 Definición de la gestión de la configuración	3-11
4.5 Definición de la carga útil.....	3-11
4.6 Definición del comportamiento del robot cerca de un punto singular.....	3-11
4.7 Desplazamiento de un programa.....	3-12
4.8 Servo suave	3-12
4.9 Valores de ajuste del robot.....	3-12
4.10 Zonas Mundo	3-13
4.11 Datos para las características de movimiento	3-13
5 Movimiento.....	3-14

5.1 Principios de programación	3-14
5.2 Instrucciones de posicionamiento	3-15
5.3 Búsqueda.....	3-15
5.4 Activación de salidas o interrupciones en posiciones específicas	3-15
5.5 Control del movimiento en caso de ocurrir un error/interrupción	3-16
5.6 Control de los ejes externos	3-16
5.7 Ejes independientes.....	3-17
5.8 Funciones de posición.....	3-17
5.9 Datos de movimiento	3-18
5.10 Datos básicos para los movimientos	3-18
6 Señales de entrada y salida	3-19
6.1 Principios de programación	3-19
6.2 Cambio del valor de una señal	3-19
6.3 Lectura del valor de una señal de entrada.....	3-19
6.4 Lectura del valor de una señal de salida	3-19
6.5 Comprobación de entradas en señales de salida	3-20
6.6 Inhabilitación y habilitación de los módulos de E/S.....	3-20
6.7 Definición de las señales de entrada y salida.....	3-20
7 Comunicación.....	3-21
7.1 Principios de programación	3-21
7.2 Comunicación utilizando la unidad de programación	3-22
7.3 Lectura desde o escritura en un canal/archivo serie basado en caracteres....	3-22
7.4 Comunicación mediante canales serie/archivos binarios.....	3-22
7.5 Datos de los canales serie	3-23
8 Interrupciones	3-24
8.1 Principios de programación	3-24
8.2 Conexión de las interrupciones a las rutinas de tratamiento de interrupción	3-24
8.3 Petición de interrupciones.....	3-25
8.4 Anulación de interrupciones	3-25
8.5 Habilitación/Inhabilitación de las interrupciones	3-25
8.6 Tipos de datos de las interrupciones	3-25
9 Recuperación de errores	3-26
9.1 Principios de programación.....	3-26
9.2 Creación de una situación de error desde dentro del programa	3-26
9.3 Rearranque/regreso del gestor de errores	3-27
9.4 Datos para la gestión de los errores.....	3-27

10 Sistema & Tempo	3-28
10.1 Princípios de programação	3-28
10.2 Utilização do relógio para temporizar um evento	3-28
10.3 Leitura da hora e data atuais	3-28
11 Matemáticas	3-29
11.1 Principios de programación	3-29
11.2 Cálculos sencillos sobre datos numéricos	3-29
11.3 Cálculos más avanzados	3-29
11.4 Funciones aritméticas	3-30
12 Soldadura por Puntos	3-31
12.1 Características de la soldadura por puntos	3-32
12.2 Principios de SpotWare	3-33
12.3 Principios de programación	3-34
12.4 Instrucciones de soldadura por puntos	3-34
12.5 Datos de soldadura por puntos	3-34
13 Soldadura al Arco	3-35
13.1 Principios de programación	3-35
13.2 Instrucciones de soldadura al arco	3-35
13.3 Datos de soldadura al arco	3-36
14 GlueWare	3-37
14.1 Características de aplicación de adhesivo	3-37
14.2 Principios de programación	3-37
14.3 Instrucciones de aplicación de adhesivo	3-38
14.4 Datos de aplicación de adhesivo	3-38
15 Comunicación externa del computador	3-39
15.1 Principios de programación	3-39
15.2 Envío de un mensaje controlado por el programa desde el robot al computador	3-39
16 Instrucciones de Servicio	3-40
16.1 Direcciónamiento de un valor a una señal de test del robot	3-40
17 Funciones de cadena	3-41
17.1 Operaciones Básicas	3-41
17.2 Comparación y Búsqueda	3-41
17.3 Conversión	3-41
18 Resumen de sintaxis	3-42
18.1 Instrucciones	3-42

	Página
18.2 Funciones	3-46
4	
5 Características Básicas RAPID	5-1
1 Elementos Básicos	5-3
1.1 Identificadores.....	5-3
1.2 Espacios y caracteres de fin de línea.....	5-4
1.3 Valores numéricos	5-4
1.4 Valores lógicos.....	5-4
1.5 Valores de cadena	5-4
1.6 Comentarios	5-5
1.7 Comodines	5-5
1.8 Encabezado de archivo	5-6
1.9 Sintaxis.....	5-6
2 Módulos	5-8
2.1 Módulos del programa	5-8
2.2 Módulos del sistema	5-9
2.3 Declaraciones de los módulos.....	5-9
2.4 Sintaxis.....	5-10
3 Rutinas	5-11
3.1 Alcance de las rutinas	5-11
3.2 Parámetros.....	5-12
3.3 Final de una rutina	5-13
3.4 Declaraciones de rutina.....	5-13
3.5 Llamada de procedimiento.....	5-14
3.6 Sintaxis.....	5-15
4 Tipos de Datos	5-18
4.1 Tipos de datos sin valor	5-18
4.2 Tipos de datos iguales a (equivalente a)	5-18
4.3 Sintaxis.....	5-19
5 Datos	5-20
5.1 Alcance de los datos.....	5-20
5.2 Declaración de un dato variable.....	5-21
5.3 Declaración de un dato persistente	5-22
5.4 Declaración de un dato constante	5-22
5.5 Datos de inicio	5-22
5.6 Sintaxis.....	5-23
6 Instrucciones	5-25

	Página
6.1 Sintaxis	5-25
7 Expresiones.....	5-26
7.1 Expresiones aritméticas.....	5-26
7.2 Expresiones lógicas.....	5-27
7.3 Expresiones de cadena	5-27
7.4 Utilización de datos en las expresiones.....	5-28
7.5 Utilización de agregados en las expresiones.....	5-29
7.6 Utilización de llamadas a función en las expresiones.....	5-29
7.7 Prioridad entre los operadores.....	5-30
7.8 Sintaxis	5-31
8 Recuperación de Errores.....	5-34
8.1 Gestores de error	5-34
9 Interrupciones	5-36
9.1 Procesamiento de las interrupciones	5-36
9.2 Rutinas de tratamiento de las interrupciones.....	5-37
10 Ejecución hacia atrás.....	5-39
10.1 Gestores de ejecución hacia atrás.....	5-39
10.2 Limitación de instrucciones de movimiento en el gestor de ejecución hacia atrás	5-40
11 Multitareas	5-42
11.1 Sincronización de las tareas	5-43
11.2 Comunicación entre tareas	5-45
11.3 Tipo de tarea.....	5-46
11.4 Prioridades	5-46
11.5 Tamaño de las tareas	5-47
11.6 Recomendación importante.....	5-48
6 Principios de Movimiento y de E/S	6-1
1 Sistemas de Coordenadas.....	6-3
1.1 El punto central de la herramienta del robot (TCP)	6-3
1.2 Sistemas de coordenadas utilizados para determinar la posición del TCP ...	6-3
1.3 Sistemas de coordenadas utilizados para determinar la dirección de la herramienta 6-8	
1.4 Información relacionada	6-12
2 Posicionamiento durante la Ejecución del Programa	6-13
2.1 Generalidades.....	6-13

	Página
2.2 Interpolación de la posición y orientación de la herramienta	6-13
2.3 Interpolación de las trayectorias esquina	6-16
2.4 Ejes independientes.....	6-22
2.5 Servo Suave	6-25
2.6 Paro y rearranque	6-25
2.7 Información relacionada	6-26
3 Sincronización con Instrucciones Lógicas	6-27
3.1 Ejecución secuencial del programa en los puntos de paro.....	6-27
3.2 Ejecución secuencial del programa en los puntos de paso	6-27
3.3 Ejecución concurrente del programa	6-28
3.4 Sincronización de la trayectoria.....	6-30
3.5 Información relacionada	6-31
4 Configuración del Robot	6-32
4.1 Datos para la configuración del robot 6400C	6-34
4.2 Información relacionada	6-35
5 Puntos Singulares	6-36
5.1 Ejecución del programa con puntos singulares.....	6-37
5.2 Movimiento con puntos singulares	6-37
5.3 Información relacionada	6-38
6 Zonas Mundo	6-39
6.1 Utilización de las Zonas.....	6-39
6.2 Utililidad de las Zonas Mundo.....	6-39
6.3 Definición de las Zonas Mundo en el sistema de coordenadas mundo	6-39
6.4 Supervisión del TCP del robot.....	6-40
6.5 Acciones.....	6-41
6.6 Tamaño mínimo de las Zonas Mundo	6-42
6.7 Reinicio después de un corte del suministro de alimentación	6-42
6.8 Información relacionada	6-43
7 Principios de E/S.....	6-44
7.1 Características de las señales	6-44
7.2 Señales del sistema	6-45
7.3 Conexiones enlazadas	6-45
7.4 Limitaciones.....	6-46
7.5 Información relacionada	6-47
7 Tipos de Datos	7-1
bool - Valores lógicos.....	7-bool-1

clock - Medida del tiempo	7-clock-1
confdata - Datos de configuración del robot	7-confdata-1
corrdescr - Descriptor de generador de correcciones.....	7-corrdescr-5
dionum - Valores digitales 0 - 1	7-dionum-1
errnum - Número de error	7-errnum-1
extjoint - Posición de los ejes externos	7-extjoint-1
intnum - Identificación de la interrupción	7-intnum-1
iodev - Canales y archivos de serie	7-iodev-1
jointtarget - Datos de posición de los ejes	7-jointtarget-1
loaddata - Datos de carga.....	7-loaddata-1
mecunit - Unidad mecánica.....	7-mecunit-1
motsetdata - Datos de movimiento	7-motsetdata-1
num - Valores numéricos (registros).....	7-num-1
o_jointtarget - Dato de posición original de un eje	7-o_jointtarget-1
orient - Orientación	7-orient-1
o_robtarget - Datos de posición original.....	7-o_robtarget-1
pos - Posiciones (sólo X, Y y Z)	7-pos-1
pose - Transformación de coordenadas	7-pose-1
progdisp - Desplazamiento de programa.....	7-progdisp-1
robjoint - Posición de los ejes del robot	7-robjoint-1
robtarget - Datos de posición	7-robtarget-1
shapedata - Datos de forma de una zona mundo	7-shape-1
signalxx - Señales digitales y analógicas	7-signalxx-1
speeddata - Datos de velocidad.....	7-speeddata-1
string - Cadenas de Carácteres.....	7-string-1
symnum - Número simbólico	7-symnum-1
tooldata - Datos de herramienta	7-tooldata-1
tpnum - Número de ventana de la Unidad de Programación ...	7-tpnum-1
trigedata - Eventos de posicionamiento - disparo.....	7-trigedata-1
tunetype - Tipo de ajuste servo.....	7-tunetype-1
wobjdata - Datos del objeto de trabajo.....	7-wobjdata-1
wzstationary - Datos de zona mundo estacionaria	7-wzstationary-1
wztemporary - Datos de zona mundo temporal	7-wztemporary-1
zonedata - Datos de zona	7--zonedata-1
Datos del Sistema	7-System data-1

8 Instrucciones	8-1
“:=” - Asignación de un valor	8-:=1
AccSet - Reducción de la aceleración.....	8-AccSet-1
ActUnit - Activación de una unidad mecánica.....	8-ActUnit-1
Add - Adición de un valor numérico.....	8-Add-1
Break - Interrupción de la ejecución del programa	8-Break-1
CallByVar - Llamada de un procedimiento mediante una variable	8-CallByVar-1
Clear - Borrado de un valor.....	8-Clear-1
ClkReset - Puesta a cero de un reloj para el cronometraje	8-ClkReset-1
ClkStart - Arranque de un reloj para el cronometraje.....	8-ClkStart-1
ClkStop - Paro de un reloj de cronometraje	8-ClkStop-1
Close - Cerrar un archivo o un canal serie.....	8-Close-1
comment - Comentarios	8-comment-1
Compact IF - Si se cumple una condición, entonces... (unas instrucciones).....	8-Compact IF-1
ConfJ - Control de la configuración durante el movimiento eje a eje	8-ConfJ-1
ConfL - Control de la configuración durante el movimiento lineal	8-ConfL-1
CONNECT - Conexión de una interrupción a una rutina de tratamiento de interrupciones.....	8-CONNECT-1
CorrClear - Eliminación de todos los generadores de correcciones	8-CorrClear-1
CorrCon - Conexión de un generador de correcciones.....	8-CorrCon-1
CorrDiscon - Desconexión de un generador de correcciones	8-CorrDiscon-1
CorrWrite - Escritura en un generador de correcciones	8-CorrWrite-1
DeactUnit - Desactivación de una unidad mecánica.....	8-DeActUnit-1
Decr - Disminución de 1	8-Decr-1
EOffsOff - Desactivación de un offset de los ejes externos	8-EOffsOff-1
EOffsOn - Activación de un offset de los ejes externos	8-EOffsOn-1
EOffsSet - Activación de un offset de ejes externos utilizando un valor	8-EOffsSet-1
ErrWrite - Escribir un Mensaje de Error	8-ErrWrite-1
EXIT - Fin de ejecución del programa	8-EXIT-1
ExitCycle - Interrupción del ciclo actual y arranque del siguiente	8-ExitCycle-1
FOR - Repetición de un número dado de veces	8-FOR-1
GOTO - Ir a una instrucción nueva identificada por una etiqueta (label)	8-GOTO-1
GripLoad - Definición de la carga útil del robot	8-GripLoad-1
IDelete - Anulación de una interrupción	8-IDelete-1

IDisable - Inhabilitación de las interrupciones	8-IDisable-1
IEnable - Habilitación de las interrupciones.....	8-IEnable-1
IF - Si se cumple una condición, entonces...; si no...	8-IF-1
Incr - Incremento de 1	8-Incr-1
IndAMove - Movimiento de una posición absoluta independiente	8-IndAMove-1
IndCMove - Movimiento continuo independiente	8-IndCMove-1
IndDMove - Movimiento de una posición delta independiente.	8-IndRMove-1
IndReset - Reinicialización independiente.....	8-IndReset-1
IndRMove - Movimiento de una posición relativa independiente	8-IndRMove-1
InvertDO - Inversión del valor de una señal de salida digital ...	8-InvertDO-1
IODisable - Inhabilitar la unidad de E/S.....	8-IODisable-1
IOEnable - Habilitar la unidad de E/S.....	8-IOEnable-1
ISignalDI - Orden de interrupción a partir de una señal de entrada digital.....	8-ISignalDI-1
ISignalDO - Orden de interrupción a partir de una señal de salida digital	8-ISignalDO-1
ISleep - Desactivación de una interrupción.....	8-ISleep-1
ITimer - Ordena una interrupción temporizada	8-ITimer-1
IVarValue - Ordena una interrupción de un valor de variable	8-IVarVal-1
IWatch - Habilitación de una interrupción	8-IWatch-1
label - Nombre de línea.....	8-label-1
Load - Cargar un módulo de programa durante la ejecución...	8-Load-1
MoveAbsJ - Movimiento del robot a una posición de ejes absoluta	8-MoveAbsJ-1
MoveC - Movimiento circular del robot	8-MoveC-1
MoveJ - Movimiento eje a eje del robot.....	8-MoveJ-1
MoveL - Movimiento lineal del robot	8-MoveL-1
Open - Apertura de un archivo o de un canal serie.....	8-Open-1
PathResol - Ajuste de la resolución de trayectoria	8-PathResol-1
PDispOff - Desactivación de un desplazamiento de programa..	8-PDispOff-1
PDispOn - Activación de un desplazamiento de programa	8-PDispOn-1
PDispSet - Activación de un desplazamiento del programa utilizando un valor	8-PDispSet-1
ProcCall - Llamada de un procedimiento nuevo	8-ProcCall-1
PulseDO - Generación de un pulso en una señal de salida digital	8-PulseDO-1
RAISE - Llamada al gestor de error.....	8-RAISE-1
Reset - Puesta a cero de una señal de salida digital	8-Reset-1
RestoPath - Restauración de la trayectoria después de una	

interrupción	8-RestoPath-1
RETRY - Rearranque después de un error	8-RETRY-1
RETURN - Fin de ejecución de una rutina	8-RETURN-1
Rewind - Reiniciar la posición del archivo.....	8-Rewind-1
SearchC - Búsqueda circular utilizando el robot	8-SearchC-1
SearchL - Búsqueda lineal utilizando el robot.....	8-SearchL-1
Set - Activación de una señal de salida digital	8-Set-1
SetAO - Cambio del valor de una señal de salida analógica.....	8-SetAO-1
SetDO - Cambio del valor de una señal de salida digital	8-SetDO-1
SetGO - Cambio del valor de un grupo de señales de salidas digitales	8-SetGO-1
SingArea - Definición de la interpolación en torno a puntos singulares	8-SingArea-1
SoftAct - Activación del servo suave	8-SoftAct-1
SoftDeact - Desactivación del servo suave	8-SoftDeAct-1
StartMove - Rearranque del movimiento del robot	8-StartMove-1
Stop - Paro de la ejecución del programa.....	8-Stop-1
StopMove - Paro del movimiento del robot.....	8-StopMove-1
StorePath - Almacenamiento de una trayectoria cuando se produce una interrupción.....	8-StorePath-1
TEST - Dependiendo del valor de una expresión...	8-TEST-1
TPErase - Borrado del texto impreso en la unidad de programación	8-TPErase-1
TPReadFK - Lectura de las teclas de función.....	8-TPReadFK-1
TPReadNum - Lectura de un número en la unidad de programación.....	8-TPReadNum-1
TPShow - Cambio de ventana de la unidad de programación ...	8-tpshow-1
TPWrite - Escritura en la unidad de programación	8-TPWrite-1
TriggC - Movimiento circular del robot - con eventos.....	8-TriggC-1
TriggEquip - Definición de un evento de E/S de tiempo-posición fijas	8-TriggEquip-1
TriggInt - Definición de una interrupción relativa a una posición	8-TriggInt-1
TriggIO - Definición de un evento de E/S de posición fija	8-TriggIO-1
TriggJ - Movimientos de los ejes del robot con eventos	8-TriggJ-1
TriggL - Movimientos lineales del robot con eventos.....	8-TriggL-1
TRYNEXT - Salto de una instrucción que ha causado un error	8-TRYNEXT-1
TuneReset - Reinicialización del ajuste del servo	8-TuneServo-1
TuneServo - Ajuste de los servos.....	8-TuneServo-1
UnLoad - Descarga de un módulo de programa durante la ejecución	8-UnLoad-1
VelSet - Cambio de la velocidad programada.....	8-VelSet-1
WaitDI - Espera hasta la activación de una señal - de entrada digital	8-WaitDI-1

WaitDO - Espera hasta la activación de una señal de salida digital	8-WaitDO-1
WaitTime - Espera durante un tiempo especificado	8-WaitTime-1
WaitUntil - Esperar hasta el cumplimiento de una condición...	8-WaitUntil-1
WZBoxDef - Definición de una zona mundo en forma de caja rectangular	8-WZBoxDef-1
WZCylDef - Definición de una zona mundo en forma de cilindro	8-WZCylDef-1
WZDisable - Desactivación de la supervisión de la zona mundo temporal	8-WZDisable-1
WZDOSet - Activación de la zona mundo para activar una salida digital.....	8-WZDOSet-1
WZEnable - Activación de la supervisión de una zona mundo temporal	8-WZEnable-1
WZFree - Borrado de la supervisión de la zona mundo temporal	8-WZFree-1
WZLimSup - Activación de la supervisión del límite de la zona mundo	8-WZLimSup-1
WZSphDef - Definición de una zona mundo en forma de esfera	8-WZSphDef-1
WHILE - Repetición de una instrucción mientras...	8-WHILE-1
Write - Escritura en un archivo basado en caracteres o en un canal serie	8-Write-1
WriteBin - Escritura en un canal serie binario.....	8-WriteBin-1
WriteStrBin - Escritura de una cadena en un canal serie binario	8-WriteStrBin-1
9 Funciones.....	9-1
Abs - Obtención del valor absoluto	9-Abs-1
ACos - Cálculo del valor del arco coseno.....	9-ACos-1
AOutput - Lectura del valor de una señal de salida analógica..	9-AOutput-1
ArgName - Coger el nombre de un argumento.....	9-ArgName-1
ASin - Cálculo del valor del arco seno	9-ASin-1
ATan - Cálculo del valor del arco tangente.....	9-ATan-1
ATan2 - Cálculo del valor del arco tangente2.....	9-ATan2-1
ByteToStr - Conversión de un byte en un dato de cadena	9-ByteToStr-1
CDate - Lectura de la fecha actual como una cadena	9-CDate-1
CJointT - Lectura de los ángulos actuales de los ejes.....	9-CJointT-1
ClkRead - Lectura de un reloj utilizado para el cronometraje.	9-ClkRead-1
CorrRead - Lectura de todos los offsets utilizados	9-CorrRead-1
Cos - Cálculo del valor del coseno	9--Cos-1
CPos - Lectura de los datos de posición actuales (pos).....	9-CPos-1
CRobT - Lectura de los datos de la posición actuales (robtarget)	9-CRobT-1
CTime - Lectura de la hora actual como una cadena.....	9-CTime-1
CTool - Lectura de los datos de herramienta actuales	9-CTool-1

CWObj - Lectura de los datos del objeto de trabajo actuales ...	9-CWobj-1
DefDFrame - Definición de una base de desplazamiento	9-DefDFrame-1
DefFrame - Definición de una base de coordenadas	9-DefFrame-1
Dim - Obtención del tamaño de una matriz.....	9-Dim-1
DOutput - Lectura del valor de una señal de salida digital.....	9-DOutput-1
EulerZYX - Obtención de ángulos Euler a partir de una variable de orientación	9-EulerZYX-1
Exp - Cálculo del valor exponencial	9-Exp-1
GetTime - Lectura de la hora actual como un valor numérico..	9-GetTime-1
GOutput - Lectura del valor de un grupo de señales de salida digital	9-GOutput-1
IndInpos - Estado de la posición independiente	9-IndInpos-1
IndSpeed - Estado de la velocidad independiente	9-IndSpeed-1
IsPers - Es Persistente	9-IsPers-1
IsVar - Es Variable.....	9-IsVar-1
MirPos - Creación de la imagen espejo de una posición.....	9-MirPos-1
NumToStr - Conversión de un valor numérico en cadena	9-NumToStr-1
Offs - Desplazamiento de una posición del robot	9-Offs-1
OpMode - Lectura del modo de operación	9-OpMode-1
OrientZYX - Cálculo de una variable de orientación a partir de ángulos Euler	9-OrientZYX-1
ORobT - Eliminación de un desplazamiento de programa de una posición	9-ORobT-1
PoseInv - Inversión de la posición.....	9-PoseInv-1
PoseMult - Multiplicación de los datos de posición.....	9-PoseMult-1
PoseVect - Aplicación de una transformación a un vector.....	9-PoseVect-1
Pow - Cálculo de la potencia de un valor	9-Pow-1
Present - Comprobación de la utilización de un parámetro opcional	9-Present-1
ReadBin - Lectura a partir de un canal serie binario o archivo	9-ReadBin-1
ReadMotor - Lectura de los ángulos del motor actuales	9-ReadMotor-1
ReadNum - Lectura de un número a partir de un archivo o de un canal serie	9-ReadNum-1
ReadStr - Lectura de una cadena a partir de un archivo o de un canal serie.....	9-ReadStr-1
RelTool - Ejecución de un desplazamiento relativo a la herramienta	9-RelTool-1
Round - Round es un valor numérico	9-Round-1
RunMode - Lectura del modo de funcionamiento.....	9-RunMode-1
Sin - Cálculo del valor del seno	9-Sin-1
Sqrt - Cálculo del valor de la raíz cuadrada.....	9-Sqrt-1

StrFind - Búsqueda de un carácter en una cadena.....	9-StrFind-1
StrLen - Obtención de la longitud de la cadena.....	9-StrLen-1
StrMap - Mapa de una cadena	9-StrMap-1
StrMatch - Búsqueda de una estructura en una cadena	9-StrMatch-1
StrMemb - Comprobar si un carácter pertenece a un conjunto	9-StrMemb-1
StrOrder - Comprobar si las cadenas están ordenadas	9-StrOrder-1
StrPart - Obtención de una parte de una cadena	9-StrPart-1
StrToByte - Conversión de una cadena en un dato byte	9-StrToByte-1
StrToVal - Conversión de una cadena en un valor numérico....	9-StrToVal-1
Tan - Cálculo del valor de la tangente	9-Tan-1
TestDI - Comprobación de la activación de una entrada digital	9-TestDI-1
Trunc - Truncar un valor numérico	9-Trunc-1
ValToStr - Conversión de un valor en una cadena.....	9-ValToStr-1
10 Datos y Programas Predefinidos	10-1
1 Módulo User del Sistema.....	10-3
1.1 Contenido	10-3
1.2 Creación de nuevos datos en este módulo	10-3
1.3 Eliminación de estos datos	10-4
11 Programación Off-line	11-1
1 Programación Off-line.....	11-3
1.1 Formato de los archivos	11-3
1.2 Edición	11-3
1.3 Comprobación de la sintaxis	11-3
1.4 Ejemplos.....	11-4
1.5 Creación de instrucciones propias del usuario	11-5
12	
13 ArcWare	13-1
seamdata - Datos iniciales y finales de soldadura	13-seamdata-1
weavedata - Datos de oscilación.....	13-weavedata-1
welldata - Datos de soldadura.....	13-welldata-1
ArcC - Soldadura al arco con movimiento circular.....	13-ArcC-1
arcdata - Datos de proceso de soldadura al arco	13-arcdata-11
ArcL - Soldadura al arco con movimiento lineal	13-ArcL-1
ArcKill - Aborto del proceso de soldadura al arco	13-ArcKill-1
ArcRefresh - Restablecer datos de soldadura al arco	13-ArcRefr-1

14	SpotWare	14-1
	gundata - Dado da pinça de solda a ponto.....	14-gundata-1
	spotdata - Dado de solda a ponto	14-spotdata-1
	SpotL - Soldadura por puntos con movimiento	14-SpotL-1
	Módulo del Sistema SWUSRC	14-SWUSRC-1
	Módulo del sistema SWUSRF	14-SWUSRF-1
	Módulo del Sistema SWTOOL.....	14-SWTOOL-1
15	GlueWare.....	15-1
	ggundata - Datos de pistola de aplicación de adhesivo.....	15-ggundata-1
	GlueC - Aplicación de adhesivo con un movimiento circular	15-GlueC-1
	GlueL - Aplicación de adhesivo con un movimiento lineal.....	15-GlueL-1
	Módulo del Sistema GLUSER	15-GLUSER-1
16		
17		
18	Referencia Rápida	18-1
	1 La Ventana de Movimiento	18-3
	1.1 Ventana: Movimiento	18-3
	2 La Ventana de Entradas/Salidas	18-4
	2.1 Ventana: Entradas/Salidas	18-4
	3 La Ventana de Programa	18-6
	3.1 Movimiento entre diferentes partes del programa	18-6
	3.2 Menús generales.....	18-7
	3.3 Ventana: Instrucciones de Programa	18-10
	3.4 Ventana: Rutinas del Programa	18-12
	3.5 Ventana: Datos del Programa	18-14
	3.6 Ventana: Tipos de Datos del Programa	18-16
	3.7 Ventana: Test del Programa.....	18-17
	3.8 Ventana: Módulos del Programa	18-18
	4 La Ventana de Producción.....	18-19
	4.1 Ventana: Funcionamiento en Producción	18-19
	5 El Administrador de Archivos.....	18-21
	5.1 Ventana: Administrador de Archivos	18-21
	6 La ventana de Servicio	18-23
	6.1 Menús generales.....	18-23
	6.2 Ventana Registro de Servicio.....	18-25

6.3 Ventana de Calibración de Servicio	18-26
6.4 Ventana de Comutación de Servicio	18-27
7 Los Parámetros del Sistema.....	18-28
7.1 Ventana: Parámetros del Sistema.....	18-28
8 Ventana especiales ArcWare	18-30
8.1 Ventana: Producción	18-30
19 Funciones especiales de este robot	
20 Indice, Glosario.....	20-1

Safety

INDICE

	Página
1 Otros Manuales	3
2 Cómo utilizar este Manual	3
2.1 Convenciones tipográficas.....	4
2.2 Reglas de sintaxis	5
2.3 Sintaxis formal.....	6

Introducción

Introducción

Esta es una guía de referencia que contiene una explicación detallada del lenguaje de programación así como de todos los *tipos de datos, instrucciones y funciones*. Si el usuario desea llevar a cabo una programación off-line, esta guía será particularmente útil al respecto.

Se recomienda utilizar en primer lugar la Guía del Usuario cuando se desee empezar a programar el robot, hasta haber adquirido cierta familiaridad con el sistema.

1 Otros Manuales

Antes de utilizar el robot por primera vez, se recomienda al usuario leer detenidamente el documento titulado *Funcionamiento Básico*. Ello le proporcionará las bases necesarias para el manejo del robot.

La *Guía del Usuario* proporciona las instrucciones paso a paso para la realización de diferentes tareas, como por ejemplo, para mover manualmente el robot, para programar, o para arrancar un programa dentro del funcionamiento de producción.

El *Manual de Producto* describe como se deberá instalar el robot, así como los procedimientos de mantenimiento y el diagnóstico de averías. Este manual contiene también un documento titulado *Especificación del Producto* que proporciona una visión general de las características y capacidades del robot.

2 Cómo utilizar este Manual

Para encontrar respuesta a preguntas del tipo *¿Qué instrucción debo utilizar?* o *¿Qué significa esta instrucción?*, se deberá consultar el *Capítulo 3: Resumen RAPID*. Este capítulo describe brevemente todas las instrucciones, funciones y tipos de datos agrupados de acuerdo con las listas de selección de instrucciones que se utilizan en la programación. Asimismo, contiene un resumen de la sintaxis, lo cual es particularmente útil cuando se lleva a cabo una programación off-line.

En el *Capítulo 4: Ejemplos*, se encontrarán una serie de ejemplos del programa.

El *Capítulo 5: Características Básicas* describe los detalles internos del lenguaje. El usuario normalmente no deberá leer este capítulo a menos de ser un programador experimentado.

El *Capítulo 6: Principios de Movimiento y de E/S* describe los diferentes sistemas de coordenadas del robot, su velocidad, así como otras características de movimiento utilizadas durante diferentes tipos de ejecución.

Los *Capítulos 7 a 9* describen todos los *tipos de datos, instrucciones y funciones*. Para una máxima facilidad de manejo, aparecen descritos siguiendo un orden alfabético.

Este manual ofrece una descripción de todos los datos y programas disponibles en el robot en el momento de la entrega. Además de estos datos, existe una serie de datos y programas predefinidos que son suministrados con el robot, y que vienen en un soporte informático separado, o bien directamente cargados en el robot. El

Introducción

Capítulo 10: Programas y Datos Predefinidos describe lo que ocurre cuando dichos datos ya están cargados en el robot.

En el caso en que el usuario desea realizar una programación off-line, deberá consultar el *Capítulo 11: Programación off-line* en el que se encontrarán distintas recomendaciones.

Los *Capítulos 13 a 15* contiene las diversas funcionalidades del robot cuando éste está equipado con algún software de proceso como ArcWare, GlueWare o SpotWare.

Si el usuario desea buscar información sobre la función de un comando particular, deberá referirse al *Capítulo 18: Referencia Rápida*. Este capítulo podrá utilizarse también como guía de bolsillo cuando se está trabajando con el robot.

Para facilitar la localización y la comprensión de los términos utilizados, el *Capítulo 20* contiene un *índice* y un *glosario*.

En el caso en que el robot sea entregado, o ampliado con algún tipo de funcionalidad adicional, ello aparecerá descrito en el *Capítulo 19: Funcionalidad especial de este Robot*.

2.1 Convenciones tipográficas

Los comandos situados debajo de cada una de las cinco teclas del menú en la parte superior del visualizador de la unidad de programación se encuentran escritas bajo la forma de **Menú: Comando**. Por ejemplo, para activar el comando de Impresión del menú Archivo, se deberá seleccionar **Archivo: Imprimir**.

Los nombres de las teclas de función y los nombres de las entradas de campos están escritas en letra itálica y negrita, ejemplo ***Modpos***.

Las palabras que pertenecen al lenguaje de programación utilizado, como por ejemplo los nombres de instrucción, aparecen escritas en letra itálica, por ejemplo, *MoveL*.

Los ejemplos de programas aparecen siempre visualizados de la misma forma en que aparecen en disquetes o impresora. Lo que difiere de lo que aparece en la unidad de programación se refiere a lo siguiente:

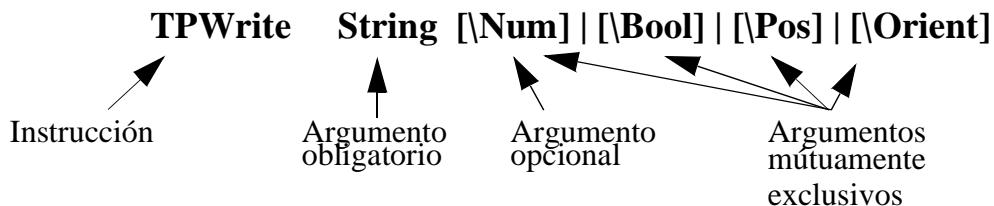
- Algunos códigos de control que son ocultos en el visualizador de la unidad de programación y que aparecen impresos, por ejemplo, los códigos que indican el principio y el final de una rutina.
- Datos y rutinas que aparecen impresos de manera formal, por ejemplo, *VAR num reg1;*

2.2 Reglas de sintaxis

Las instrucciones y las funciones aparecen descritas utilizando tanto la sintaxis simplificada como la sintaxis formal. Si el usuario desea utilizar la unidad de programación para llevar a cabo las operaciones de programación, normalmente le bastará con utilizar la sintaxis simplificada, ya que el robot automáticamente se asegura de que se está utilizando la sintaxis adecuada.

Sintaxis simplificada

Ejemplo:



- Los argumentos opcionales están marcados entre corchetes []. Estos argumentos podrán ser omitidos.
- Los argumentos que son mutuamente exclusivos, es decir, los que no pueden estar en una misma instrucción al mismo tiempo, están separados por una barra vertical |.
- Los argumentos que pueden ser repetidos un número arbitrario de veces están marcados entre llaves { }.

2.3 Sintaxis formal

Ejemplo:

```
TPWrite  
[String':=] <expression (IN) of string>  
[\'Num':= <expression (IN) of num> ] |  
[\'Bool':= <expression (IN) of bool> ] |  
[\'Pos':= <expression (IN) of pos> ] |  
[\'Orient':= <expression (IN) of orient> ]';
```

- El texto indicado entre corchetes [] puede ser omitido.
- Los argumentos que son mutuamente exclusivos, es decir, los que no pueden constar en una misma instrucción al mismo tiempo, está separados por una barra vertical |.
- Los argumentos que pueden ser repetidos un número arbitrario de veces están marcados entre llaves { }.
- Los símbolos que están escritos con la finalidad de obtener una sintaxis correcta están marcados entre comillas sencillas ' '.
- El tipo de datos del argumento (italicas) y las demás características están marcados entre ángulos <>. Véase a la descripción de los parámetros de una rutina para información más detallada.

Los elementos básicos del lenguaje así como algunas instrucciones específicas están escritas utilizando una sintaxis especial, EBNF, que está basada en las mismas normas, pero con algunas añadiduras.

Ejemplo:

```
GOTO <identifier>;  
<identifier> ::= <ident>  
              | <ID>  
<ident> ::= <letter> {<letter> | <digit> | '_'}  
              | '_'
```

- El símbolo ::= significa *se define como*.
- El texto marcado entre ángulos <> aparece definido en una línea separada.

ÍNDICE

	Página
1 A Estrutura da Linguagem	5
2 Control del Flujo del Programa.....	6
2.1 Principios de programación	6
2.2 Llamada a otra rutina.....	6
2.3 Control del programa dentro de la rutina.....	7
2.4 Paro de la ejecución del programa.....	7
2.5 Paro del ciclo actual.....	7
3 Instrucciones varias	8
3.1 Asignar un valor a un dato.....	8
3.2 Condición de Espera	8
3.3 Comentarios.....	8
3.4 Carga de módulos de programa.....	9
3.5 Funciones varias	9
3.6 Datos básicos	9
3.7 Función de conversión	9
4 Características de movimiento.....	10
4.1 Principios de programación	10
4.2 Definición de la velocidad.....	10
4.3 Definición de la aceleración	11
4.4 Definición de la gestión de la configuración	11
4.5 Definición de la carga útil	11
4.6 Definición del comportamiento del robot cerca de un punto singular	11
4.7 Desplazamiento de un programa	12
4.8 Servo suave	12
4.9 Valores de ajuste del robot	12
4.10 Zonas Mundo	13
4.11 Datos para las características de movimiento.....	13
5 Movimiento	14
5.1 Principios de programación	14
5.2 Instrucciones de posicionamiento.....	15
5.3 Búsqueda	15
5.4 Activación de salidas o interrupciones en posiciones específicas	15
5.5 Control del movimiento en caso de ocurrir un error/interrupción.....	16
5.6 Control de los ejes externos	16
5.7 Ejes independientes	17
5.8 Funciones de posición	17

Sumário RAPID

	Página
5.9 Datos de movimiento.....	18
5.10 Datos básicos para los movimientos	18
6 Señales de entrada y salida.....	19
6.1 Principios de programación.....	19
6.2 Cambio del valor de una señal	19
6.3 Lectura del valor de una señal de entrada	19
6.4 Lectura del valor de una señal de salida.....	19
6.5 Comprobación de entradas en señales de salida.....	20
6.6 Inhabilitación y habilitación de los módulos de E/S	20
6.7 Definición de las señales de entrada y salida	20
7 Comunicación.....	21
7.1 Principios de programación.....	21
7.2 Comunicación utilizando la unidad de programación	22
7.3 Lectura desde o escritura en un canal/archivo serie basado en caracteres	22
7.4 Comunicación mediante canales serie/archivos binarios	22
7.5 Datos de los canales serie	23
8 Interrupciones	24
8.1 Principios de programación.....	24
8.2 Conexión de las interrupciones a las rutinas de tratamiento de interrupción.....	24
8.3 Petición de interrupciones	25
8.4 Anulación de interrupciones.....	25
8.5 Habilitación/Inhabilitación de las interrupciones	25
8.6 Tipos de datos de las interrupciones.....	25
9 Recuperación de errores.....	26
9.1 Principios de programación	26
9.2 Creación de una situación de error desde dentro del programa.....	26
9.3 Rearranque/regreso del gestor de errores	27
9.4 Datos para la gestión de los errores	27
10 Sistema & Tempo	28
10.1 Princípios de programação	28
10.2 Utilização do relógio para temporizar um evento	28
10.3 Leitura da hora e data atuais	28
11 Matemáticas.....	29
11.1 Principios de programación.....	29
11.2 Cálculos sencillos sobre datos numéricos	29
11.3 Cálculos más avanzados	29
11.4 Funciones aritméticas	30

	Página
12 Soldadura por Puntos	31
12.1 Características de la soldadura por puntos	32
12.2 Principios de SpotWare	33
12.3 Principios de programación	34
12.4 Instrucciones de soldadura por puntos.....	34
12.5 Datos de soldadura por puntos.....	34
13 Soldadura al Arco	35
13.1 Principios de programación.....	35
13.2 Instrucciones de soldadura al arco.....	35
13.3 Datos de soldadura al arco.....	36
14 GlueWare	37
14.1 Características de aplicación de adhesivo	37
14.2 Principios de programación	37
14.3 Instrucciones de aplicación de adhesivo.....	38
14.4 Datos de aplicación de adhesivo.....	38
15 Comunicación externa del computador	39
15.1 Principios de programación	39
15.2 Envío de un mensaje controlado por el programa desde el robot al computador	39
16 Instrucciones de Servicio	40
16.1 Direccionamiento de un valor a una señal de test del robot	40
17 Funciones de cadena	41
17.1 Operaciones Básicas	41
17.2 Comparación y Búsqueda.....	41
17.3 Conversión.....	41
18 Resumen de sintaxis	42
18.1 Instrucciones	42
18.2 Funciones.....	46

Sumário RAPID

1 A Estrutura da Linguagem

O programa é constituído por um número de instruções que descrevem o trabalho do robô. Portanto, existem instruções específicas para os vários comandos, tais como, movimentar o robô, ligar uma saída, etc.

As instruções geralmente possuem um número de argumentos associados que definem o que deve ocorrer em uma instrução específica. Por exemplo, a instrução para desligar uma saída contém um argumento que define qual saída deve ser desligada; por exemplo *Reset do5*. Estes argumentos podem ser especificados das seguintes maneiras:

- como um valor numérico, por exemplo 5 ou 4.6
- como uma referência a um dado, por exemplo *reg1*
- como uma expressão, por exemplo *5+reg1*2*
- como uma chamada funcional, por exemplo *Abs(reg1)*
- como um valor “string”, por exemplo “*Produção parte A*”

Existem três tipos de rotinas – *procedimentos, funções e rotinas “trap”*.

- um procedimento é usado como um subprograma.
- uma função retorna um valor de um tipo específico e é usado como um argumento de uma instrução.
- as rotinas “trap” são um meio de responder a interrupções. Uma rotina “trap” pode ser associada a uma interrupção específica; p. ex. quando uma entrada for ligada, ela é automaticamente executada se a interrupção em particular ocorrer.

As informações podem também ser armazenadas em dados (data), por exemplo, dados de ferramenta (que contêm todas as informações sobre a ferramenta, tais como, seu TCP e peso) e dados numéricos (que podem ser usados, por exemplo, para contar o número de peças a serem processadas). Os dados são agrupados em diferentes tipos de tipos de dados (data types) que descrevem diferentes tipos de informações, tais como ferramentas, posições e cargas. Como estes dados podem ser criados e possuirem nomes arbitrários, não existe limite de seu número (exceto imposto pela memória). Estes dados podem existir tanto globalmente no programa ou localmente dentro da rotina.

Existem três tipos de dados – *constantes, variáveis e persistentes*.

- uma constante representa um valor estático e seu valor somente pode ser alterado manualmente.
- uma variável pode ter seu valor alterado durante a execução do programa.
- uma persistente pode ser descrita como uma variável “persistente”. Quando um programa é salvo, os valores de inicialização refletem os seus valores atuais.

Outras características da linguagem são:

- Parâmetros da Rotina
- Expressões lógicas e aritméticas
- Tratamento automático de erros
- Programas Modulares
- Multitarefa

2 Control del Flujo del Programa

El programa se ejecuta generalmente de forma secuencial, es decir, instrucción por instrucción. En algunas ocasiones, para poder procesar diferentes situaciones aparecidas durante la ejecución del programa son necesarias instrucciones que interrumpen esta ejecución secuencial y que llaman a otra instrucción.

2.1 Principios de programación

El flujo del programa puede ser controlado de acuerdo con cinco principios diferentes:

- Llamando a otra rutina (procedimiento) y, una vez que esta rutina ha sido ejecutada, el sistema continúa la ejecución con la siguiente instrucción de la llamada de la rutina.
- Ejecutando diferentes instrucciones según si una condición dada se cumple o no.
- Repitiendo una secuencia de instrucciones un cierto número de veces o hasta que se haya cumplido una condición dada.
- Saltando a una etiqueta dentro de la misma rutina.
- Parando la ejecución del programa.

2.2 Llamada a otra rutina

<u>Instrucción</u>	<u>Sirve para:</u>
<i>ProcCall</i>	Llamar (saltar a) a otra rutina
<i>CallByVar</i>	Llamar procedimientos con nombres específicos
<i>RETURN</i>	Regresar a la rutina original

2.3 Control del programa dentro de la rutina

<u>Instrucción</u>	<u>Sirve para:</u>
<i>Compact IF</i>	Ejecutar una instrucción únicamente si se ha cumplido una condición
<i>IF</i>	Ejecutar una secuencia de instrucciones diferentes dependiendo de si se ha cumplido o no una condición
<i>FOR</i>	Repetir una sección del programa un cierto número de veces
<i>WHILE</i>	Repetir una secuencia de instrucciones diferentes mientras se cumpla una condición dada.
<i>TEST</i>	Ejecutar diferentes instrucciones dependiendo del valor de una expresión
<i>GOTO</i>	Saltar a una etiqueta
<i>label</i>	Especificar una etiqueta (nombre de línea)

2.4 Paro de la ejecución del programa

<u>Instrucción</u>	<u>Sirve para:</u>
<i>Stop</i>	Parar la ejecución del programa
<i>EXIT</i>	Parar la ejecución del programa cuando el rearranque de un programa no es permitido.
<i>Break</i>	Parar la ejecución del programa temporalmente para el diagnóstico y solución de averías.

2.5 Paro del ciclo actual

<u>Instrucción</u>	<u>Sirve para:</u>
<i>Exit cycle</i>	Parar el ciclo actual y mover el indicador de punto de arranque del programa a la primera instrucción de la rutina principal. Cuando se selecciona el modo de ejecución <i>CONT</i> , la ejecución continuará con el siguiente ciclo de programa.

3 Instrucciones varias

Instrucciones varias sirve para realizar las siguientes operaciones

- asignar valores a datos,
- esperar cierto tiempo o esperar hasta que se haya cumplido una condición,
- introducir un comentario en el programa
- cargar módulos de programa.

3.1 Asignar un valor a un dato

Se puede asignar un valor arbitrario a los datos. El dato puede ser inicializado con un valor constante, por ejemplo 5, o actualizado, mediante una expresión aritmética, por ejemplo, *reg1+5*reg3*.

<u>Instrucción</u>	<u>Sirve para:</u>
<code>:=</code>	Asignar un valor a un dato

3.2 Condición de Espera

Se puede programar el robot para que espere durante cierto tiempo, o que espere hasta que una condición arbitraria se haya cumplido; por ejemplo, esperar hasta que se haya activado una entrada.

<u>Instrucción</u>	<u>Sirve para:</u>
<code>WaitTime</code>	Esperar cierto tiempo o esperar hasta que el robot se detenga
<code>WaitUntil</code>	Esperar hasta que se haya cumplido una condición
<code>WaitDI</code>	Esperar hasta que se haya activado una entrada digital
<code>WaitDO</code>	Esperar hasta que se haya activado una salida digital

3.3 Comentarios

Los comentarios son introducidos en el programa únicamente para aumentar su facilidad de lectura. La ejecución del programa no se ve afectada por un comentario.

<u>Instrucción</u>	<u>Sirve para:</u>
<code>comment</code>	Introducir comentarios en el programa

3.4 Carga de módulos de programa

Los módulos de programa podrán ser cargados desde la memoria de masa o ser borrados desde la memoria de programa. De esta forma programas extensos podrán ser cargados con una memoria pequeña.

<u>Instrucción</u>	<u>Sirve para:</u>
<i>Load</i>	Cargar un módulo de programa en la memoria de programa
<i>UnLoad</i>	Descargar un módulo de programa desde la memoria de programa

3.5 Funciones varias

<u>Función</u>	<u>Sirve para:</u>
<i>OpMode</i>	Leer el modo de funcionamiento que está utilizando el robot
<i>RunMode</i>	Leer el modo de ejecución del programa que está utilizando el robot
<i>Dim</i>	Obtener las dimensiones de una matriz
<i>Present</i>	Descubrir si un parámetro opcional estaba presente en el momento en que se hizo la llamada a la rutina.
<i>IsPers</i>	Comprobar si un parámetro es un dato persistente
<i>IsVar</i>	Comprobar si un parámetro es un dato variable

3.6 Datos básicos

<u>Tipos de datos</u>	<u>Sirven para definir:</u>
<i>bool</i>	Datos lógicos (con los valores verdaderos o falsos)
<i>num</i>	Valores numéricos (decimales o enteros)
<i>symnum</i>	Datos numéricos con un valor simbólico
<i>string</i>	Cadenas de caracteres
<i>switch</i>	Parámetros de rutina sin valor

3.7 Función de conversión

<u>Función</u>	<u>Sirve para:</u>
<i>StrToByte</i>	Convertir un byte en un dato de cadena con un formato de dato byte definido.
<i>ByteToStr</i>	Convertir una cadena con un formato de dato byte definido en un dato byte.

4 Características de movimiento

Algunas de las características de movimiento del robot se determinan utilizando instrucciones lógicas que se aplican a todos los movimientos:

- Velocidad máxima y corrección de la velocidad
- Aceleración
- Gestión de las diferentes configuraciones del robot
- Carga útil
- Comportamiento del robot cerca de un punto singular
- Desplazamiento del programa
- Servo suave
- Valores de ajuste

4.1 Principios de programación

Las características básicas del movimiento del robot están determinadas por los datos especificados en cada instrucción de posicionamiento. Sin embargo, algunos datos están especificados en instrucciones separadas y se aplican a todos los movimientos hasta que estos datos cambian.

Las características generales de movimiento están especificadas mediante una serie de instrucciones, pero también pueden leerse mediante las variables del sistema *C_MOTSET* o *C_PROGDISP*.

Los valores por defecto quedan activados automáticamente (ejecutando la rutina *SYS_RESET* en el módulo del sistema *BASE*):

- al arrancar el sistema,
- al cargar un programa nuevo,
- al arrancar un programa desde el principio.

4.2 Definición de la velocidad

La velocidad absoluta es programada como un argumento en la instrucción de posicionamiento. Además, la velocidad máxima y el ajuste de la velocidad (un porcentaje de la velocidad programada) puede ser definida.

Instrucción

Sirve para definir:

VelSet

La velocidad máxima y el ajuste de la velocidad

4.3 Definición de la aceleración

Cuando se manipulan piezas frágiles, por ejemplo, se podrá reducir la aceleración para una parte del programa.

<u>Instrucción</u>	<u>Sirve para definir:</u>
<i>AccSet</i>	La aceleración máxima permitida

4.4 Definición de la gestión de la configuración

La configuración del robot suele ser comprobada normalmente durante el movimiento. Si se utiliza un movimiento eje a eje, la configuración correcta será adoptada. Si se utiliza un movimiento lineal o circular, el robot se moverá siempre hacia la configuración más cercana a la programada, pero se realizará una comprobación para ver si es la misma que la programada. Pero esto podrá cambiarse si se desea.

<u>Instrucción</u>	<u>Sirve para definir:</u>
<i>ConfJ</i>	El control de la configuración activado/desactivado durante el movimiento eje a eje
<i>ConfL</i>	El control de la configuración activado/desactivado durante el movimiento lineal

4.5 Definición de la carga útil

Para obtener la mejor capacidad de producción del robot, se deberá definir la carga útil correcta.

<u>Instrucción</u>	<u>Sirve para definir:</u>
<i>GripLoad</i>	La carga útil de la pinza

4.6 Definición del comportamiento del robot cerca de un punto singular

Se podrá programar el robot para que evite los puntos singulares cambiando la orientación de la herramienta automáticamente.

<u>Instrucción</u>	<u>Sirve para definir:</u>
<i>SingArea</i>	El método de interpolación a través de los puntos singulares

4.7 Desplazamiento de un programa

Cuando una parte del programa debe ser desplazada, por ejemplo, después de una búsqueda, se añadirá un desplazamiento de programa.

<u>Instrucción</u>	<u>Sirve para:</u>
<i>PDispOn</i>	Activar el desplazamiento de programa
<i>PDispSet</i>	Activar el desplazamiento de programa especificando un valor
<i>PDispOff</i>	Desactivar un desplazamiento de programa
<i>EOffsOn</i>	Activar un offset de un eje externo
<i>EOffsSet</i>	Activar un offset de un eje externo especificando un valor
<i>EOffsOff</i>	Desactivar un offset de un eje externo
<u>Función</u>	<u>Sirve para:</u>
<i>DefDFrame</i>	Calcular un desplazamiento de programa a partir de tres posiciones
<i>DefFrame</i>	Calcular un desplazamiento de programa a partir de seis posiciones
<i>ORobT</i>	Eliminar un desplazamiento de programa a partir de una posición

4.8 Servo suave

La función servo suave puede aplicarse a uno o varios ejes del robot. Esta función proporciona al robot una gran flexibilidad y puede, por ejemplo, sustituir una herramienta.

<u>Instrucción</u>	<u>Sirve para:</u>
<i>SoftAct</i>	Activar el servo suave para uno o varios ejes
<i>SoftDeact</i>	Desactivar el servo suave

4.9 Valores de ajuste del robot

Por lo general, el robot tiene la facultad de optimizar su rendimiento; sin embargo, en algunos casos muy especiales, puede ocurrir un desajuste. Por ello, se podrá ajustar los valores de ajuste del robot a fin de conseguir siempre el máximo rendimiento.

<u>Instrucción</u>	<u>Sirve para:</u>
<i>TuneServo</i>	Ajustar los valores de ajuste del robot
<i>TuneReset</i>	Reiniciar el ajuste a valores normales
<i>PathResol</i>	Ajustar la resolución de la trayectoria geométrica

<u>Tipo de dato</u>	<u>Sirve para:</u>
<i>tunetype</i>	Representar el tipo de ajuste como una constante simbólica

4.10 Zonas Mundo

Se podrá definir hasta 10 volúmenes diferentes dentro del área de trabajo del robot. Estos volúmenes sirven para:

- Indicar que el TCP del robot constituye una parte definida del área de trabajo.
- Delimitar el área de trabajo del robot y evitar una colisión con la herramienta.
- Crear una área de trabajo común para dos robots. El área de trabajo será entonces disponible para un robot a la vez.

<u>Instrucción</u>	<u>Sirve para:</u>
<i>WZBoxDef</i> ¹	Definir una zona mundo en forma de caja rectangular
<i>WZCylDef</i> ¹	Definir una zona mundo en forma de cilindro
<i>WZSphDef</i>	Definir una zona mundo en forma de esfera
<i>WZLimSup</i> ¹	Activar la supervisión de límite de una zona mundo
<i>WZDOSet</i> ¹	Activar la zona mundo para que se activen salidas digitales
<i>WZDisable</i> ¹	Desactivar la supervisión de una zona mundo temporal
<i>WZEnable</i> ¹	Activar la supervisión de una zona mundo temporal
<i>WZFree</i> ¹	Borrar la supervisión de una zona mundo temporal
<u>Tipo de dato</u>	<u>Sirve para:</u>
<i>wztemporary</i>	Identificar una zona mundo temporal
<i>wzstationary</i>	Identificar una zona mundo estacionaria
<i>shapedata</i>	Describir la geometría de una zona mundo

4.11 Datos para las características de movimiento

<u>Tipos de dato</u>	<u>Sirve para definir:</u>
<i>motsetdata</i>	Características de movimiento excepto el desplazamiento de programa
<i>progdisp</i>	Desplazamiento de programa

1. Únicamente cuando el robot está equipado con la opción “Funciones Avanzadas”

5 Movimiento

Los movimientos del robot están programados como movimientos posición por posición, es decir, «movimiento a partir de la posición actual a una posición nueva». La trayectoria entre estas dos posiciones es entonces calculada automáticamente por el robot.

5.1 Principios de programación

Las características básicas de movimiento, tales como el tipo de trayectoria, se especifican escogiendo la instrucción de posicionamiento adecuada.

Las características de movimiento restantes se especifican definiendo los datos que son los argumentos de la instrucción:

- Datos de posición (posición final del robot y de los ejes externos)
- Datos de velocidad (velocidad deseada)
- Datos de zona (precisión de la posición)
- Datos de herramienta (por ejemplo, la posición del TCP)
- Datos de la pieza de trabajo (por ejemplo, el sistema de coordenadas utilizado)

Algunas de las características de movimiento del robot se determinan mediante instrucciones lógicas que se aplican a todos los movimientos (Véase *Características de movimiento* página 10):

- Velocidad máxima y ajuste de la velocidad
- Aceleración
- Gestión de las distintas configuraciones del robot
- Carga útil
- Comportamiento del robot cerca de los puntos singulares
- Desplazamiento del programa
- Servo suave
- Valores de ajuste

Tanto el robot como los ejes externos se posicionan utilizando las mismas instrucciones. Los ejes externos se mueven a una velocidad constante y llegan a la posición final al mismo tiempo que el robot.

5.2 Instrucciones de posicionamiento

<u>Instrucción</u>	<u>Tipo de movimiento:</u>
<i>MoveC</i>	El Punto Central de la Herramienta (TCP) sigue una trayectoria circular
<i>MoveJ</i>	Movimiento eje a eje
<i>MoveL</i>	El Punto Central de la Herramienta (TCP) sigue una trayectoria lineal.
<i>MoveAbsJ</i>	Movimiento absoluto de los ejes

5.3 Búsqueda

Durante el movimiento, el robot puede realizar una búsqueda de, por ejemplo, la posición de una pieza. La posición buscada (indicada por una señal de un sensor) es almacenada y podrá ser utilizada posteriormente para posicionar el robot o para calcular un desplazamiento de programa.

<u>Instrucción</u>	<u>Tipo de movimiento:</u>
<i>SearchC</i>	Búsqueda a lo largo de una trayectoria circular
<i>SearchL</i>	Búsqueda a lo largo de una trayectoria lineal

5.4 Activación de salidas o interrupciones en posiciones específicas

Normalmente, las instrucciones lógicas se ejecutan durante la transición entre una instrucción de posicionamiento a otra. No obstante, si se utilizan instrucciones de movimiento especiales, éstas podrán ejecutarse cuando el robot se encuentra en una posición específica.

<u>Instrucción</u>	<u>Sirve para:</u>
<i>TriggIO</i> ¹	Definir una condición de disparo para la activación de una salida en una posición determinada.
<i>TriggInt</i> ¹	Definir una condición de disparo para la ejecución de una rutina de tratamiento de interrupciones en una posición determinada.
<i>TriggEquip</i> ¹	Define una condición de disparo para activar una salida a una posición específica con la posibilidad de incluir una compensación de tiempo por el retraso en el equipo externo.
<i>TriggC</i> ¹	Hacer funcionar el robot (TCP) de forma circular con una condición de disparo activada
<i>TriggJ</i> ¹	Hacer funcionar el robot eje a eje con una condición de disparo activada
<i>TriggL</i> ¹	Hacer funcionar el robot (TCP) de forma lineal con una condición de disparo activada
<u>Tipo de dato</u>	<u>Sirve para definir:</u>
<i>trigndata</i> ¹	Condiciones de disparo

5.5 Control del movimiento en caso de ocurrir un error/interrupción

Para poder solucionar un error o atender a una interrupción, el movimiento podrá ser detenido temporalmente y luego rearrancado de nuevo.

<u>Instrucción</u>	<u>Sirve para:</u>
<i>StopMove</i>	Parar los movimientos del robot
<i>StartMove</i>	Rearrancar los movimientos del robot
<i>StorePath</i> ¹	Almacenar la última trayectoria generada
<i>RestoPath</i> ¹	Volver a generar una trayectoria almacenada con anterioridad

1. Unicamente si el robot está equipado con la opción «Funciones Avanzadas».

5.6 Control de los ejes externos

El robot y los ejes externos se suelen posicionar utilizando las mismas instrucciones. Algunas instrucciones, sin embargo, sólo afectan los movimientos de los ejes externos.

<u>Instrucción</u>	<u>Sirve para:</u>
<i>DeactUnit</i>	Desactivar una unidad mecánica externa
<i>ActUnit</i>	Activar una unidad mecánica externa

1. Unicamente si el robot está equipado con la opción «Funciones Avanzadas».

5.7 Ejes independientes

El eje 6 del robot (y 4 en el IRB 2400/4400), o un eje externo, podrán ser movidos independientemente de los demás movimientos. El área de trabajo de un eje también podrá ser reinicializada, lo cual tendrá por efecto reducir los tiempos de ciclo.

<u>Función</u>	<u>Sirven para:</u>
<i>IndAMove</i> ²	Cambiar un eje al modo independiente y mover el eje a una posición absoluta
<i>IndCMove</i> ²	Cambiar un eje al modo independiente y arrancar un movimiento continuo del eje
<i>IndDMove</i> ²	Cambiar un eje al modo independiente y mover el eje de una distancia delta
<i>IndRMove</i> ²	Cambiar un eje al modo independiente y mover el eje a una posición relativa (dentro de la revolución de ejes)
<i>IndReset</i> ²	Cambiar un eje al modo dependiente o/y reinicializar el área de trabajo
<i>IndInpos</i> ²	Comprobar si un eje independiente está en su posición
<i>IndSpeed</i> ²	Comprobar si un eje independiente ha alcanzado la velocidad programada

2. Únicamente si el robot está equipado con la opción «Movimiento Avanzado».

5.8 Funciones de posición

<u>Funciones</u>	<u>Sirven para:</u>
<i>Offs</i>	Añadir un offset a una posición del robot, expresado respecto al objeto de trabajo
<i>RelTool</i>	Añadir un offset, expresado en el sistema de coordenadas de la herramienta
<i>CPos</i>	Leer la posición utilizada (sólo <i>x</i> , <i>y</i> , <i>z</i> del robot)
<i>CRobT</i>	Leer la posición utilizada (la <i>robtarget completa</i>)
<i>CJointT</i>	Leer los ángulos utilizados de los ejes
<i>ReadMotor</i>	Leer los ángulos utilizados del motor
<i>CTool</i>	Leer el valor utilizado del dato de herramienta
<i>CWObj</i>	Leer el valor utilizado del dato del objeto
<i>ORobT</i>	Eliminar un desplazamiento de programa desde una posición
<i>MirPos</i>	Realizar una copia espejo de una posición

5.9 Datos de movimiento

Los datos de movimiento se utilizan como un argumento en las instrucciones de posicionamiento.

<u>Tipo de datos</u>	<u>Sirven para definir:</u>
<i>robtarget</i>	La posición final
<i>jointtarget</i>	La posición final de una instrucción <i>MoveAbsJ</i>
<i>speeddata</i>	La velocidad
<i>zonedata</i>	La precisión de la posición (punto de paro o punto de paso)
<i>tooldata</i>	El sistema de coordenadas de la herramienta y el peso de la herramienta
<i>wobjdata</i>	El sistema de coordenadas del objeto de trabajo

5.10 Datos básicos para los movimientos

<u>Tipo de datos</u>	<u>Sirven para definir:</u>
<i>pos</i>	Una posición (x, y, z)
<i>orient</i>	Una orientación
<i>pose</i>	Un sistema de coordenadas (posición + orientación)
<i>confdata</i>	La configuración de los ejes del robot
<i>extjoint</i>	La posición de los ejes externos
<i>robjoint</i>	La posición de los ejes del robot
<i>o_robtarget</i>	La posición original del robot cuando se utiliza <i>Limit ModPos</i>
<i>o_jointtarget</i>	La posición original del robot cuando se utiliza <i>Limit ModPos</i> para <i>MoveAbsJ</i>
<i>loaddata</i>	Una carga
<i>mecunit</i>	Una unidad mecánica externa

6 Señales de entrada y salida

El robot está normalmente equipado con una serie de señales digitales y analógicas para el usuario que pueden ser leídas y cambiadas desde dentro del programa.

6.1 Principios de programación

Los nombres de las señales se definen en los parámetros del sistema y mediante estos nombres, se podrán leer las señales desde el programa.

El valor de una señal analógica o de un grupo de señales digitales está especificado como un valor numérico.

6.2 Cambio del valor de una señal

<u>Instrucción</u>	<u>Sirve para:</u>
<i>InvertDO</i>	Invertir el valor de una señal de salida digital
<i>PulseDO</i>	Generar un pulso en una señal de salida digital
<i>Reset</i>	Reinicilizar una señal de salida digital (poner a 0)
<i>Set</i>	Activar una señal de salida digital (poner a 1)
<i>SetAO</i>	Cambiar el valor de una señal de salida analógica
<i>SetDO</i>	Cambiar el valor de una señal de salida digital (valor simbólico; por ejemplo, <i>high/low, activado/desactivado</i>)
<i>SetGO</i>	Cambiar el valor de un grupo de señales de salida digitales

6.3 Lectura del valor de una señal de entrada

Se podrá leer directamente el valor de una señal de entrada, por ejemplo, IF di1=1 THEN...

6.4 Lectura del valor de una señal de salida

<u>Función</u>	<u>Sirve para leer:</u>
<i>DOutput</i>	El valor de una señal de salida digital
<i>GOutput</i>	El valor de un grupo de señales de salida digitales
<i>AOutput</i>	El valor corriente desde una señal de salida analógica

6.5 Comprobación de entradas en señales de salida

<u>Instrucción</u>	<u>Sirve para:</u>
WaitDI	Esperar hasta que una entrada digital sea activada o reinicializada
WaitDO	Esperar hasta que una salida digital sea activada o reinicializada
<u>Función</u>	<u>Sirve para:</u>
TestDI	Comprobar si se ha activado una entrada digital

6.6 Inhabilitación y habilitación de los módulos de E/S

Los módulos de E/S son automáticamente habilitados a la puesta en marcha, y pueden ser inhabilitados durante la ejecución del programa y ser habilitados de nuevo posteriormente.

<u>Instrucción</u>	<u>Sirve para:</u>
IODisable	Inhabilitar un módulo de E/S
IOEnable	Habilitar un módulo de E/S

6.7 Definición de las señales de entrada y salida

<u>Tipo de datos</u>	<u>Sirve para definir:</u>
<i>dionum</i>	El valor simbólico de una señal digital
<i>signalai</i>	El nombre de una señal de entrada analógica*
<i>signalao</i>	El nombre de una señal de salida analógica*
<i>signaldi</i>	El nombre de una señal de entrada digital*
<i>signaldo</i>	El nombre de una señal de salida digital*
<i>signalgi</i>	El nombre de un grupo de señales de entrada digitales*
<i>signalgo</i>	El nombre de un grupo de señales de salida digitales*
<u>Instrucción</u>	<u>Sirve para:</u>
<i>AliasIO</i> ¹	Definir una señal con otro nombre similar

* Sólo podrán ser definidas utilizando los parámetros del sistema.

1. Únicamente si el robot está equipado con la opción “Developer’s Functions”

7 Comunicación

Existen cuatro formas distintas de comunicar a través de los canales serie:

- Los mensajes pueden visualizarse en la unidad de programación y el usuario puede responder a preguntas, como por ejemplo, el número de piezas que deben ser procesadas.
- Se puede escribir o leer información de tipo caracteres a partir de archivos de texto desde la memoria de masa, por ejemplo. De esta manera, se podrá almacenar por ejemplo, estadísticas de producción que podrán ser almacenadas y procesadas posteriormente en un PC. La información también podrá ser imprimida directamente en una impresora conectada al sistema robot.
- Información binaria que puede ser transferida entre el robot y un sensor, por ejemplo.
- Información binaria que puede ser transferida entre el robot y otro computador, por ejemplo, con un protocolo de enlace.

7.1 Principios de programación

La decisión de utilizar información basada en caracteres o información binaria dependerá de la manera en que el equipo, con el que el robot comunica, manipula esa información. Así por ejemplo, un archivo puede contener datos que se almacenan bajo forma de caracteres o en binario.

Si el sistema requiere una comunicación en ambos sentidos de forma simultánea, se necesitará un tipo de transmisión binaria.

Cada canal serie o archivo deberá en primer lugar ser abierto. Al realizar esta operación, el canal/archivo recibe un nombre que se utiliza posteriormente como una referencia en el momento de la lectura/escritura. La unidad de programación se podrá utilizar en cualquier momento y no necesita ser abierta.

Tanto el texto como el valor de algunos tipos de datos podrán ser impresos.

7.2 Comunicación utilizando la unidad de programación

<u>Instrucción</u>	<u>Sirve para:</u>
<i>TPErase</i>	Borrar la información contenida en el visualizador de la unidad de programación
<i>TPWrite</i>	Escribir texto en el visualizador de la unidad de programación
<i>ErrWrite</i>	Escribir texto en el visualizador de la unidad de programación y simultáneamente almacenar este mensaje en la lista de errores del programa
<i>TPReadFK</i>	Poner una etiqueta a las teclas de función y leer qué tecla ha sido pulsada
<i>TPReadNum</i>	Leer un valor numérico desde la unidad de programación
<i>TPShow</i>	Seleccionar una ventana en la unidad de programación desde programas en RAPID

7.3 Lectura desde o escritura en un canal/archivo serie basado en caracteres

<u>Instrucción</u>	<u>Sirve para:</u>
<i>Open</i> ¹	Abrir un canal/archivo para la lectura o escritura
<i>Write</i> ¹	Escribir texto en el canal/archivo
<i>Close</i> ¹	Cerrar el canal/archivo
<u>Función</u>	<u>Sirve para:</u>
<i>ReadNum</i>	Leer un valor numérico
<i>ReadStr</i> ¹	Leer una cadena de texto

7.4 Comunicación mediante canales serie/archivos binarios

<u>Instrucción</u>	<u>Sirve para:</u>
<i>Open</i> ¹	Abrir un canal/archivo serie para la transferencia binaria de datos
<i>WriteBin</i> ¹	Escribir en un canal serie/archivo binario
<i>WriteStrBin</i> ¹	Escribir una cadena en un canal serie/archivo binario
<i>Rewind</i> ¹	Colocar la posición del archivo al principio del archivo
<i>Close</i> ¹	Cerrar el canal/archivo

1. Únicamente si el robot está equipado con la opción «Funciones avanzadas».

<u>Función</u>	<u>Sirve para:</u>
<i>ReadBin</i> ¹	Leer desde un canal serie binario

7.5 Datos de los canales serie

<u>Tipo de datos</u>	<u>Sirve para definir:</u>
<i>iodev</i>	Una referencia a un canal/archivo serie, que posteriormente podrá ser utilizada para la lectura y escritura.

8 Interrupciones

Las interrupciones son utilizadas por el programa para permitirle tratar directamente cualquier evento ocurrido en el sistema, independientemente de la instrucción que se está ejecutando cuando ocurre la interrupción.

El programa es interrumpido, por ejemplo, cuando se activa una entrada específica en 1. Cuando ello ocurre, el programa utilizado se interrumpe y se ejecuta un tipo especial de rutina de tratamiento de interrupción. Una vez esto se ha ejecutado, la ejecución del programa prosigue a partir de donde se había interrumpido.

8.1 Principios de programación

Cada interrupción tiene asignada una identificación de interrupción propia, que se obtiene creando una variable (de tipo de datos *intnum*) y relacionándola con una rutina de tratamiento de interrupción.

La identificación de la interrupción (variable) se utiliza entonces para dar la orden a una interrupción, es decir, para especificar el motivo de la interrupción. El motivo puede ser uno de los siguientes acontecimientos:

- Una entrada o una salida está activada en 1 o en 0.
- Ha pasado un cierto periodo de tiempo después de que se haya dado la orden a una interrupción.
- Se ha alcanzado una posición específica.

Cuando se da la orden a una interrupción, ésta se encuentra automáticamente habilitada, pero podrá ser temporalmente inhabilitada. Ello se puede producir de dos maneras:

- Todas las interrupciones pueden ser inhabilitadas. Todas las interrupciones que ocurren durante este tiempo quedan almacenadas en una cola y son posteriormente generadas de forma automática cuando vuelvan a ser habilitadas de nuevo.
- Las interrupciones pueden ser desactivadas individualmente. El sistema no tendrá en cuenta ninguna interrupción que ocurra durante este tiempo.

8.2 Conexión de las interrupciones a las rutinas de tratamiento de interrupción

Instrucción	Sirve para:
CONNECT	Conectar una variable (identificación de la interrupción) a una rutina de tratamiento de interrupción.

8.3 Petición de interrupciones

<u>Instrucción</u>	<u>Sirve para solicitar:</u>
<i>ISignalDI</i>	Una interrupción desde una señal de entrada digital
<i>ISignalDO</i>	Una interrupción desde una señal de salida digital
<i>ITimer</i>	Una interrupción temporizada
<i>TriggInt</i> ¹	Una interrupción de posición fija (de la lista de selección de Movimiento)

8.4 Anulación de interrupciones

<u>Instrucción</u>	<u>Sirve para:</u>
<i>IDelete</i>	Anular (borrar) una interrupción

8.5 Habilitación/Inhabilitación de las interrupciones

<u>Instrucción</u>	<u>Sirve para:</u>
<i>ISleep</i>	Desactivar una interrupción individual
<i>IWatch</i>	Activar una interrupción individual
<i>IDisable</i>	Inhabilitar todas las interrupciones
<i>IEnable</i>	Habilitar todas las interrupciones

8.6 Tipos de datos de las interrupciones

<u>Tipo de dato</u>	<u>Sirve para definir:</u>
<i>intnum</i>	La identificación de una interrupción.

1. Unicamente si el robot está equipado con la opción «Funciones Avanzadas».

9 Recuperación de errores

Muchos de los errores que ocurren cuando se está ejecutando un programa pueden ser procesados en el programa, esto significa que la ejecución del programa no tiene por qué ser interrumpida. Estos errores pueden ser de dos tipos: un error detectado por el robot, como por ejemplo una división por cero, o bien un error detectado por el programa, como los errores que ocurren cuando se lee un valor incorrecto con un lector de código de barras.

9.1 Principios de programación

Cuando ocurre un error, se llama al gestor de error de la rutina (siempre y cuando haya uno). También se puede crear un error desde dentro del programa y luego saltar al gestor de errores.

Si la rutina no dispone de un gestor de errores, se realizará una llamada al gestor de errores de la rutina que ha llamado la rutina actual que se está utilizando. Si no encuentra ninguno allí, se realizará una llamada al gestor de errores de la rutina que llamó aquella rutina, y así sucesivamente, hasta alcanzar el gestor de errores interno del robot que procesará el error, generará un mensaje de error y detendrá la ejecución del programa.

En el gestor de errores, estos podrán ser tratados utilizando instrucciones normales. Los datos de sistema *ERRNO* podrán ser utilizados para determinar el tipo de error que se ha producido. Para regresar del gestor de errores existen varias formas.



En versiones más modernas, si la rutina corriente no posee ningún gestor de errores, el gestor de errores interno del robot se hace cargo inmediatamente del error. El gestor de errores interno genera un mensaje de error, detendrá la ejecución del programa y situará el indicador del programa en la instrucción defectuosa.

Por consiguiente, una buena norma a aplicar en estos casos consiste en lo siguiente: si el usuario desea llamar el gestor de error de la rutina que llamó la rutina actual (propagar el error), entonces, deberá:

- Añadir un gestor de error en la rutina corriente
- Añadir la instrucción RAISE en este gestor de error

9.2 Creación de una situación de error desde dentro del programa

Instrucción

RAISE

Sirve para:

«Crear» un error y llamar el gestor de errores

9.3 Rearranque/regreso del gestor de errores

<u>Instrucción</u>	<u>Sirve para:</u>
<i>EXIT</i>	Detener la ejecución del programa en el caso de ocurrir un error muy grave
<i>RAISE</i>	Llamada del gestor de errores de la rutina que ha llamado la rutina actual utilizada
<i>RETRY</i>	Volver a ejecutar la instrucción que ha originado el error
<i>TRYNEXT</i>	Ejecutar la instrucción siguiente a la que ha provocado el error
<i>RETURN</i>	Regresar a la rutina que ha llamado la rutina actual utilizada

9.4 Datos para la gestión de los errores

<u>Tipo de dato</u>	<u>Sirve para definir:</u>
<i>errnum</i>	El motivo que ha originado el error

10 Sistema & Tempo

As instruções do sistema e de tempo possibilitam ao usuário medir, inspecionar e gravar o tempo.

10.1 Princípios de programação

As instruções do relógio (clock) possibilitam ao usuário a utilização do relógio para funcionar como um “monitorador” para paradas. Deste modo, o programa do robô pode ser usado para temporizar qualquer evento desejado.

A hora e a data atuais podem ser recuperadas em texto (string). Este pode ser mostrado ao operador na unidade de programação ou ser usado em arquivos de registro.

Também é possível recuperar componentes da hora atual do sistema como um valor numérico. Isto possibilita ao programa do robô a executar uma ação em uma certa hora ou em um certo dia da semana.

10.2 Utilização do relógio para temporizar um evento

<u>Instrução</u>	<u>Usada para:</u>
<i>ClkReset</i>	Resetar um relógio usado para temporização
<i>ClkStart</i>	Iniciar um relógio usado para temporização
<i>ClkStop</i>	Parar um relógio usado para temporização
<u>Função</u>	<u>Usada para:</u>
<i>ClkRead</i>	Ler um relógio usado para temporização
<u>Tipo de dado</u>	<u>Usado para:</u>
<i>clock</i>	Temporização – armazena uma medição em segundos

10.3 Leitura da hora e data atuais

<u>Função</u>	<u>Usada para:</u>
<i>CDate</i>	Ler a data atual como um texto (string)
<i>CTime</i>	Ler a hora atual como um texto (string)
<i>GetTime</i>	Ler a hora atual como um valor numérico

11 Matemáticas

Las instrucciones y las funciones matemáticas sirven para calcular y cambiar el valor de los datos.

11.1 Principios de programación

Los cálculos suelen realizarse utilizando la instrucción de asignación, por ejemplo, $reg1 := reg2 + reg3 / 5$. Existen también algunas instrucciones para llevar a cabo operaciones sencillas, como la puesta a cero de una variable numérica.

11.2 Cálculos sencillos sobre datos numéricos

<u>Instrucción</u>	<u>Sirve para:</u>
<i>Clear</i>	Poner a cero un valor
<i>Add</i>	Sumar o restar un valor
<i>Incr</i>	Incrementar en 1
<i>Decr</i>	Disminuir en 1

11.3 Cálculos más avanzados

<u>Instrucción</u>	<u>Sirve para:</u>
$:=$	Llevar a cabo cálculos con cualquier tipo de datos

11.4 Funciones aritméticas

<u>Función</u>	<u>Sirve para:</u>
<i>Abs</i>	Calcular el valor absoluto
<i>Round</i>	Redondear un valor numérico
<i>Trunc</i>	Truncar un valor numérico
<i>Sqrt</i>	Calcular la raíz cuadrada
<i>Exp</i>	Calcular el valor exponencial con la base “e”
<i>Pow</i>	Calcular el valor exponencial con una base arbitraria
<i>ACos</i>	Calcular el valor del arco coseno
<i>ASin</i>	Calcular el valor del arco seno
<i>ATan</i>	Calcular el valor del arco tangente incluido entre [-90,90]
<i>ATan2</i>	Calcular el valor del arco tangente incluido entre [-180,180]
<i>Cos</i>	Calcular el valor del coseno
<i>Sin</i>	Calcular el valor del seno
<i>Tan</i>	Calcular el valor de la tangente
<i>EulerZYX</i>	Calcular los ángulos Euler desde una orientación
<i>OrientZYX</i>	Calcular la orientación desde los ángulos Euler
<i>PoseInv</i>	Invertir una posición
<i>PoseMult</i>	Multiplicar una posición
<i>PoseVect</i>	Multiplicar una posición y un vector

12 Soldadura por Puntos

El paquete de software SpotWare contiene utilidades para las aplicaciones de soldadura por puntos que estén equipadas con un temporizador de soldadura y con una pinza de soldadura.

La aplicación SpotWare ofrece un posicionamiento rápido y preciso combinando la manipulación de la pinza, el arranque del proceso y la supervisión de un temporizador externo de soldadura.

La comunicación con el equipo de soldadura se realiza mediante las entradas y salidas digitales. El sistema acepta algunos interfaces serie de temporizador de soldadura como: Bosch PSS5000, NADEX, ABB Timer. Véase el documento separado.

Deberá tenerse en cuenta que SpotWare es un paquete que puede ser ampliamente personalizado. Las rutinas del usuario sirven para ser adaptadas a la situación del entorno.

12.1 Características de la soldadura por puntos

El paquete SpotWare ofrece las siguientes características:

- Posicionamiento rápido y preciso
- Manipulación de una pinza con dos carreras
- Pinza sencilla/doble
- Semicierre de la pinza
- Supervisión definida por el usuario del equipo periférico antes del arranque de la soldadura
- Supervisión definida por el usuario del equipo periférico después de la soldadura
- Supervisión y apertura/cierre de la pinza definidas por el usuario
- Determinación de la presión definida por el usuario
- Cálculo del tiempo de semicierre definido por el usuario
- Monitorización del temporizador externo de soldadura
- Recuperación de errores de soldadura mediante una resoldadura automática
- Regreso a la posición de soldadura por puntos
- Contador de puntos de soldadura
- Liberación del movimiento después de una soldadura en función del tiempo o de una señal
- Rearranque rápido después de una soldadura
- Rutinas de servicio definidas por el usuario
- Preactivación y comprobación de la presión de la pinza
- Soldadura de simulación
- Ejecución a la inversa con control de la pinza
- Interfaces serie y paralelos del temporizador de soldadura
- Acepta temporizadores de inicio del arco y temporizadores de programa
- Información de los datos corrientes de SpotL
- Información de identidad de puntos de soldadura: el nombre del parámetro de los datos de soldadura corriente (formato de cadena)
- Transferencia de identidad de puntos de soldadura al temporizador de soldadura serial BOSCH PSS 5000
- Supervisión autónoma definida por el usuario, como por ejemplo la señal de corriente de soldadura controlada por estado y el arranque de la refrigeración del agua. Nota: Esta característica requiere la opción Multitarea
- Soldadura manual, apertura y cierre de la pinza iniciados por entrada digital
- Posibilidad de arranque del proceso de soldadura sin tener en cuenta el evento de posición
- Recuperación de error opcional definido por el usuario

12.2 Principios de SpotWare

La aplicación SpotWare está basada en una gestión separada del movimiento, de la soldadura por puntos y, si la opción Multitarea está instalada, de la supervisión continua. En su camino hacia la posición programada, la tarea de movimiento disparará acciones dentro de la tarea de soldadura por puntos.

Los eventos asociados a posición son activados por señales digitales virtuales.

Las tareas utilizan tanto sus propias variables internas encapsuladas como datos persistentes que son completamente transparentes para todas las tareas.

Para entradas bien definidas, las llamadas a rutinas del usuario ofrecen adaptaciones al entorno de la planta. Hay una serie de parámetros predefinidos que están también disponibles para adaptar el comportamiento de la instrucción SpotL.

Un paro de programa tendrá por efecto únicamente el paro de la ejecución de la tarea de movimiento. El proceso y la supervisión llevan a cabo sus tareas hasta que alcanzan un paro de proceso bien definido. Entre otras cosas, ello provocará la apertura de la pinza después de que se haya terminado una soldadura incluso después de haber ocurrido el paro del programa.

La apertura y el cierre de la pinza se ejecutan siempre mediante rutinas RAPID incluso si se han activado manualmente desde la ventana de E/S de la unidad de programación. Estas rutinas de la pinza podrán ser cambiadas, desde las simples funciones de activado/desactivado, hasta las funciones más complejas como el control analógico de la pinza y pueden contener una supervisión adicional de la pinza.

Dado que las tareas de proceso y de supervisión actúan sobre eventos asociados a posición de E/S, ejecutarán el disparo que ha sido enviado por el movimiento (SpotL) o que ha sido manualmente activado (unidad de programación o de forma externa). Esto ofrece la posibilidad de realizar una soldadura independiente en cualquier sitio sin tener que programar una posición nueva.

También se podrá definir nuevos eventos de supervisión y conectarlos a eventos asociados a posiciones de señal digital. Por defecto, el sistema tiene incorporado una señal de potencia de soldadura en función del estado y un control de la señal de refrigeración del agua.

Equipo:

- Un temporizador de soldadura para la monitorización con un interface estándar paralelo (serie). El temporizador de soldadura puede ser de dos tipos: disparado por señal de arranque o por programa.
- Cualquier tipo de cierre de la pinza sencilla/doble y de control de apertura de la pinza.
- Cualquier tipo de presión predeterminada.
- Las acciones que controla SpotL son independientes del equipo de soldadura por puntos, como contactores, etc. (Nota: se requiere la opción Multitarea).

12.3 Principios de programación

Tanto el movimiento lineal del robot como el control del proceso de la soldadura por puntos están contenidos en una sola instrucción, *SpotL*.

El proceso de la soldadura por puntos está especificado por:

- Datos de soldadura por puntos (Spotdata): datos del proceso de la soldadura por puntos
- Datos de la pinza (Gundata): datos de la pinza de soldadura por puntos
- El módulo del sistema SWUSRF y SWUSRC: rutinas en RAPID y datos generales para funciones de personalización del sistema. Ver el apartado *Datos y Programas predefinidos - ProcessWare*.
- Parámetros del sistema: la configuración de las E/S. Véase la Guía del Usuario *Parámetros del Sistema*.

12.4 Instrucciones de soldadura por puntos

<u>Instrucción</u>	<u>Sirve para:</u>
<i>SpotL</i>	Controlar el movimiento, la apertura/cierre de la pinza y el proceso de soldadura. Mover el TCP a lo largo de una trayectoria lineal y realizar una soldadura por puntos en la posición final.

12.5 Datos de soldadura por puntos

<u>Tipos de datos</u>	<u>Sirve para definir:</u>
<i>spotdata</i>	El control del proceso de la soldadura por puntos
<i>gundata</i>	La pinza de soldadura por puntos

13 Soldadura al Arco

El paquete de software Arcware permite llevar a cabo una gran cantidad de funciones de soldadura. Así, por ejemplo, el relleno del cráter y el arranque por oscilación, son funciones que pueden ser programadas. Mediante la utilización de Arcware, el proceso completo de soldadura será entonces controlado y monitorizado por el robot a través de diferentes entradas y salidas digitales y analógicas.

13.1 Principios de programación

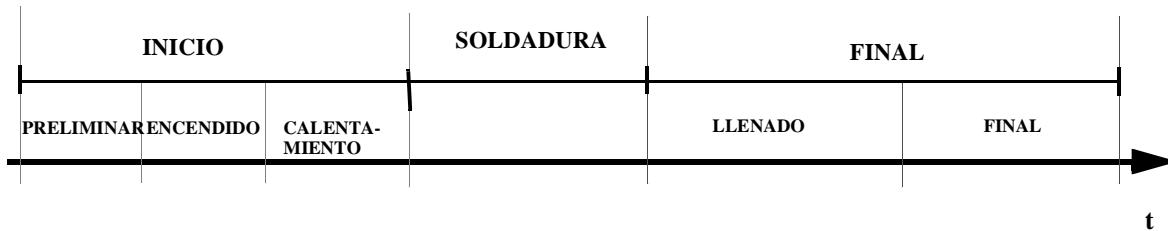
Las mismas instrucciones serán utilizadas tanto para el control de los movimientos del robot como para el proceso de soldadura correspondiente. Las instrucciones de soldadura al arco indican los datos de soldadura y los datos iniciales y finales que se utilizan en la soldadura correspondiente.

Las características de la soldadura en la fase principal de soldadura están definidas en los datos de soldadura. Las fases de inicio y final están definidas en los datos iniciales y finales de soldadura.

Las oscilaciones están definidas en los datos de oscilación, que también forman parte de las instrucciones de soldadura al arco.

Ciertas funciones, como el arranque por oscilación, se encuentran definidas en los parámetros del sistema.

El proceso de soldadura está dividido en las siguientes fases:



13.2 Instrucciones de soldadura al arco

<u>Instrucción</u>	<u>Tipo de movimiento:</u>
<i>ArcC</i>	Movimiento del TCP a lo largo de una trayectoria circular
<i>ArcL</i>	Movimiento del TCP a lo largo de una trayectoria lineal

13.3 Datos de soldadura al arco

<u>Tipos de datos</u>	<u>Sirven para definir:</u>
<i>welldata</i>	La fase de soldadura
<i>seamdata</i>	La fase de inicio y final de una soldadura
<i>weavedata</i>	Las características de oscilación
<i>trackdata</i>	Seguimiento de la costura mediante el interface serie de sensor

14 GlueWare

El paquete GlueWare proporciona las facilidades para las funciones de aplicación de adhesivo que estén equipadas con una o dos pistolas de aplicación de adhesivo.

La aplicación GlueWare proporciona un posicionamiento rápido y preciso combinado con una manipulación de la pistola, y el arranque y paro del proceso.

La comunicación con el equipo de aplicación de adhesivo se lleva a cabo mediante salidas digitales y analógicas.

14.1 Características de aplicación de adhesivo

El paquete GlueWare contiene las siguientes características:

- Posicionamiento rápido y preciso
- Gestión de pistolas de act./desact. y de pistolas proporcionales
- El sistema es capaz de gestionar dos pistolas diferentes en un mismo programa, y cada pistola está controlada por una señal digital (on/off) y dos señales analógicas (flujos).
- Semiapertura y semicerre de la pistola respectivamente
- Aplicación simulada de adhesivo

14.2 Principios de programación

Tanto el movimiento del robot como el control del proceso de la aplicación de adhesivo están incluidos en una instrucción, *GlueL* y *GlueC* respectivamente.

El proceso de aplicación de adhesivo está especificado por:

- Gundata: Datos de la pistola de aplicación de adhesivo. Véase *Tipos de datos - ggundata*.
- El módulo del sistema GLUSER: rutinas RAPID y datos globales para la personalización del sistema. Véase *Datos y Programas predefinidos - Módulo del Sistema GLUSER*.
- Parámetros del sistema: la configuración de E/S. Véase *Parámetros del Sistema - Aplicación de adhesivo*

14.3 Instrucciones de aplicación de adhesivo

<u>Instrucción</u>	<u>Sirve para:</u>
<i>GlueL</i>	Mover el TCP a lo largo de una trayectoria lineal y llevar a cabo la aplicación de adhesivo con los datos especificados.
<i>GlueC</i>	Mover el TCP a lo largo de una trayectoria circular y llevar a cabo la aplicación de adhesivo con los datos especificados.

14.4 Datos de aplicación de adhesivo

<u>Tipo de datos</u>	<u>Sirven para definir:</u>
<i>ggundata</i>	La pistola de aplicación de adhesivo utilizada

15 Comunicación externa del computador

El robot puede estar controlado a partir de un computador superior. En este caso, se utilizará un protocolo especial de comunicaciones para la transferencia de información.

15.1 Principios de programación

Dado que se usa un protocolo de comunicaciones común para la transferencia de información a partir del robot hacia el computador superior y viceversa, el robot y el computador pueden comunicar entre ellos sin necesidad de una programación. El computador puede, por ejemplo, cambiar los valores en los datos del programa sin que el usuario tenga que realizar ningún tipo de programación (excepto para la definición de estos datos). La programación es necesaria únicamente cuando la información controlada por el programa tiene que ser enviada desde el robot al computador superior.

15.2 Envío de un mensaje controlado por el programa desde el robot al computador

<u>Instrucción</u>	<u>Sirve para:</u>
<i>SCWrite</i> ¹	Enviar un mensaje al computador superior

1. Solamente si el robot está equipado con la opción «Enlace Serie RAP».

16 Instrucciones de Servicio

El sistema dispone de una serie de instrucciones que sirven para hacer el test del sistema robot. Véase el capítulo referente a las Herramientas para la detección de averías del Manual de Producto para más información.

16.1 Direccionamiento de un valor a una señal de test del robot

Se podrá dirigir una señal de referencia, como por ejemplo la velocidad de un motor, hacia una señal de salida analógica situada en la tarjeta del bus del robot.

<u>Instrucción</u>	<u>Sirve para:</u>
<i>TestSign</i>	Definir y activar una señal de test
<u>Tipo de dato</u>	<u>Sirve para definir:</u>
<i>testsignal</i>	El tipo de la señal de test

17 Funciones de cadena

Las funciones de cadena se utilizan para operaciones realizadas con cadenas como por ejemplo, la copia, la concatenación, la comparación, la búsqueda, la conversión, etc.

17.1 Operaciones Básicas

<u>Tipo de datos</u>	<u>Sirve para definir:</u>
<i>string</i>	Cadena. Constantes predefinidas STR_DIGIT, STR_UPPER, STR_LOWER y STR_WHITE
<u>Instrucción/Operador</u>	<u>Sirve para:</u>
<code>:=</code>	Asignar un valor (copia de una cadena)
<code>+</code>	Concatenación de cadena
<u>Función</u>	<u>Sirve para:</u>
<i>StrLen</i>	Tomar la longitud de la cadena
<i>StrPart</i>	Tomar parte de la cadena

17.2 Comparación y Búsqueda

<u>Operador</u>	<u>Sirve para:</u>
<code>=</code>	Comprobar si es igual a
<code><></code>	Comprobar si no es igual a
<u>Función</u>	<u>Sirve para:</u>
<i>StrMemb</i>	Comprobar si el carácter forma parte de un conjunto
<i>StrFind</i>	Buscar un carácter en una cadena
<i>StrMatch</i>	Buscar una estructura en una cadena
<i>StrOrder</i>	Comprobar si las cadenas están ordenadas

17.3 Conversión

<u>Función</u>	<u>Sirve para:</u>
<i>NumToStr</i>	Convertir un valor numérico en cadena
<i>ValToStr</i>	Convertir un valor en una cadena
<i>StrToVal</i>	Convertir una cadena en un valor
<i>StrMap</i>	Realizar un mapa de la cadena

18 Resumen de sintaxis

18.1 Instrucciones

Data := Value

AccSet Acc Ramp

ActUnit MecUnit

Add Name AddValue

Break

CallBy Var Name Number

Clear Name

ClkReset Clock

ClkStart Clock

ClkStop Clock

Close IODevice

! Comment

ConfJ [\On] | [\Off]

ConfL [\On] | [\Off]

CONNECT Interrupt **WITH** Trap routine

CorrCon Descr

CorrDiscon Descr

CorrWrite Descr Data

CorrClear

DeactUnit MecUnit

Decr Name

EOffsSet EAxOffs

ErrWrite [\W] Header Reason [\RL2] [\RL3] [\RL4]

Exit

ExitCycle

FOR Loop counter **FROM** Start value **TO** End value
[**STEP** Step value] **DO** ... **ENDFOR**

GOTO Label

GripLoad Load

IDelete Interrupt

IF Condition ...

IF Condition **THEN** ...
 {**ELSEIF** Condition **THEN** ...}
 [ELSE ...]

ENDIF

Incr Name

IndAMove MecUnit Axis [\ToAbsPos] | [\ToAbsNum] Speed
[\Ramp]

IndCMove MecUnit Axis Speed [\Ramp]

IndDMove MecUnit Axis Delta Speed [\Ramp]

IndReset MecUnit Axis [\RefPos] | [\RefNum] | [\Short] | [\Fwd] |
[\Bwd] | [\Old]

IndRMove MecUnit Axis [\ToRelPos] | [\ToRelNum] | [\Short] |
[\Fwd] | [\Bwd] | Speed [\Ramp]

InvertDO Signal

IODisable UnitName MaxTime

IOEnable UnitName MaxTime

ISignalDI [\Single] Signal TriggValue Interrupt

ISignalDO [\Single] Signal TriggValue Interrupt

ISleep Interrupt

ITimer [\Single] Time Interrupt

IVarValue VarNo Value, Interrupt

IWatch Interrupt

Label:

MoveAbsJ [\Conc] ToJointPos Speed [\V] | [\T] Zone [\Z]
Tool [\WObj]

MoveC [\Conc] CirPoint ToPoint Speed [\V] | [\T] Zone [\Z]
Tool [\WObj]

MoveJ [\Conc] ToPoint Speed [\V] | [\T] Zone [\Z] Tool
[\WObj]

MoveL [\Conc] ToPoint Speed [\V] | [\T] Zone [\Z] Tool
[\WObj]

Open Object [\File] IODevice [\Read] | [\Write] | [\Append] | [\Bin]

PathResol Value

PDispOn [\Rot] [\ExeP] ProgPoint Tool [\WObj]

PDispSet DispFrame

Procedure { Argument }

PulseDO [\PLength] Signal

RAISE [Error no]

Reset Signal

RETURN [Return value]

Rewind IODevice

SearchC [\Stop] | [\PStop] | [\Sup] Signal SearchPoint CirPoint
ToPoint Speed [\V] | [\T] Tool [\WObj]

SearchL [\Stop] | [\PStop] | [\Sup] Signal SearchPoint ToPoint
Speed [\V] | [\T] Tool [\WObj]

Set Signal

SetAO Signal Value

SetDO [\SDelay] Signal Value

SetGO Signal Value

SingArea [\Wrist] | [\Arm] | [\Off]

SoftAct Axis Softness [\Ramp]

Stop [\NoRegain]

TEST Test data {CASE Test value {, Test value} : ...}
[DEFAULT: ...] ENDTEST

TPReadFK Answer String FK1 FK2 FK3 FK4 FK5 [\MaxTime]
[\DIBreak] [\BreakFlag]

TPReadNum Answer String [\MaxTime] [\DIBreak] [\BreakFlag]

TPWrite String [\Num] | [\Bool] | [\Pos] | [\Orient]

TriggC CirPoint ToPoint Speed [\T] Trigg_1 [\T2] [\T3]
[\T4] Zone Tool [\WObj]

TriggInt TriggData Distance [\Start] | [\Time] Interrupt

TriggIO TriggData Distance [\Start] | [\Time] [\DOp] | [\GOp] |
[\AOp] SetValue [\DODelay] | [\AORamp]

TriggJ ToPoint Speed [\T] Trigg_1 [\T2] [\T3] [\T4]
Zone Tool [\WObj]

TriggL ToPoint Speed [\T] Trigg_1 [\T2] [\T3] [\T4]
Zone Tool [\WObj]

TuneServo MecUnit Axis TuneValue

TuneServo MecUnit Axis TuneValue [\Type]

Unload FilePath [\File]

VelSet Override Max

WaitDI Signal Value [\MaxTime] [\TimeFlag]

WaitDO Signal Value [\MaxTime] [\TimeFlag]

WaitTime [\InPos] Time

WaitUntil [\InPos] Cond [\MaxTime] [\TimeFlag]

WHILE Condition DO ... ENDWHILE

Write IODevice String [\Num] | [\Bool] | [\Pos] | [\Orient]
[NoNewLine]

WriteBin IODevice Buffer NChar

WriteStrBin IODevice Str

WZBoxDef [|Inside] | [|Outside] Shape LowPoint HighPoint 1

WZCylDef [|Inside] | [|Outside] Shape CentrePoint Radius Height

WZDisable WorldZone

WZDOSet [|Temp] | [|Stat] WorldZone [|Inside] | [|Before] Shape
Signal SetValue

WZEnable WorldZone

WZFree WorldZone

WZLimSup [|Temp] | [|Stat] WorldZone Shape

WZSphDef [|Inside] | [|Outside] Shape CentrePoint Radius

18.2 Funciones

Abs (Input)

ACos (Value)

AOutput (Signal)

ArgName (Parameter)

ASin (Value)

ATan (Value)

ATan2 (Y X)

ByteToStr (ByteData [|Hex] | [|Okt] | [|Bin] | [|Char])

ClkRead (Clock)

CorrRead

Cos (Angle)

CPos ([Tool] [\WObj])

CRobT ([Tool] [\WObj])

DefDFrame (OldP1 OldP2 OldP3 NewP1 NewP2 NewP3)

DefFrame (NewP1 NewP2 NewP3 [\Origin])

Dim (ArrPar DimNo)

DOoutput (Signal)

EulerZYX ([\X] | [\Y] | [\Z] Rotation)

Exp (Exponent)

GOutput (Signal)

GetTime ([\WDay] | [\Hour] | [\Min] | [\Sec])

IndInpos MecUnit Axis

IndSpeed MecUnit Axis [\InSpeed] | [\ZeroSpeed]

IsPers (DatObj)

IsVar (DatObj)

MirPos (Point MirPlane [\WObj] [\MirY])

NumToStr (Val Dec [\Exp])

Offs (Point XOffset YOffset ZOffset)

OrientZYX (ZAngle YAngle XAngle)

ORobT (OrgPoint [\InPDisp] | [\InEOffs])

PoseInv (Pose)

PoseMult (Pose1 Pose2)

PoseVect (Pose Pos)

Pow (Base Exponent)

Present (OptPar)

ReadBin (IODevice [\Time])

ReadMotor [**MecUnit**] **Axis**
ReadNum (**IODevice** [**Time**])
ReadStr (**IODevice** [**Time**])
RelTool (**Point** **Dx** **Dy** **Dz** [**Rx**] [**Ry**] [**Rz**])
Round (**Val** [**Dec**])
Sin (**Angle**)
Sqrt (**Value**)
StrFind (**Str** **ChPos** **Set** [NotInSet])
StrLen (**Str**)
StrMap (**Str** **FromMap** **ToMap**)
StrMatch (**Str** **ChPos** **Pattern**)
StrMemb (**Str** **ChPos** **Set**)
StrOrder (**Str1** **Str2** **Order**)
StrPart (**Str** **ChPos** **Len**)
StrToByte (**ConStr** [**Hex**] | [**Okt**] | [**Bin**] | [**Char**])
StrToVal (**Str** **Val**)
Tan (**Angle**)
TestDI (**Signal**)
Trunc (**Val** [**Dec**])
ValToStr (**Val**)

INDICE

	Página
1 Elementos Básicos	3
1.1 Identificadores	3
1.2 Espacios y caracteres de fin de línea	4
1.3 Valores numéricos	4
1.4 Valores lógicos	4
1.5 Valores de cadena	4
1.6 Comentarios	5
1.7 Comodines	5
1.8 Encabezado de archivo	6
1.9 Sintaxis	6
2 Módulos.....	8
2.1 Módulos del programa.....	8
2.2 Módulos del sistema	9
2.3 Declaraciones de los módulos	9
2.4 Sintaxis	10
3 Rutinas	11
3.1 Alcance de las rutinas	11
3.2 Parámetros	12
3.3 Final de una rutina	13
3.4 Declaraciones de rutina	13
3.5 Llamada de procedimiento	14
3.6 Sintaxis	15
4 Tipos de Datos	18
4.1 Tipos de datos sin valor	18
4.2 Tipos de datos iguales a (equivalente a)	18
4.3 Sintaxis	19
5 Datos	20
5.1 Alcance de los datos	20
5.2 Declaración de un dato variable	21
5.3 Declaración de un dato persistente	22
5.4 Declaración de un dato constante	22
5.5 Datos de inicio	22
5.6 Sintaxis	23
6 Instrucciones	25
6.1 Sintaxis	25
7 Expresiones	26

Características Básicas RAPID

Página

7.1 Expresiones aritméticas	26
7.2 Expresiones lógicas	27
7.3 Expresiones de cadena.....	27
7.4 Utilización de datos en las expresiones.....	28
7.5 Utilización de agregados en las expresiones	29
7.6 Utilización de llamadas a función en las expresiones	29
7.7 Prioridad entre los operadores.....	30
7.8 Sintaxis	31
8 Recuperación de Errores.....	34
8.1 Gestores de error	34
9 Interrupciones	36
9.1 Procesamiento de las interrupciones.....	36
9.2 Rutinas de tratamiento de las interrupciones	37
10 Ejecución hacia atrás	39
10.1 Gestores de ejecución hacia atrás	39
10.2 Limitación de instrucciones de movimiento en el gestor de ejecución hacia atrás	40
11 Multitareas.....	42
11.1 Sincronización de las tareas.....	43
11.2 Comunicación entre tareas.....	45
11.3 Tipo de tarea	46
11.4 Prioridades	46
11.5 Tamaño de las tareas.....	47
11.6 Recomendación importante	48

1 Elementos Básicos

1.1 Identificadores

Los identificadores sirven para nombrar los módulos, las rutinas, los datos y las etiquetas;

por ejemplo:

```
MODULE nombre_módulo
PROC nombre_rutina()
VAR pos nombre_dato;
nombre_etiqueta :
```

El primer carácter de un identificador deberá ser siempre una letra. Los demás caracteres podrán ser letras, cifras o subrayado “_”.

La longitud máxima de cualquier identificador es de 16 caracteres, y cada uno de estos caracteres es significante. Los identificadores que son idénticos excepto que están escritos en letras mayúsculas o viceversa, serán considerados como un mismo identificador.

Palabras reservadas

Las palabras que se listan a continuación son palabras reservadas. Esto significa que tienen un significado particular en lenguaje RAPID y por lo tanto no deberán utilizarse como identificadores.

Asimismo, hay una serie de nombres predefinidos para los tipos de datos, los datos del sistema, instrucciones y funciones, que no deberán utilizarse como identificadores. Véase los capítulos 7, 8, 9, 10, 13, 14 y 15 de este manual.

ALIAS	AND	BACKWARD	CASE
CONNECT	CONST	DEFAULT	DIV
DO	ELSE	ELSEIF	ENDFOR
ENDFUNC	ENDIF	ENDMODULE	ENDPROC
ENDRECORD	ENDTEST	ENDTRAP	ENDWHILE
ERROR	EXIT	FALSE	FOR
FROM	FUNC	GOTO	IF
INOUT	LOCAL	MOD	MODULE
NOSTEPIN	NOT	NOVIEW	OR
PERS	PROC	RAISE	READONLY
RECORD	RETRY	RETURN	STEP
SYSMODULE	TEST	THEN	TO
TRAP	TRUE	TRYNEXT	VAR
VIEWONLY	WHILE	WITH	XOR

1.2 Espacios y caracteres de fin de línea

El lenguaje de programación RAPID es un lenguaje sin formatos, lo que significa que los espacios podrán utilizarse en cualquier parte excepto en:

- los identificadores
- las palabras reservadas
- los valores numéricos
- los comodines.

Los caracteres de fin de línea, los tabuladores y los caracteres de fin de página pueden usarse en todos los lugares donde se puede usar un espacio, excepto dentro de los comentarios.

Los identificadores, las palabras reservadas y los valores numéricos deberán estar separados entre sí por un espacio, un carácter de fin de línea, un tabulador, o un carácter de fin de línea.

Los espacios innecesarios y los caracteres de fin de línea serán automáticamente borrados de un programa cargado en la memoria de programa. Por ello, deberá recordarse que los programas cargados a partir de un disquete y luego almacenados de nuevo podrán no ser iguales.

1.3 Valores numéricos

Un valor numérico puede expresarse de diferentes formas:

- como un número entero, por ejemplo: 3, -100, 3E2
- un número decimal, por ejemplo: 3,5, -0,345, -245E-2

El valor deberá situarse dentro de los límites especificados por el formato estándar ANSI IEEE 754-1985 de coma flotante (simple precisión).

1.4 Valores lógicos

Un valor lógico podrá expresarse como TRUE (verdadero) o FALSE (falso).

1.5 Valores de cadena

Un valor de cadena es una secuencia de caracteres (ISO 8859-1) y de caracteres de control (caracteres que no están incluidos en ISO 8859-1 en la gama de códigos numéricos entre 0-255). Se podrán incluir códigos de caracteres, pudiendo contener también caracteres que no se pueden imprimir (datos binarios). La longitud máxima de la cadena es de 80 caracteres.

Ejemplo: "Esto es una cadena"
 "Esta cadena se termina con el código de control BEL \07"

En el caso en que se incluya una barra invertida \ (que indica un código de carácter) o unas comillas de abrir y cerrar, ambos deberán escribirse dos veces.

Ejemplo: "Esta cadena contiene un carácter "" comillas"
 "Esta cadena contiene un carácter \\ barra invertida"

1.6 Comentarios

Los comentarios sirven para facilitar la comprensión del programa. Por lo tanto, se debe saber que no afectan de ningún modo el funcionamiento del programa.

Un comentario empieza siempre con un signo de exclamación “!” y acaba con un carácter de fin de línea. Siempre ocupa una línea entera y no puede estar nunca entre dos módulos;

por ejemplo:

```
! comentario
IF reg1 > 5 THEN
    ! comentario
    reg2 := 0;
ENDIF
```

1.7 Comodines

Los comodines se utilizarán para representar de forma temporal ciertas partes del programa que todavía no han sido definidas. Un programa que contiene comodines será correcto desde el punto de vista sintáctico y podrá ser cargado en la memoria del programa.

<u>Comodín</u>	<u>Representa:</u>
<TDN>	definición del tipo de dato
<DDN>	una declaración de datos
<RDN>	una declaración de rutina
<PAR>	un parámetro alternativo formal opcional
<ALT>	un parámetro formal opcional
<DIM>	una definición de la dimensión formal de la matriz
<SMT>	una instrucción
<VAR>	una referencia (variable, persistente o parámetro) a datos de objeto
<EIT>	la cláusula <i>else</i> de una instrucción <i>if</i>
<CSE>	la cláusula <i>case</i> de una instrucción <i>test</i>
<EXP>	una expresión
<ARG>	un argumento de llamada de procedimiento
<ID>	un identificador

1.8 Encabezado de archivo

Un archivo de un programa empieza siempre con el siguiente encabezado:

```
%%%%
VERSION:1                               (Versión M94 o M94A del programa)
LANGUAGE:ENGLISH                         (o cualquier otro idioma: alemán o
%%%%                                         francés)
```

1.9 Sintaxis

Identificadores

```
<identificador> ::= 
    <ident>
    | <ID>
<ident> ::= <letra> {<letra> | <dígito> | '_'}
```

Valores numéricicos

```
<número literal> ::=
    <íntegro> [ <exponente> ]
    | <íntegro> '.' [ <íntegro> ] [ <exponente> ]
    | [ <íntegro> ] '.' <íntegro> [ <exponente> ]
<íntegro> ::= <dígito> {<dígito>}
<exponente> ::= ('E' | 'e') [ '+' | '-' ] <íntegro>
```

Valores lógicos

```
<booleno literal> ::= TRUE | FALSE
```

Cadenas de valores

```
<cadena literal> ::= '"' {<carácter> | <código carácter>} '"'
<código carácter> ::= '\' <dígito hex> <dígito hex>
<dígito hex> ::= <dígito> | A | B | C | D | E | F | a | b | c | d | e | f
```

Comentarios

```
<comentario> ::=
    '!' {<carácter> | <tab>} <nuevalínea>
```

Caracteres

```
<carácter> ::= -- ISO 8859-1 --
<nuevalínea> ::= -- carácter control nuevalínea--
```

<dígito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<letra> ::=

<letra mayúscula>
| <letra minúscula>

<letra mayúscula> ::=

A | B | C | D | E | F | G | H | I | J
| K | L | M | N | O | P | Q | R | S | T
| U | V | W | X | Y | Z | À | Á | Â | Ã
| Ä | Å | Æ | Ç | È | É | Ê | Ë | Ì | Í
| Î | Ï | ¹⁾ | Ñ | Ò | Ó | Ô | Õ | Ö | Ø
| Ù | Ú | Û | Ü | ²⁾ | ³⁾

<letra minúscula> ::=

a | b | c | d | e | f | g | h | i | j
| k | l | m | n | o | p | q | r | s | t
| u | v | w | x | y | z | ß | à | á | â
| ã | ä | å | æ | ç | è | é | ê | ë | ì
| í | î | ï | ¹⁾ | ñ | ò | ó | ô | õ | ö
| ø | ù | ú | û | ü | ²⁾ | ³⁾ | ÿ

- 1) Letra eth islandesa.
- 2) Letra Y con acento agudo
- 3) Letra thorn islandesa.

2 Módulos

Una aplicación está dividida en un *programa* y *módulos del sistema*. El programa está también dividido a su vez en módulos (véase la Figura 1).

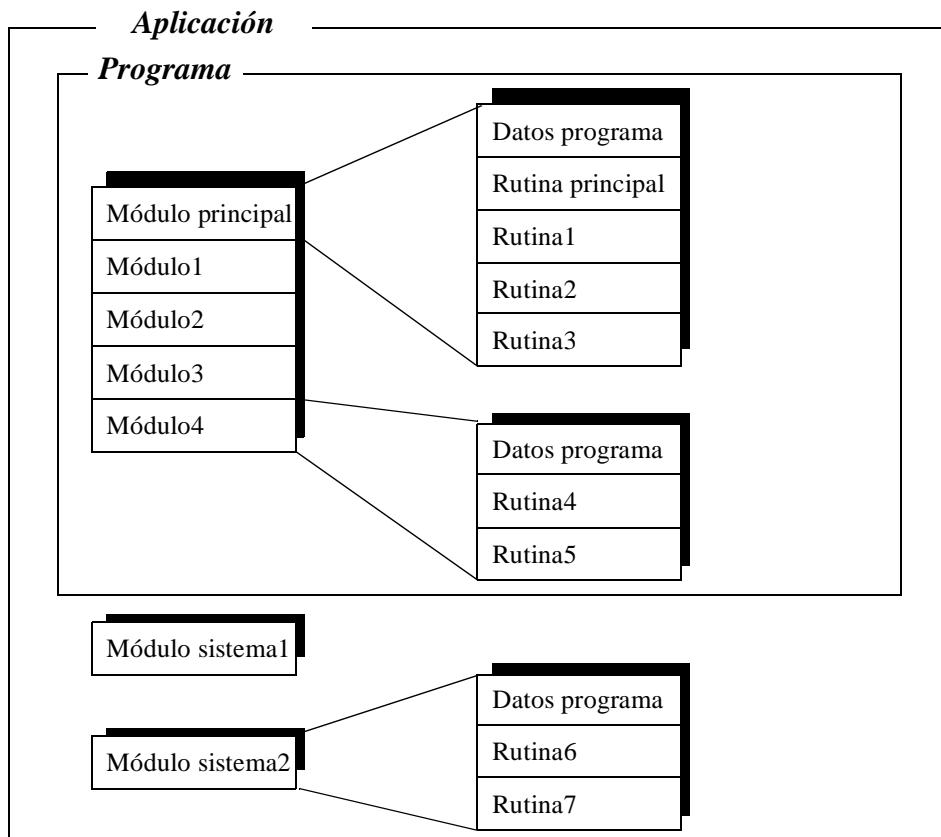


Figura 1 El programa está dividido en módulos.

2.1 Módulos del programa

Un módulo de programa puede estar formado de diferentes datos y rutinas. Cada módulo o el programa entero, podrá ser copiado en un disquete, a la memoria RAM, etc., y viceversa.

Uno de los módulos contiene el procedimiento de entrada, un procedimiento global llamado *main (principal)*. Ejecutar el programa significa, en hechos reales, ejecutar el procedimiento principal. El programa puede incluir varios módulos, pero sólo uno de ellos contiene un procedimiento principal.

Un módulo puede, por ejemplo, definir el interface con el equipo externo o contener datos geométricos que han sido generados o bien por sistemas CAD o creados in situ mediante la unidad de programación.

Mientras que las pequeñas instalaciones suelen estar contenidas en un módulo, las instalaciones más complejas pueden tener un módulo principal que hace referencia a rutinas y/o a datos contenidos en uno o varios otros módulos.

2.2 Módulos del sistema

Los módulos del sistema sirven para la definición de los datos y rutinas normales, específicos del sistema, como por ejemplo, las herramientas. No serán incluidos cuando se salva un programa, lo cual significa que cualquier actualización realizada en un módulo del sistema afectará a todos los programas que se encuentran en él, o que han sido cargados en una etapa ulterior en la memoria del programa.

2.3 Declaraciones de los módulos

Una declaración de módulo especifica el nombre y los atributos de este módulo. Dichos atributos sólo podrán añadirse de forma off-line, y no mediante la unidad de programación. A continuación se ofrecen algunos ejemplos de atributos de un módulo:

<u>Atributo</u>	<u>Si está especificado:</u>
SYSMODULE	el módulo será un módulo del sistema; de lo contrario, será un módulo del programa.
NOSTEPIN	no se podrá entrar en el módulo durante una ejecución paso a paso.
VIEWONLY	el módulo no podrá ser modificado
READONLY	el módulo no podrá ser modificado, pero sus atributos podrán ser eliminados
NOVIEW	el módulo no podrá ser visualizado, sólo ejecutado. Las rutinas globales podrán ser alcanzadas desde otros módulos y siempre son ejecutadas como NOSTEPIN. Los valores actuales de los datos globales podrán ser alcanzados desde otros módulos o desde la ventana de datos de la unidad de programación. Un módulo o un programa que contiene un módulo de programa NOVIEW no podrá ser salvado. Por esta razón, NOVIEW deberá utilizarse con prioridad para los módulos del sistema. NOVIEW sólo podrá ser definido off-line desde un PC.
por ejemplo:	<pre>MODULE nombre_módulo (SYSMODULE, VIEWONLY) !definición tipo datos !declaraciones datos !declaraciones rutinas ENDMODULE</pre>

Un módulo no podrá tener el mismo nombre que otro módulo o que una rutina global o que un dato.

2.4 Sintaxis

Declaración de un módulo

```
<declaración módulo> ::=  
    MODULE <nombre módulo> [ <lista atributos módulos> ]  
    <lista definición tipo>  
    <lista declaración datos>  
    <lista declaración rutina>  
    ENDMODULE  
<nombre módulo> ::= <identificador>  
<lista atributos módulos> ::= '(' <atributos del módulo> { ';' <atributos del módulo>  
    } ')'  
<atributos del módulo> ::=  
| SYSMODULE  
| NOVIEW  
| NOSTEPIN  
| VIEWONLY  
| READONLY
```

(**Nota** Si se utilizan dos o más atributos, deberán estar en el orden en que se indica anteriormente, el atributo NOVIEW sólo podrá ser especificado solo o junto con el atributo SYSMODULE.)

```
<lista definición tipo> ::= { <definición tipos> }  
<lista declaración datos> ::= { <declaración de datos> }  
<lista declaración rutina> ::= { <declaración de rutina> }
```

3 Rutinas

Existen tres tipos de rutinas (subprogramas): los *procedimientos*, las *funciones* y las *rutinas de tratamiento de interrupciones*.

- Los procedimientos no devuelven ningún valor y se utilizan en el contexto de las instrucciones.
- Las funciones devuelven un valor de un tipo específico y se utilizan en el contexto de las expresiones.
- Las rutinas de tratamiento de interrupciones proporcionan una manera de procesar las interrupciones. Una rutina de tratamiento de interrupciones puede estar asociada a una interrupción específica y entonces, en el caso en que dicha interrupción ocurra en una etapa ulterior, se volverá a ejecutar automáticamente. No se podrá nunca llamar explícitamente una rutina de tratamiento de interrupciones desde el programa.

3.1 Alcance de las rutinas

El alcance de una rutina indica el área en la que la rutina está accesible. La directiva opcional de una declaración de una rutina, la clasifica como una rutina local (dentro del módulo), o como global.

Ejemplo:

```
LOCAL PROC rutina_local (...  
PROC rutina_global (...
```

Las siguientes pautas referentes al alcance son aplicables a las rutinas (véase el ejemplo de la Figura 2):

- El alcance de una rutina global puede incluir cualquier módulo.
- El alcance de una rutina local comprende el módulo en el que se encuentra.
- Dentro de su alcance, una rutina local oculta todas las rutinas globales o datos que tengan el mismo nombre.
- Dentro de su alcance, una rutina oculta las instrucciones y rutinas predefinidas o datos que tengan el mismo nombre.

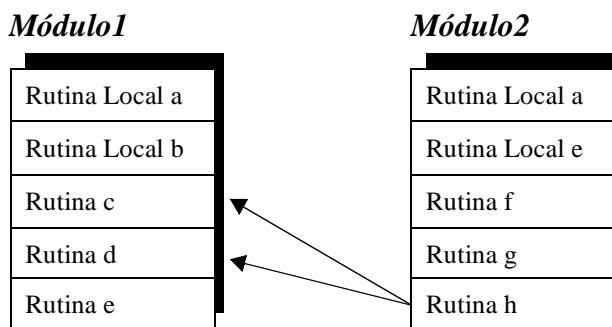


Figura 2 Ejemplo: Las rutinas siguientes podrán ser llamadas desde la Rutina h:
Módulo1 - Rutina c, d. Módulo2 - Todas las rutinas.

Una rutina no deberá tener el mismo nombre que otra rutina o datos dentro del mismo módulo. Una rutina global no deberá tener el mismo nombre que otra rutina global o que datos globales de otro módulo.

3.2 Parámetros

La lista de parámetros de una declaración de rutina especifica los argumentos (parámetros actuales) que deberán ser suministrados cuando se llama a una rutina.

Existen cuatro tipos diferentes de parámetros (en el modo de acceso normal):

- Normalmente, un parámetro suele ser utilizado únicamente como una entrada y es procesado como una variable de rutina. El cambio de esta variable no cambiará el argumento correspondiente.
- Un parámetro INOUT especifica que el argumento correspondiente debe ser una variable (entero, elemento o componente) o un entero persistente que puede ser cambiado por la rutina.
- Un parámetro VAR especifica que el argumento correspondiente debe ser una variable (entero, elemento o componente) que puede ser cambiada por la rutina.
- Un parámetro PERS especifica que el argumento correspondiente deberá ser un persistente entero que podrá ser cambiado por la rutina.

Si un parámetro INOUT, VAR o PERS es actualizado, ello significa que el argumento en sí será actualizado, es decir, que se posibilita la utilización de argumentos para devolver valores a la rutina que llama.

Ejemplo: PROC rutina1 (num par_in, INOUT num par_inout, VAR num par_var,PERS num par_pers)

Un parámetro puede ser opcional y puede ser omitido en la lista de argumentos de una llamada de rutina. Un parámetro opcional se reconoce por la barra invertida “\” que le precede.

Ejemplo: PROC rutina2 (num par_necesario \num par_opcional)

No se podrá hacer referencia a un valor de un parámetro opcional que es omitido en una llamada de rutina. Esto significa que las llamadas de rutina deberán ser comprobadas por si se encuentra algún parámetro opcional antes de poder utilizar un parámetro opcional.

Puede ocurrir que dos o más parámetrosopcionales sean mutuamente exclusivos (es decir, declarados para excluirse mutuamente), y esto significa que sólo podrá haber uno de ellos presente en la llamada de rutina. Esto está indicado por una línea “|” situada entre los parámetros correspondientes.

Ejemplo: PROC rutina3 (\num excluye1 | num excluye2)

El tipo especial, *switch*, sólo podrá ser asignado a parámetros opcionales y proporciona una manera de utilizar los argumentos switch, es decir, argumentos que sólo están especificados por los nombres (y no por los valores). Un valor no podrá ser transferido a un parámetro switch. La única forma de utilizar un parámetro switch es comprobar su presencia mediante la función predefinida, *Present*.

Ejemplo: PROC rutina4 (\switch on | switch off)

```
...
IF present (off ) THEN
...
ENDPROC
```

Las matrices pueden utilizarse como argumentos. El grado de un argumento de matriz deberá corresponder con el grado del parámetro formal correspondiente. La dimensión de la matriz será definida por el parámetro (señalada con un “*”). Así, la dimensión actual dependerá de la dimensión del argumento correspondiente en una llamada de rutina. Una rutina podrá determinar las dimensiones actuales de un parámetro utilizando la función

predefinida, *Dim*.

Ejemplo: PROC rutina5 (VAR num palet{*,*})

3.3 Final de una rutina

El final de la ejecución de un procedimiento está indicado explícitamente mediante una instrucción RETURN o bien puede estar indicado de forma implícita cuando se alcance el final (ENDPROC, BACWARD o ERROR) del procedimiento.

La evaluación de una función deberá terminarse con una instrucción RETURN.

El final de la ejecución de una rutina de tratamiento de interrupciones está indicado de forma explícita mediante la instrucción RETURN o bien, de forma implícita cuando se alcanza el final (ENDTRAP o ERROR) de esta rutina de tratamiento de interrupciones. La ejecución continuará a partir del punto en que ocurrió la interrupción.

3.4 Declaraciones de rutina

Una rutina puede comprender declaraciones de rutina (incluyendo parámetros), datos, un cuerpo de instrucciones, un gestor de ejecución hacia atrás (sólo para procedimientos) y un gestor de errores (véase la Figura 3). Las declaraciones de rutina no podrán ser anidadas, es decir, que no se podrá declarar una rutina dentro de una rutina.

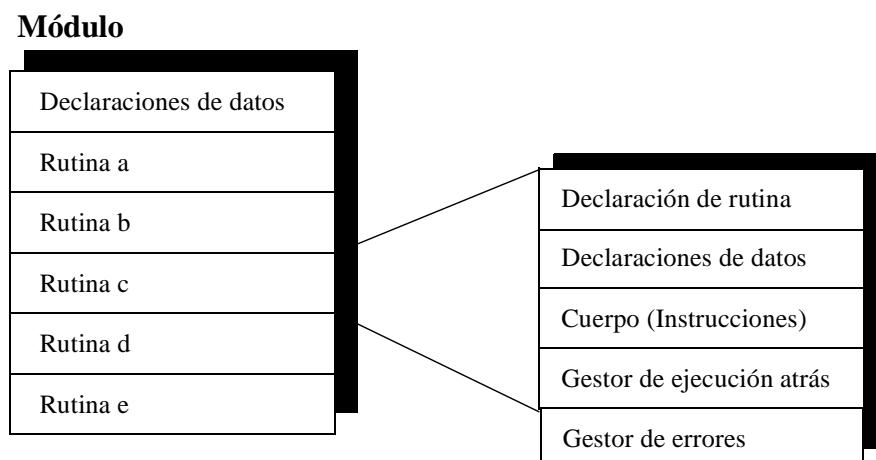


Figura 3 Una rutina puede contener declaraciones, datos, un cuerpo de instrucciones, un gestor de ejecución hacia atrás y un gestor de errores.

Declaración de un procedimiento

Ejemplo: Multiplicación de todos los elementos de una matriz numérica por un factor;

```
PROC mulmatrix (VAR num matriz {*} , num factor)
    FOR índice FROM 1 TO dim( matriz, 1 ) DO
        matriz{índice} := matriz{índice} * factor;
    ENDFOR
ENDPROC
```

Declaración de una función

Una función puede devolver cualquier valor de tipo de dato, pero no un valor de matriz.

Ejemplo: Devolver la longitud de un vector;

```
FUNC num veclen (pos vector)
    RETURN Sqrt(Pow(vector.x,2)+Pow(vector.y,2)+Pow(vector.z,2));
ENDFUNC
```

Declaración de una rutina de tratamiento de interrupciones

Ejemplo: Respuesta a la interrupción de alimentador vacío;

```
TRAP alim_vacío
espera_alim;
RETURN;
ENDTRAP
```

3.5 Llamada de procedimiento

Cuando se llama un procedimiento, los argumentos que corresponden a los parámetros del procedimiento deberán utilizarse:

- Los parámetros de mandato deberán ser especificados. Además, deberán ser especificados siguiendo el orden correcto.
- Los argumentos opcionales podrán ser omitidos.
- Los argumentos condicionales podrán utilizarse para transferir los parámetros de una llamada de rutina a otra.

Véase el capítulo *Utilización de llamadas a función en las expresiones* en la página 29 para más detalles.

El nombre del procedimiento podrá ser especificado de forma fija utilizando un identificador (*early binding*) o bien podrá ser evaluado durante el tiempo de funcionamiento desde una expresión de tipo cadena (*late binding*). Incluso si el método *early binding* debe ser considerado como la forma de llamada de proceso ‘normal’, *late binding* con frecuencia proporciona un código compacto y muy eficaz. *Late binding* se define colo-

cando el signo de porcentaje antes y después de la cadena que denota el nombre del proceso.

Ejemplo:

```

! early binding
TEST products_id
CASE 1:
    proc1 x, y, z;
CASE 2:
    proc2 x, y, z;
CASE 3:
    ....
!
```

! mismo ejemplo utilizando binding

```

% "proc" + NumToStr(product_id, 0) % x, y, z;
....
```

! mismo ejemplo otra vez utilizando otra variante de late binding

```

VAR string procname {3} :=["proc1","proc2", "proc3"];
....
% procname {product_id} % x, y, z;
....
```

Téngase en cuenta que late binding está disponible únicamente para llamadas de procedimientos, y no para llamadas de función. Si se hace referencia a un procedimiento desconocido utilizando late binding, la variable del sistema ERRNO se activa en ERR_REFUNKPRC. Si se hace referencia a un error de llamada de procedimiento (sintaxis, no procedimiento) utilizando late binding, la variable del sistema ERRNO se activa en ERR_CALLPROC.

3.6 Sintaxis

Declaración de una rutina

```

<declaración rutina> ::=

    [LOCAL] ( <declaración procedimiento>
        | <declaración función>
        | <declaración trap> )
    | <comentario>
    | <RDN>
```

Parámetros

```

<lista parámetros> ::=

    <primera declaración parámetro>{ <siguiente declaración parámetro>
    }

<primera declaración parámetro> ::=
    <declaración parámetro>
    | <declaración parámetro opcional>
    | <PAR>
```

```

<siguiente declaración parámetro> ::= 
    ',' <declaración parámetro>
    | <declaración parámetro opcional>
    | ',' <PAR>

<declaración parámetro opcional> ::= 
    '\' ( <declaración parámetro> | <ALT> )
    { '}' ( <declaración parámetro> | <ALT> ) }

<declaración parámetro> ::= 
    [ VAR | PERS | INOUT] <tipo dato>
    <identificador> [ '{' '*' { ',', '*' } ')' | <DIM> ] '}'
    | 'switch' <identificador>

```

Declaración de un procedimiento

```

<declaración procedimiento> ::= 
    PROC <nombre procedimiento>
    '(' [ <lista parámetro> ] ')'
    <lista declaración datos>
    <lista instrucción>
    [ BACKWARD <lista instrucción> ]
    [ ERROR <lista instrucción> ]
    ENDPROC

<nombre procedimiento> ::= <identificador>
<lista declaración datos> ::= {<declaración datos>}

```

Declaración de una función

```

<declaración función> ::= 
    FUNC <tipo valor dato>
    <nombre función>
    '(' [ <lista parámetro> ] ')'
    <lista declaración datos>
    <lista instrucción>
    [ ERROR <lista instrucción> ]
    ENDFUNC

<nombre función> ::= <identificador>

```

Declaración de una rutina de tratamiento de interrupciones

```

<declaración trap> ::= 
    TRAP <nombre trap>
    <lista declaración datos>
    <lista instrucción>
    [ ERROR <lista instrucción> ]
    ENDTRAP

<nombre trap> ::= <identificador>

```

Llamada de procedimiento

```
<llamada procedimiento> ::= <procedimiento> [ <lista argumento procedimiento> ] ';'
```

```
<procedimiento> ::=
    = <identificador>
    | '%' <expresión> '%'
<lista argumento procedimiento> ::= <primer argumento procedimiento> { <argu-
    mento procedimiento> }
<primer argumento procedimiento> ::=
    <argumento procedimiento requerido>
    | <argumento procedimiento opcional>
    | <argumento procedimiento condicional>
    | <ARG>
<argumento procedimiento> ::=
    ',' <argumento procedimiento requerido>
    | <argumento procedimiento opcional>
    | <argumento procedimiento condicional>
    | ',' <ARG>
<argumento procedimiento requerido> ::= [ <identificador> ':=' ] <expresión>
<argumento procedimiento opcional> ::= '\' <identificador> [ ':=' <expresión> ]
<argumento procedimiento condicional> ::= '\' <identificador> '?' ( <parámetro> |
    <VAR> )
```

4 Tipos de Datos

Existen dos grupos diferentes de tipos de datos:

- El tipo *atómico* se refiere a los datos cuya definición no está basada en ningún otro tipo y que no puede ser dividido en partes ni componentes, por ejemplo, *num*.
 - El tipo *registro* es un tipo de dato compuesto formado por componentes ordenados y nombrados, por ejemplo, *pos*. Un componente puede ser de tipo atómico o registro.

Un valor de registro puede ser expresado mediante una representación *agregada*;

por ejemplo: [300, 500, long] Valor agregado registro pos.

Se podrá acceder a un componente específico de un dato registro utilizando el nombre de este componente;

por ejemplo: pos1.x := 300; asignación del componente x de pos1.

4.1 Tipos de datos *sin valor*

Todo tipo de dato disponible puede ser un tipo de dato *valor* o bien un tipo de dato *sin valor*. De forma general, diremos que un tipo de dato valor representa alguna forma de “valor”. Los datos sin valor no podrán ser utilizados en operaciones que tratan con valores, es decir en:

- Inicialización
 - Asignación ($:=$)
 - Comprobaciones del tipo Igual a ($=$) y no igual a (\neq)
 - Instrucciones TEST
 - Parámetros IN (modo de acceso) en llamadas de rutina
 - Tipos de datos de función (retorno)

Los tipos de datos de entrada (*signalai*, *signaldi*, *signalgi*) son del tipo de dato llamado *semi valores*. Estos datos pueden utilizarse en operaciones que tratan con valores, excepto para la inicialización y la asignación.

Sólo los tipos de datos *sin valor* o *semi valores* serán definidos en la descripción de los tipos de datos.

4.2 Tipos de datos *iguales a* (*equivalente a*)

Un tipo de dato *equivalente* será definido como siendo igual a otro tipo. Los datos del mismo tipo podrán ser sustituidos uno por otro.

Ejemplo: VAR dionum alto:=1;
 VAR num nivel;
 nivel := alto;

Esto es correcto siempre y cuando dionum sea un tipo equivalente a num.

4.3 Sintaxis

```
<type definition> ::=  
[LOCAL] ( <record definition>  
          | <alias definition> )  
| <comment>  
| <TDN>  
  
<record definition> ::=  
RECORD <identifier>  
      <record component list> ';'  
ENDRECORD  
  
<record component list> ::=  
    <record component definition> |  
    <record component definition> <record component list>  
  
<record component definition> ::=  
    <data type> <record component name>  
  
<alias definition> ::=  
    ALIAS <data type> <identifier> ';'  
  
<data type> ::= <identifier>
```

5 Datos

Existen tres tipos de datos: *variables*, *persistentes* y *constantes*.

- Un dato variable podrá tener asignado un valor nuevo durante la ejecución del programa.
- Un dato persistente puede describirse como una variable “persistente”. Es una variable que tiene la particularidad de que su valor de inicialización se actualiza a medida que va variando. (Cuando se guarda un programa, el valor de inicialización de cualquier declaración persistente refleja el último valor que ha tomado el dato persistente).
- Un dato constante representa un valor estático al que no se le podrá asignar ningún valor nuevo.

Una declaración de datos introduce datos asociando un nombre (identificador) a un tipo de dato. Todos los datos utilizados deben ser siempre declarados excepto cuando se trata de datos predefinidos y de variables de bucle.

5.1 Alcance de los datos

El alcance de los datos indica el área en la que el dato es visible. La directiva local opcional de una declaración de dato clasifica los datos en dos grupos: locales (dentro del módulo), y globales. Observar que la directiva local podrá utilizarse únicamente al nivel de módulo, no dentro de una rutina.

Ejemplo:

```
LOCAL VAR num variable_local;
      VAR num variable_global;
```

Los datos declarados frente de una rutina se llaman *datos de programa*. Las siguientes normas relativas al alcance se aplican a los datos de programa:

- El alcance de los datos de programa predefinidos o globales pueden incluir cualquier módulo.
- El alcance de los datos de programa locales comprenden el módulo en el que están definidos.
- Dentro de su alcance, los datos de programa locales ocultan todos los datos globales o rutinas que tengan el mismo nombre (incluyendo las instrucciones, las rutinas predefinidas y los datos).

Los datos de programa no deberán tener el mismo nombre que otros datos o que una rutina en el mismo módulo. Los datos de programa globales no deberán tener el mismo nombre que otros datos globales o que una rutina en otro módulo. Un dato persistente no deberá tener el mismo nombre que otro dato persistente en el mismo programa.

Los datos declarados dentro de una rutina se llaman *datos de rutina*. Observar que los parámetros de una rutina son procesados como datos de rutina. Las siguientes pautas referidas al alcance se aplican a los datos de rutina:

- El alcance de los datos de rutina comprenden la rutina en que se encuentran.
- Dentro de su alcance, los datos de rutina ocultan cualquier otra rutina o dato que tenga el mismo nombre.

Véase el ejemplo de la Figura 4.

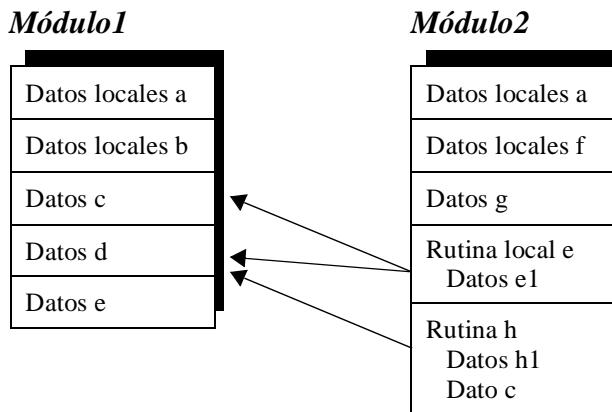


Figura 4 Ejemplo: Los siguientes datos podrán ser llamados a partir de la rutina e:

Módulo1: Datos c, d.

Módulo2: Datos a, f, g, e1.

Los siguientes datos podrán ser llamados a partir de la rutina h:

Módulo1: Datos d.

Módulo2: Datos a, f, g, h1, c.

Los datos de rutina no deberán tener el mismo nombre que otros datos o una etiqueta en la misma rutina.

5.2 Declaración de un dato variable

Un dato variable es introducido por medio de una declaración de dato variable.

Ejemplo: VAR num x;

Se podrá atribuir a todo tipo de variables un formato de matriz (del grado 1, 2 o 3) añadiendo una información dimensional a la declaración. Una dimensión es un valor entero mayor que 0.

Ejemplo: VAR pos palet{14, 18};

Los datos variables con tipos de valor podrán ser inicializados (es decir, que se les atribuirá un valor inicial). La expresión utilizada para inicializar una variable de programa deberá ser constante. Observar que el valor de una variable no inicializada podrá utilizarse, pero será no definida, es decir que estará en 0.

Ejemplo: VAR cadena nombre_autor := "John Smith";
 VAR pos inicio:= [100, 100, 50];
 VAR num nummax{10} := [1, 2, 3, 9, 8, 7, 6, 5, 4, 3];

El valor de inicialización se activa cuando:

- se abre el programa,
- se ejecuta el programa desde el principio.

5.3 Declaración de un dato persistente

Los datos persistentes sólo podrán ser declarados en el nivel de módulo, y por lo tanto, no podrán ser declarados dentro de una rutina, y deberán tener un valor inicial. Este valor de inicialización deberá ser un valor simple (sin datos ni operandos), o un agregado simple con componentes que, a su vez, son valores o agregados simples.

Ejemplo: PERS pos puntref:= [100.23, 778.55, 1183.98];

Se podrá dar un formato de matriz (de grado 1, 2 o 3) a los datos persistentes de cualquier tipo añadiendo información dimensional a la declaración. Una dimensión es un valor entero mayor que 0.

Ejemplo: PERS pos pallet {14, 18};

Observar que si el valor de un dato persistente es actualizado, ello provocará automáticamente la actualización del valor de inicialización en la declaración del dato persistente.

Ejemplo: PERS num reg1 := 0;
 ...
 reg1 := 5;

Después de la ejecución, el programa tendrá el siguiente aspecto:

PERS num reg1 := 5;
 ...
 reg1 := 5;

5.4 Declaración de un dato constante

Un dato constante es introducido por medio de una declaración constante. El valor de una constante no podrá ser modificado.

Ejemplo: CONST num pi := 3.141592654;

Se podrá dar un formato de matriz (del grado 1, 2 o 3) a una constante de cualquier tipo añadiendo una información dimensional a la declaración. Una dimensión es un valor entero mayor que 0.

Ejemplo: CONST pos dat {3} := [[614, 778, 1020],
 [914, 998, 1021],
 [814, 998, 1022]];

5.5 Datos de inicio

En la siguiente tabla, el usuario podrá observar lo que ocurre en diferentes situaciones, por ejemplo, cuando se produce un arranque en caliente, un programa nuevo, un arranque de programa, etc.

Tabla 1

Eventos del sistema Afecta a	Potencia act. (Arranque caliente)	Abrir , cerrar o programa nuevo	Arranque programa (Mover PP a principal)	Arranque programa (Mover PP a Rutina)	Arranque programa (Mover PP a cursor)	Arranque programa (llamada rutina)	Arranque programa (Después de ciclo)	Arranque programa (Después de paro)
Constante	Incambiada	Init	Init	Init	Incambiada	Incambiada	Incambiada	Incambiada
Variable	Incambiada	Init	Init	Init	Incambiada	Incambiada	Incambiada	Incambiada
Persistente	Incambiada	Init	Init	Init	Incambiada	Incambiada	Incambiada	Incambiada
Interrupciones ordenadas	Re-ordenadas	Desaparece	Desaparece	Desaparece	Incambiada	Incambiada	Incambiada	Incambiada
Rutina de arranque Reini_sist (con caract. de mov.)	No ejecutar	Ejecutar*	Ejecutar	No Ejecutar	No Ejecutar	No Ejecutar	No Ejecutar	No Ejecutar
Archivos	Se cierra	Se cierra	Se cierra	Se cierra	Incambiada	Incambiada	Incambiada	Incambiada
Trayectoria	Recreada después de potencia desactivada	Desaparece	Desaparece	Desaparece	Desaparece	Desaparece	Incambiada	Incambiada

* Genera un error cuando hay un error semántico en la tarea actual del programa.

5.6 Sintaxis

Declaración de un dato

```
<declaración dato> ::= 
    [LOCAL] ( <declaración variable>
        | <declaración persistente>
        | <declaración constante> )
    | <comentario>
    | <DDN>
```

Declaración de un dato variable

```
<declaración variable> ::= 
    VAR <tipo datos> <definición variable> ;
<definición variable> ::= 
    <identificador> [ '{' <dim> { ',' <dim> } '}' ]
    [ ':=' <expresión constante> ]
<dim> ::= <expresión constante>
```

Declaración de un dato persistente

```
<declaración persistente> ::= 
    PERS <tipo datos> <definición persistente> ;
```

```
<definición persistente> ::=  
    <identificador> [ '{' <dim> { ',' <dim> } '}' ]  
    ':=' <expresión literal>
```

Declaración de un dato constante

```
<declaración constante> ::=  
    CONST <tipo datos> <definición constante> ';'  
<definición constante> ::=  
    <identificador> [ '{' <dim> { ',' <dim> } '}' ]  
    ':=' <expresión constante>  
<dim> ::= <expresión constante>
```

6 Instrucciones

Las instrucciones son ejecutadas de forma sucesiva a menos que una instrucción de flujo de programa, una interrupción o un error obligue a continuar la ejecución en cualquier otro lugar.

La mayoría de instrucciones terminan con un punto y coma “;”. Una etiqueta termina con dos puntos “:”. Ciertas instrucciones podrán contener otras instrucciones y entonces finalizarán con palabras clave específicas, según se indica a continuación:

<u>Instrucción</u>	<u>Palabra de finalización</u>
IF	ENDIF
FOR	ENDFOR
WHILE	ENDWHILE
TEST	ENDTEST

Ejemplo: WHILE índice < 100 DO

índice := índice + 1;
ENDWHILE

6.1 Sintaxis

```
<lista instrucción> ::= { <instrucción> }
<instrucción> ::=
    [<instrucción de acuerdo con el capítulo correspondiente de este
    manual>
    | <SMT>]
```

7 Expresiones

Una expresión especifica la evaluación de un valor. Podrá utilizarse, por ejemplo:

- en una instrucción de asignación ej. $a := 3 * b / c;$
 - como una condición en una instrucción IF ej. IF $a \geq 3$ THEN ...
 - como un argumento en una instrucción ej. `WaitTime tiempo;`
 - como un argumento en una llamada de función ej. $a := \text{Abs}(3 * b);$

7.1 Expresiones aritméticas

Una expresión aritmética sirve para evaluar un valor numérico.

Ejemplo: $2\pi r$

En la Tabla 2 se indican los diferentes tipos de operaciones posibles.

Tabla 2 Operaciones aritméticas

Operador	Operación	Tipos de operando	Tipo de resultado
+	suma	num + num	num ³⁾
+	mantener el signo	+num o +pos	igual ¹⁾³⁾
+	suma de un vector	pos + pos	pos
-	resta	num - num	num ³⁾
-	cambio de signo	-num o -pos	igual ¹⁾³⁾
-	resta de un vector	pos - pos	pos
*	multiplicación	num * num	num ³⁾
*	multiplicación de un vector por un escalar	num * pos o pos * num	pos
*	producto de vectores	pos * pos	pos
*	enlace de rotaciones	orient * orient	orient
/	división	num / num	num
DIV ²⁾	división entera	num DIV num	num
MOD ²⁾	módulo entero; resto	num MOD num	num

1. El resultado será del mismo tipo que el operando. Si el operando tiene un tipo de datos equivalente, el resultado será del tipo equivalente de “base” (num o pos).
 2. Operaciones con enteros, por ejemplo: $14 \text{ DIV } 4 = 3$, $14 \text{ MOD } 4 = 2$.
(No se aceptarán operandos que no sean enteros)
 3. Conserva una representación (exacta) con enteros siempre y cuando los operandos y resultados estén dentro del subconjunto de enteros del tipo num.

7.2 Expresiones lógicas

Una expresión lógica sirve para evaluar un valor lógico (VERDADERO/FALSO).

Ejemplo: $a > 5$ AND $b = 3$

En la Tabla 3 se indican los diferentes tipos de operaciones posibles.

Tabla 3 Operaciones lógicas

Operador	Operación	Tipo de operando	Tipo de resultado
<	menor que	num < num	bool
<=	menor o igual que	num <= num	bool
=	igual que	any ¹⁾ = any ¹⁾	bool
>=	mayor o igual que	num >= num	bool
>	mayor que	num > num	bool
\neq	no igual que	cualquiera ¹⁾ \neq cualquiera ¹⁾	bool
AND	y	bool AND bool	bool
XOR	exclusivo o	bool XOR bool	bool
OR	o	bool OR bool	bool
NOT	negación	NOT bool	bool

1) Solamente tipos de datos *valor*. Los operandos deben ser del mismo tipo.

a AND b		a XOR b			
a	b	Verdadero	Falso	Verdadero	Falso
Verdadero	Verdadero	Verdadero	Falso	Verdadero	Verdadero
	Falso	Falso	Falso		Falso
Falso	Verdadero	Falso	Verdadero	Verdadero	Falso
	Falso	Falso	Falso		

a OR b		NOT b			
a	b	Verdadero	Falso	Verdadero	Falso
Verdadero	Verdadero	Verdadero	Verdadero	Verdadero	Falso
	Falso	Verdadero	Falso		Falso
Falso	Verdadero	Falso	Verdadero	Falso	Verdadero
	Falso	Falso	Falso		

7.3 Expresiones de cadena

Una expresión de cadena sirve para llevar a cabo operaciones en las cadenas.

Ejemplo: “IN” + “PUT”

dará el resultado “INPUT”

La Tabla 4 muestra la única operación posible.

Tabla 4 Operación de cadenas

Operador	Operación	Tipo de operando	Tipo de resultado
+	concatenación de cadena	string + string	string

7.4 Utilización de datos en las expresiones

Un dato variable entero, persistente o constante puede formar parte de una expresión.

Ejemplo: 2*pi*radio

Matrices

Un dato variable, persistente o constante declarado como una matriz podrá hacer referencia a la matriz completa o a un elemento de la misma.

Para hacer referencia a un elemento de matriz, se utilizará el número de índice del elemento. El índice está formado por un valor entero mayor que 0 y no deberá exceder la dimensión declarada. El valor de índice 1 seleccionará el primer elemento. El número de elementos de la lista de índice deberá cuadrar con el valor declarado (1, 2 o 3) de la matriz.

Ejemplo:

VAR num fila {3};	
VAR num columna{3};	sólo un elemento de la matriz
VAR num valor;	todos los elementos de la matriz
.	
valor := columna{3};	
fila := columna;	

Registros

Un dato variable, persistente o constante declarados como un registro podrán hacer referencia al registro completo o a un elemento individual.

Se hará referencia a un componente de un registro utilizando el nombre del componente.

Ejemplo:

VAR pos reposo;	
VAR pos pos1;	sólo el componente Y
VAR num valor y;	la posición completa
..	
valory := reposo.y;	
pos1 := reposo;	

7.5 Utilización de agregados en las expresiones

Un agregado se utiliza en los valores de matriz o de registro.

Ejemplo:	pos := [x, y, 2*x] matpos := [[0, 0, 100], [0,0,z]]	agregado de registro pos agregado de matriz pos
----------	--	--

Debe ser posible determinar el tipo de dato de un agregado del contexto. El tipo de dato de cada miembro del agregado deberá ser igual que el tipo del miembro correspondiente del tipo determinado.

Ejemplo:	VAR pos p1; p1 :=[1, -100, 12]	pos tipo agregado - determinado por p1
----------	-----------------------------------	--

IF [1, -100, 12] = [a,b,b] THEN	incorrecto porque el tipo de dato de ninguno de los agregados puede ser determinado por el contexto.
---------------------------------	--

7.6 Utilización de llamadas a función en las expresiones

Una llamada a función inicia la evaluación de una función específica y recibe el valor devuelto por la función.

Ejemplo: Sin(ángulo)

Los argumentos de una llamada de función sirven para transferir datos a (y posiblemente de) la función llamada. El tipo de dato de un argumento debe ser el mismo que el tipo del parámetro correspondiente de la función. Los argumentos opcionales podrán ser omitidos pero el orden de los argumentos (presentes) deberá ser el mismo que el orden de los parámetros formales. Además, dos o más argumentosopcionales podrán ser declarados de forma que se excluyan mutuamente; en tal caso, sólo uno de ellos deberá estar presente en la lista de argumentos.

Un argumento requerido (obligatorio) será separado del argumento precedente por una coma “,”. El nombre del parámetro formal podrá ser omitido o incluido.

Ejemplo:	Polar(3.937, 0.785398) Polar(Dist:=3.937, Angulo:=0.785398)	dos argumentos requeridos ... utilización de nombres
----------	--	---

Un argumento opcional deberá estar precedido de una barra invertida “\” y del nombre del parámetro formal. Un argumento del tipo switch es un argumento un tanto especial; puede ocurrir que no incluya ninguna expresión de argumento. En vez de ello, un argumento de este tipo sólo podrá ser "presente" o "no presente".

Ejemplo:	Cosine(45) Cosine(0.785398\Rad) Dist(p2) Dist(\distance:=pos1, p2)	un argumento requerido ... y uno tipo switch un argumento requerido ... y uno opcional
----------	---	---

Los argumentos condicionales se utilizan para asegurar una propagación suave de los argumentos opcionales en las cadenas de llamadas de rutinas. Se considera que un argumento condicional está «presente» cuando el parámetro opcional especificado (de la función llamada) está presente, de lo contrario se considera simplemente que ha sido omitido. Observar que el parámetro especificado deberá ser opcional.

Ejemplo:

```
PROC Read_from_file (iodev File \num Maxtime)
    .. character:=ReadBin (File \Time?Maxtime);
    ! Max. tiempo será utilizado únicamente si está especificado cuando
    se llama la rutina
    ! Read_from_file
    ..
ENDPROC
```

La lista de parámetros de una función atribuye un *modo de acceso normal* a cada parámetro. El modo de acceso normal podrá ser *in*, *inout*, *var* o *pers*:

- Un parámetro IN (por defecto) permite que el argumento sea cualquier expresión. La función llamada visualiza el parámetro como una constante.
- Un parámetro INOUT requiere que el argumento correspondiente sea una variable (entero, elemento de matriz o componente de registro) o un entero persistente. La función llamada obtiene un acceso total (lectura/escritura) al argumento.
- Un parámetro VAR requiere que el argumento correspondiente sea una variable (entero, elemento de matriz o componente de registro). La función llamada obtiene un acceso total (lectura/escritura) al argumento.
- Un parámetro PERS requiere que el argumento correspondiente sea un persistente entero. La función llamada obtiene un acceso total (lectura/actualización) al argumento.

7.7 Prioridad entre los operadores

La prioridad relativa de los operadores determina el orden en que son evaluados. Los paréntesis proporcionan una manera de anular la prioridad del operador. La reglas que se indican a continuación implican la siguiente prioridad del operador:

* / DIV MOD	- más alta
+ -	
<> <> <= >= =	
AND	
XOR OR NOT	- más baja

Un operador con una prioridad alta será evaluada antes que un operador con una prioridad baja. Los operadores que tengan la misma prioridad serán evaluados de izquierda a derecha.

Ejemplo

<u>Expresión</u>	<u>Orden de evaluación</u>	<u>Comentario</u>
$a + b + c$	$(a + b) + c$	regla de izquierda a derecha
$a + b * c$	$a + (b * c)$	* más alto que +
$a \text{ OR } b \text{ OR } c$	$(a \text{ OR } b) \text{ OR } c$	regla de izquierda a derecha
$a \text{ AND } b \text{ OR } c \text{ AND } d$	$(a \text{ AND } b) \text{ OR } (c \text{ AND } d)$	AND más alto que OR
$a < b \text{ AND } c < d$	$(a < b) \text{ AND } (c < d)$	< más alto que AND

7.8 Sintaxis**Expresiones**

```

<expresión> ::= 
    <expr>
    | <EXP>

<expr> ::= [ NOT ] <término lógico> { ( OR | XOR ) <término lógico> }
<término lógico> ::= <relación> { AND <relación> }
<relación> ::= <expresión simple> [ <relop> <expresión simple> ]
<expresión simple> ::= [ <addop> ] <término> { <addop> <término> }
<término> ::= <primario> { <mulop> <primario> }
<primario> ::= 
    <literal>
    | <variable>
    | <persistente>
    | <constante>
    | <parámetro>
    | <llamada función>
    | <agregado>
    | '(' <expr> ')'
  
```

Operadores

```

<relop> ::= '<' | '<=' | '=' | '>' | '>=' | '><'
<addop> ::= '+' | '-'
<mulop> ::= '*' | '/' | DIV | MOD
  
```

Valores constantes

```

<literal> ::= <num literal>
    | <cadena literal>
    | <bool literal>
  
```

Datos

```
<variable> ::=  
    <variable entera>  
    | <elemento variable>  
    | <componente variable>  
<variable entera> ::= <ident>  
<elemento variable> ::= <variable entera> '{' <lista índice> '}'  
<lista índice> ::= <expr> { ',' <expr> }  
<componente variable> ::= <variable> '.' <nOMBRE componente>  
<nOMBRE componente> ::= <ident>  
<persistente> ::=  
    <persistente entera>  
    | <elemento persistente>  
    | <componente persistente>  
<constante> ::=  
    <constante entera>  
    | <elemento constante>  
    | <componente constante>
```

Agregados

```
<agregado> ::= '[' <expr> { ',' <expr> } ']'
```

Llamadas de función

```
<llamada función> ::= <función> '(' [ <lista argumento función> ] ')'  
<función> ::= <ident>  
<lista argumento función> ::= <primer argumento de la función> { <argumento de la función> }  
<primer argumento de la función> ::=  
    <argumento de la función requerida>  
    | <argumento de la función opcional>  
    | <argumento de la función condicional>  
<argumento de la función> ::=  
    ',' <argumento de la función requerida>  
    | <argumento de la función opcional>  
    | <argumento de la función condicional>  
<argumento de la función requerida> ::= [ <ident> ':=' ] <expr>  
<argumento de la función opcional> ::= '\' <ident> [ ':=' <expr> ]  
<argumento de la función condicional> ::= '\' <ident> '?' [ ':=' <parámetro> ]
```

Expresiones especiales

```
<expresión constante> ::= <expresión>  
<expresión literal> ::= <expresión>  
<expresión condicional> ::= <expresión>
```

Parámetros

```
<parámetro> ::=  
    <parámetro entero>  
    | <elemento parámetro>  
    | <componente parámetro>
```

8 Recuperación de Errores

Un error de ejecución es una situación anormal, si se considera el proceso de ejecución de una parte específica de un programa de trabajo. Un error hace que la continuación de la ejecución sea imposible (o por lo menos peligrosa). “Desbordamiento” y “división por cero” son ejemplos de errores. Los errores son identificados por su único *número de error* que el robot reconoce siempre. La ocurrencia de un error provoca la suspensión de la ejecución normal del programa y el control pasa a un *gestor de errores*. El concepto del gestor de errores hace que sea posible proporcionar una respuesta y posiblemente recuperar la situación de error que ha aparecido durante la ejecución del programa. En el caso en que no se pueda proseguir con la ejecución del programa, el gestor de errores podrá por lo menos asegurar que la interrupción del programa sea suave.

8.1 Gestores de error

Cualquier rutina puede incluir un gestor de errores. El gestor de errores forma realmente parte de la rutina y el alcance de cualquier dato de rutina comprende también el gestor de errores de la rutina. En el caso en que ocurra un error durante la ejecución de la rutina, el control será transferido a su gestor de errores.

Ejemplo:

```

FUNC num divsegur( num x, num y)
    RETURN x / y;
ERROR
    IF ERRNO = ERR_DIVZERO THEN
        TPWrite "El número no puede ser igual a 0";
        RETURN x;
    ENDIF
ENDFUNC

```

La variable del sistema *ERRNO* contiene el código de error del error más reciente y será utilizado por el gestor de errores para poder identificarlo. Después de que se hayan tomado todas las medidas necesarias, el gestor de errores podrá:

- Proseguir con la ejecución, empezando por la instrucción en la que se ha producido el error. Esto se llevará a cabo utilizando la instrucción RETRY. En el caso en que esta instrucción vuelva a provocar el mismo error, se producirán hasta cuatro recuperaciones del error; después de ello, la ejecución se detendrá.
- Continuar la ejecución, empezando con la instrucción siguiente a la instrucción en la que ha ocurrido el error. Esto se consigue utilizando la instrucción TRY-NEXT.
- Devolver el control al que ha llamado la rutina utilizando la instrucción RETURN. En el caso en que la rutina sea una función, la instrucción RETURN deberá especificar un valor de retorno apropiado.
- Propagar el error al que ha llamado la rutina utilizando la instrucción RAISE.

Cuando ocurre un error en una rutina que no contiene ningún gestor de errores o cuando se alcanza el final del gestor de errores (**ENDFUNC**, **ENDPROC** o **ENDTRAP**), el

gestor de errores del sistema es llamado. El gestor de errores del sistema sólo registrará el error y detendrá la ejecución.

En una cadena de llamadas de rutinas, cada rutina puede tener su propio gestor de errores. En el caso en que ocurra un error en una rutina que dispone de un gestor de errores y si el error ha sido propagado de forma explícita mediante la instrucción RAISE, volverá a aparecer el mismo error en el punto que se llamó la rutina -el error ha sido *propagado*. Si se alcanza el principio de una cadena de llamada (la rutina de entrada de la tarea) sin encontrar ningún gestor de errores, o si se alcanza el final de cualquier gestor de errores dentro de la cadena de llamada, se llamará al *gestor de errores del sistema*. El gestor de errores del sistema sólo registrará el error y detendrá la ejecución. Dado que una rutina de tratamiento de interrupciones sólo podrá ser llamada por el sistema (como una respuesta a una interrupción), cualquier propagación de un error desde una rutina de tratamiento de interrupciones se realizará al gestor de errores del sistema.

La recuperación de errores no estará disponible para las instrucciones que se encuentran en el gestor de ejecución hacia atrás. Estos tipos de errores deberán ser siempre enviados al gestor de errores del sistema.

Además de los errores detectados y generados por el robot, un programa puede, de forma explícita, provocar errores mediante la instrucción RAISE. Esta utilidad podrá utilizarse para resolver situaciones muy complejas. Podrá utilizarse, por ejemplo, para salir de posiciones profundamente anidadas, es decir, de situaciones en que se han llamado a subrutinas dentro de subrutinas de forma excesiva. Se podrá utilizar cualquier número de error entre 1 y 90 en una instrucción RAISE. Los errores suscitados de forma explícita son procesados exactamente de la misma forma que los errores provocados por el sistema.

Obsérvese, no obstante, que no será posible resolver o responder a errores que ocurren dentro de una cláusula de error. Este tipo de errores son siempre enviados al gestor de errores del sistema.

9 Interrupciones

Las interrupciones son eventos definidos por el programa, identificados mediante los *números de interrupción*. Una interrupción se produce cuando se da una *condición de interrupción*. Al contrario de los errores, la ocurrencia de una interrupción no está directamente relacionada con (sincronizada con) una posición de una instrucción específica. La ocurrencia de una interrupción provoca una suspensión de la ejecución normal del programa y el control es transferido a una *rutina de tratamiento de interrupciones*.

Aunque el robot reconozca inmediatamente la ocurrencia de una interrupción (el único retraso será debido a la velocidad del controlador), su respuesta -llamada a la rutina de tratamiento de interrupciones correspondiente- sólo podrá tener lugar en unas posiciones específicas del programa, que son las siguientes:

- cuando se entra en la siguiente instrucción,
- en cualquier momento durante la ejecución de una instrucción de espera, por ejemplo, *WaitUntil*,
- en cualquier momento durante la ejecución de una instrucción de movimiento, por ejemplo, *MoveL*.

Esto suele provocar un retraso que oscila entre 5 y 120 ms entre el reconocimiento de la interrupción y la respuesta, según el tipo de movimiento que se está realizando en el momento de la interrupción.

La generación de interrupciones podrá ser *inhabilitada* y *habilitada*. Si se inhabilitan las interrupciones, todas las que se produzcan serán puestas en una cola y no aparecerán hasta que se vuelvan a habilitar de nuevo. Observar que la cola de interrupciones puede contener más de una interrupción en espera. Las interrupciones de una cola aparecerán siguiendo un orden *FIFO* (la primera que entra es la primera que sale). Las interrupciones están siempre inhabilitadas durante la ejecución de una rutina de tratamiento de interrupciones.

Cuando se realiza una ejecución paso a paso y cuando el programa ha sido detenido, las interrupciones no serán procesadas. Las interrupciones que se producen bajo estas circunstancias no serán tratadas.

El número máximo de interrupciones simultáneas es de 40 **por tarea**. La **limitación total** determinada por la CPU de E/S es de 100 interrupciones.

9.1 Procesamiento de las interrupciones

El robot reconocerá una interrupción gracias a su definición. La definición especifica la condición de la interrupción y la habilita.

Ejemplo:

```
VAR intnum sig1int;  
ISignalDI di1, alto, sig1int;
```

Una interrupción habilitada puede posteriormente ser inhabilitada (y viceversa).

Ejemplo:	ISleep sig1int;	inhabilitada
	IWatch sig1int;	habilitada

Cuando se borra una interrupción desaparece también su definición. No es necesario borrar de forma explícita una definición de interrupción, pero no se podrá definir una interrupción nueva a una variable de interrupción hasta que se haya borrado la definición precedente.

Ejemplo: IDDelete sig1int;

9.2 Rutinas de tratamiento de las interrupciones

Las rutinas de tratamiento de las interrupciones proporcionan un medio para procesar las interrupciones. Una rutina de tratamiento de interrupciones podrá ser conectada a una interrupción particular mediante la instrucción CONNECT. En el caso en que ocurra una interrupción, el control es inmediatamente transferido a la rutina de tratamiento de interrupciones asociadas (siempre y cuando haya una). Si ocurre una interrupción que no ha sido conectada a ninguna rutina de tratamiento de interrupciones, la interrupción será considerada como un error fatal, es decir, que provoca inmediatamente el paro de la ejecución del programa.

Ejemplo: VAR intnum vacío;
VAR intnum lleno;

.PROC main()

CONNECT vacío WITH etrap;

conexión de rutina tratamiento de interrupciones

CONNECT lleno WITH ftrap;
ISignalDI di1, alto, vacío;

define las interrupciones del alimentador

ISignalDI di3, alto, lleno;

IDelete vacío;

IDelete lleno;

ENDPROC

TRAP etrap
abrir_valv ;
RETURN;
ENDTRAP

respuesta a la interrupción “alimentador vacío”

TRAP ftrap
cerrar_valv;
RETURN;
ENDTRAP

respuesta a la interrupción “alimentador lleno”

Varias interrupciones podrán ser conectadas a la misma rutina de tratamiento de interrupciones. La variable del sistema *INTNO* contiene el número de interrupción y podrá ser utilizada por una rutina de tratamiento de interrupciones para la identificación de la interrupción. Después de haber tomado las medidas necesarias, se podrá finalizar

Interrupciones

Características Básicas RAPID

una rutina de tratamiento de interrupciones utilizando la instrucción RETURN o cuando se alcanza el final (ENDTRAP o ERROR) de la rutina de tratamiento de interrupciones. La ejecución continua a partir del lugar en que ocurrió la interrupción.

10 Ejecución hacia atrás

Un programa puede ser ejecutado hacia atrás, una instrucción a la vez. Las siguientes restricciones generales se aplican a la ejecución hacia atrás:

- Las instrucciones IF, FOR, WHILE y TEST no pueden ser ejecutadas hacia atrás.
- No se podrá salir hacia atrás de una rutina, cuando se alcance el principio de la rutina.

10.1 Gestores de ejecución hacia atrás

Los procedimientos pueden contener un gestor de ejecución hacia atrás que define la ejecución hacia atrás de una llamada de procedimiento.

El gestor de ejecución hacia atrás constituye realmente una parte del procedimiento y el alcance de cualquier dato de rutina también comprende el gestor de ejecución hacia atrás del procedimiento.

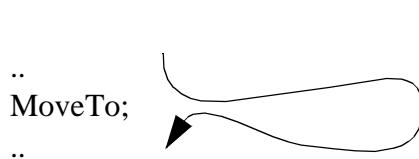
Ejemplo:

```

PROC MoveTo ()
    MoveL p1,v500,z10,tool1;
    MoveC p2,p3,v500,z10,tool1;
    MoveL p4,v500,z10,tool1;
    BACKWARD
    MoveL p4,v500,z10,tool1;
    MoveC p2, p3,v500,z10,tool1;
    MoveL p1,v500,z10,tool1;
ENDPROC

```

Cuando el procedimiento es llamado durante una ejecución hacia adelante, ocurre lo siguiente:

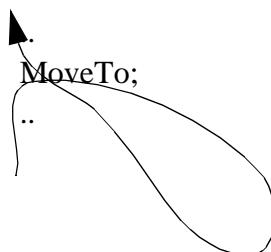


```

PROC MoveTo ()
    MoveL p1,v500,z10,tool1;
    MoveC p2,p3,v500,z10,tool1;
    MoveL p4,v500,z10,tool1;
    BACKWARD
    MoveL p4,v500,z10,tool1;
    MoveC p2, p3,v500,z10,tool1;
    MoveL p1,v500,z10,tool1;
ENDPROC

```

Cuando el procedimiento es llamado durante la ejecución hacia atrás, ocurre lo siguiente:



PROC MoveTo ()

MoveL p1,v500,z10,tool1;
MoveC p2,p3,v500,z10,tool1;
MoveL p4,v500,z10,tool1;

BACKWARD

MoveL p4,v500,z10,tool1;
MoveL p2,p3,v500,z10,tool1;
MoveL p1,v500,z10,tool1;

ENDPROC

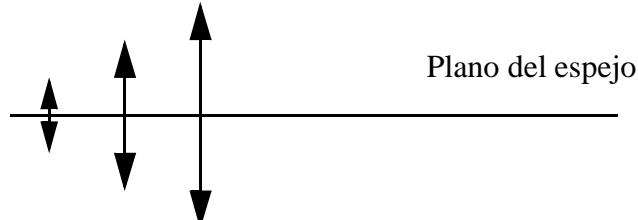
Las instrucciones que se encuentran en el gestor de errores o en el gestor de ejecución hacia atrás de una rutina no podrán ser ejecutadas hacia atrás. La ejecución hacia atrás no podrá ser anidada, es decir que dos instrucciones en una cadena de llamada no podrán ser ejecutadas simultáneamente hacia atrás.

Un procedimiento sin gestor de ejecución hacia atrás no podrá ser ejecutado hacia atrás. Un procedimiento con un gestor de ejecución hacia atrás vacío será ejecutado como «ninguna operación».

10.2 Limitación de instrucciones de movimiento en el gestor de ejecución hacia atrás

El tipo de instrucción de movimiento y la secuencia en el gestor de ejecución hacia atrás debe constituir un espejo del tipo de instrucción de movimiento y de la secuencia para la ejecución hacia adelante dentro de la misma rutina:

PROC MoveTo ()
MoveL p1,v500,z10,tool1;
MoveC p2,p3,v500,z10,tool1;
MoveL p4,v500,z10,tool1;
BACKWARD
MoveL p4,v500,z10,tool1;
MoveC p2,p3,v500,z10,tool1;
MoveL p1,v500,z10,tool1;
ENDPROC



Observar que el orden de CirPoint $p2$ y ToPoint $p3$ dentro de la instrucción MoveC debe ser el mismo.

Por instrucción de movimiento se entiende cualquier instrucción que implique algún movimiento del robot o de los ejes externos, como por ejemplo MoveL, SearchC, TriggJ, ArcC, PaintL ...



Cualquier variación de esta limitación de la programación en el gestor de ejecución hacia atrás puede provocar un movimiento hacia atrás defectuoso. Un movimiento lineal puede convertirse en un movimiento circular y viceversa, en algún tramo de la trayectoria hacia atrás.

11 Multitareas

Así como los eventos en una célula robot se producen en paralelo, de la misma forma, los programas pueden ejecutarse en paralelo.

La función **Multitarea RAPID** sirve para ejecutar programas de una forma pseudo paralela respecto a la ejecución normal. La ejecución arranca al activar el sistema y seguirá funcionando indefinidamente a menos que ocurra un error en ese programa. Un programa paralelo puede situarse en el primer plano o en el segundo plano respecto a otro programa. También puede estar en el mismo nivel que otro programa.

Para utilizar esta función, el robot deberá ser configurado con una TAREA adicional para cada programa de segundo plano.

Se podrán ejecutar hasta 10 tareas diferentes en pseudo paralelo. Cada tarea está formada por un conjunto de módulos, de la misma forma que el programa normal. Todos los módulos son locales para cada tarea.

Los datos variables y constantes son locales para cada tarea, pero los persistentes no. Un dato persistente con el mismo nombre y del mismo tipo es alcanzable en todas las tareas. En el caso en que dos datos persistentes tengan el mismo nombre y que el tipo o el tamaño (dimensión de matriz) difieran, se producirá un error de tiempo de ejecución.

Una tarea tiene su propia gestión de rutinas de tratamiento de interrupciones y las rutinas de evento se disparan únicamente en sus propios estados del sistema (por ejemplo, Inicio/Paro/Reinicio).

No obstante la función Multitareas RAPID tiene una serie de limitaciones:

- No se deberá mezclar un programa paralelo con un PLC. El tiempo de respuesta es el mismo que el tiempo de respuesta de interrupción para cada tarea. Esto se aplica, por supuesto, siempre y cuando la tarea no esté situada en segundo plano de otro programa que está funcionando.
- Sólo se dispondrá de una unidad de programación física, por lo tanto, se deberá tener en cuenta que la petición de escritura (TPWrite) de la unidad de programación no se mezcle en la ventana del usuario común a todas las tareas.
- Cuando se ejecuta una instrucción de espera en el modo manual, aparecerá una ventana de simulación al cabo de 3 segundos. Esto sólo ocurrirá en la tarea principal.
- Las instrucciones de movimiento sólo podrán ejecutarse en la tarea principal (Véase la Guía del Usuario - Parámetros del Sistema).
- La ejecución de una tarea quedará interrumpida durante el tiempo en que algunas otras tareas están accediendo al sistema de archivo, es decir, si el operador elige salvar o abrir un programa, o si el programa en una tarea utiliza las instrucciones de cargar/borrar/leer/escribir.
- La unidad de programación no podrá tener acceso a otras tareas que la tarea principal. Por ello, el desarrollo del programa RAPID para otras tareas sólo podrá llevarse a cabo si se ha cargado el código en la tarea principal o bien en off-line.

Para cualquier ajuste de parámetros, véase la Guía del Usuario - Parámetros del Sistema.

11.1 Sincronización de las tareas

En muchas aplicaciones una tarea paralela sólo podrá supervisar una unidad, independientemente de la ejecución de las demás tareas. En este caso no se necesitará ningún mecanismo de sincronización. Sin embargo, existen otras aplicaciones que por ejemplo, requieren el conocimiento de lo que está realizando la tarea principal.

Sincronización utilizando la interrogación

Esta es la forma más sencilla de realizar la sincronización, pero es muy lenta.

Los datos persistentes serán utilizados junto con las instrucciones *WaitUntil*, *IF*, *WHILE* o *GOTO*.

En el caso en que la instrucción *WaitUntil* sea utilizada, el sistema realizará una interrogación interna cada 100 ms; no obstante el sistema no será capaz de interrogar con una frecuencia más elevada en otras aplicaciones.

Ejemplo

TAREA 0

MODULE module1

```
PERS bool startsync:=FALSE;  
PROC main()
```

```
    startsync:= TRUE;
```

```
.
```

```
ENDPROC  
ENDMODULE
```

TAREA 1

```
MODULE module2  
PERS bool startsync:=FALSE;  
PROC main()
```

```
    WaitUntil startsync;
```

```
.
```

```
ENDPROC  
ENDMODULE
```

Sincronización utilizando una interrupción

Se utilizarán las instrucciones *SetDO* y *ISignalDO*.

Ejemplo

TAREA 0

```
MODULE module1  
PROC main()
```

```
    SetDO do1,1;
```

```
.
```

```
ENDPROC  
ENDMODULE
```

TAREA 1

```
MODULE module2  
VAR intnum isiint1;  
PROC main()
```

```
    CONNECT isiint1 WITH isi_trap;  
    ISignalDO do1, 1, isiint1;
```

```
    WHILE TRUE DO  
        WaitTime 200;  
    ENDWHILE
```

```
    IDElete isiint1;
```

```
ENDPROC
```

```
TRAP isi_trap
```

```
ENDTRAP
```

```
ENDMODULE
```

11.2 Comunicación entre tareas

Todos los tipos de datos pueden ser enviados entre dos (o más) tareas con variables persistentes.

Una variable persistente es global en todas las tareas. La variable persistente deberá ser del mismo tipo y tamaño (dimensión de matriz) en todas las tareas que lo declaran. De lo contrario ocurrirá un error de tiempo de funcionamiento.

Todas las declaraciones deberán especificar un valor de inicio a la variable persistente, pero solo será usado por el primer módulo cargado con la declaración.

Ejemplo

TAREA 0

```
MODULE module1
PERS bool startsync:=FALSE;
PERS string stringtosend:="";
PROC main()

    stringtosend:="this is a test";

    startsync:= TRUE

ENDPROC
ENDMODULE
```

TAREA 1

```
MODULE module2
PERS bool startsync:=FALSE;
PERS string stringtosend:="";
PROC main()

    WaitUntil startsync;

    !read string
    IF stringtosend = "this is a test" THEN

ENDPROC
ENDMODULE
```

11.3 Tipo de tarea

Cada tarea adicional (no 0) es arrancada en la secuencia de arranque del sistema. En el caso en que la tarea sea del tipo STATIC será rearrancada en la posición actual (es decir a partir de donde se encontraba el indicador de punto de arranque del programa (PP) cuando el sistema quedó desactivado) pero en cambio, si el tipo está puesto en SEMISTATIC será rearrancado desde el principio, cada vez que se active el sistema.

También se podrá activar la tarea en el tipo NORMAL, y entonces tendrá el mismo comportamiento que la tarea 0 (la tarea principal, controlando el movimiento del robot). Desde la unidad de programación no se podrá arrancar ninguna tarea excepto la tarea 0, con lo cual, la única manera de arrancar otra tarea NORMAL será utilizando el software de comunicación.

11.4 Prioridades

La forma de ejecutar las tareas por defecto consiste en ejecutar todas las tareas al mismo nivel de forma progresiva, como en una rueda, (es decir, que se da un paso básico en cada ejemplo). Siempre se podrá cambiar la prioridad de una tarea colocando dicha tarea en segundo plano respecto a otra. Entonces, el segundo plano se ejecuta únicamente cuando el primer plano está esperando algún evento o cuando ha detenido su ejecución (está parado). Un programa robot con instrucciones de movimiento estará parado la mayor parte del tiempo.

El ejemplo siguiente describe una situación en que el sistema tiene 10 tareas. (véase la Figura 5)

- | | |
|--------------------|--|
| Cadena en rueda 1: | tareas 0, 1, y 8 están funcionando |
| Cadena en rueda 2: | tareas 0, 3, 4, 5 y 8 están funcionando
tareas 1 y 2 están paradas |
| Cadena en rueda 3: | tareas 2, 4 y 5 están funcionando
tareas 0, 1, 8 y 9 están paradas |
| Cadena en rueda 4: | tareas 6 y 7 están funcionando
tareas 0, 1, 2, 3, 4, 5, 8 y 9 están paradas |

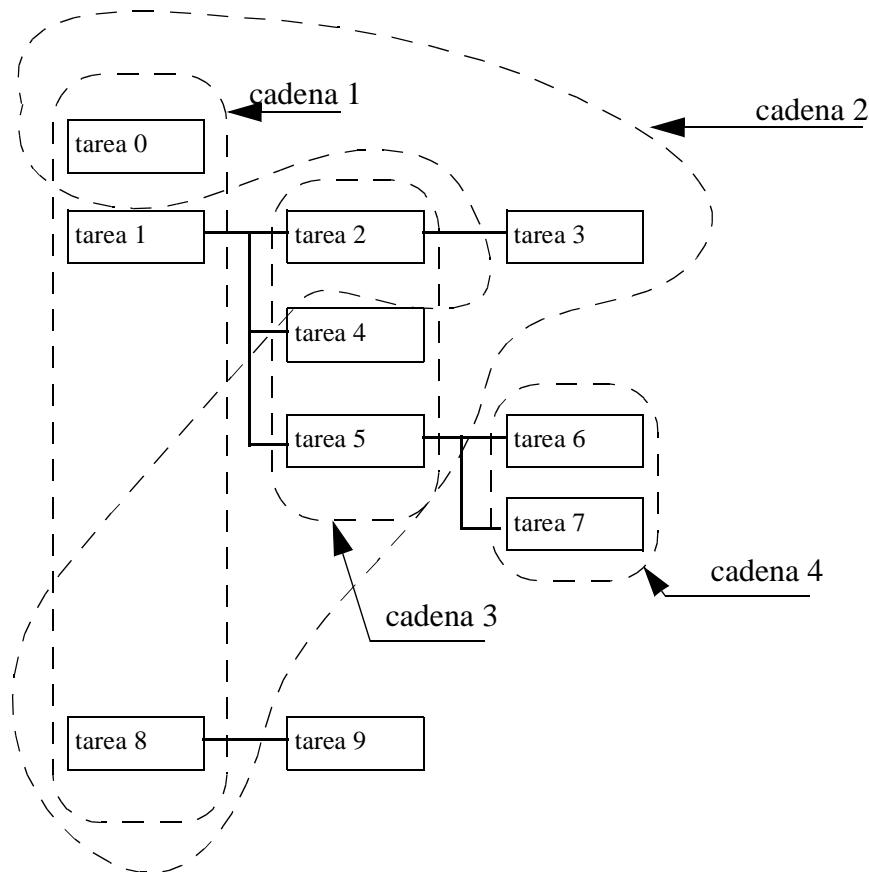


Figura 5 Las tareas pueden tener distintas prioridades.

11.5 Tamaño de las tareas

El sistema suministra una área de memoria con una instalación que depende del tamaño. Esta área es compartida por todas las tareas.

Si la memoria de una tarea ha sido definida en los parámetros del sistema, le será atribuida una cierta cantidad de memoria.

Todas las tareas sin especificación de tamaños compartirán la parte libre del área proporcionada de la siguiente forma. Si el tamaño de la tarea principal no ha sido especificada, se le atribuirá el porcentaje del área libre que está especificada en el parámetro *MemoryTask0* (véase Parámetros del Sistema). El resto del área libre quedará dividida entre las demás tareas de forma equitativa.

El valor de una variable persistente quedará almacenada en una parte separada del sistema, y no afectará el área de la memoria anterior. Véase Parámetros del Sistema.

11.6 Recomendación importante

En el momento de especificar las prioridades de las tareas, el usuario deberá recordar lo siguiente:

- Utilizar siempre el mecanismo de interrupción o bucles con retardos en las tareas de supervisión. De lo contrario, la unidad de programación no tendrá el tiempo suficiente para la interacción con el usuario. Y en el caso en que las tareas de supervisión estén en primer plano, el sistema no permitirá que ninguna otra tarea sea ejecutada en segundo plano.

Principios de Movimiento y de E/S

INDICE

	Página
1 Sistemas de Coordenadas	3
1.1 El punto central de la herramienta del robot (TCP).....	3
1.2 Sistemas de coordenadas utilizados para determinar la posición del TCP	3
1.2.1 Sistema de coordenadas de la base	3
1.2.2 Sistema de coordenadas mundo	4
1.2.3 Sistema de coordenadas del usuario.....	5
1.2.4 Sistema de coordenadas del objeto	5
1.2.5 Sistema de coordenadas de desplazamiento.....	6
1.2.6 Ejes externos coordinados.....	7
1.3 Sistemas de coordenadas utilizados para determinar la dirección de la herramienta	8
1.3.1 Sistema de coordenadas de la muñeca	9
1.3.2 Sistema de coordenadas de la herramienta	9
1.3.3 TCPs estacionarios	10
1.4 Información relacionada	12
2 Posicionamiento durante la Ejecución del Programa.....	13
2.1 Generalidades	13
2.2 Interpolación de la posición y orientación de la herramienta	13
2.2.1 Interpolación eje a eje	13
2.2.2 Interpolación lineal	14
2.2.3 Interpolación circular	15
2.2.4 Sing Area\Wrist.....	16
2.3 Interpolación de las trayectorias esquina.....	16
2.3.1 Interpolación eje a eje en las trayectorias esquina	17
2.3.2 Interpolación lineal de una posición en las trayectorias esquina	18
2.3.3 Interpolación lineal de la orientación en las trayectorias esquina.....	19
2.3.4 Interpolación de los ejes externos en trayectorias esquina	19
2.3.5 Trayectorias esquina cuando se cambia el método de interpolación	20
2.3.6 Interpolación al cambiar de sistema de coordenadas	20
2.3.7 Trayectorias esquina con zonas que se superponen	20
2.3.8 Especificación del tiempo en la programación de los puntos de paso	21
2.4 Ejes independientes	22
2.4.1 Ejecución del programa	22
2.4.2 Ejecución paso a paso	23
2.4.3 Movimiento	23
2.4.4 Area de trabajo	23
2.4.5 Velocidad y aceleración.....	24
2.4.6 Ejes del robot	24

Principios de Movimiento y de E/S

2.5 Servo Suave	25
2.6 Paro y rearanque.....	25
2.7 Información relacionada.....	26
3 Sincronización con Instrucciones Lógicas	27
3.1 Ejecución secuencial del programa en los puntos de paro	27
3.2 Ejecución secuencial del programa en los puntos de paso	27
3.3 Ejecución concurrente del programa	28
3.4 Sincronización de la trayectoria	30
3.5 Información relacionada.....	31
4 Configuración del Robot	32
4.1 Datos para la configuración del robot 6400C.....	34
4.2 Información relacionada.....	35
5 Puntos Singulares.....	36
Figura 39 Puntos singulares/IRB 6400C	37
5.1 Ejecución del programa con puntos singulares	37
5.2 Movimiento con puntos singulares.....	37
5.3 Información relacionada.....	38
6 Zonas Mundo.....	39
6.1 Utilización de las Zonas	39
6.2 Utilidad de las Zonas Mundo	39
6.3 Definición de las Zonas Mundo en el sistema de coordenadas mundo	39
6.4 Supervisión del TCP del robot	40
6.4.1 TCPs estacionarios.....	40
6.5 Acciones	41
6.5.1 Activación de una salida digital cuando el TCP está dentro de una Zona Mundo	41
6.5.2 Activación de una salida digital antes de que el TCP alcance una Zona Mundo	41
6.5.3 Paro del robot antes de que el TCP alcance una Zona Mundo	41
6.6 Tamaño mínimo de las Zonas Mundo	42
6.7 Reinicio después de un corte del suministro de alimentación.....	42
6.8 Información relacionada.....	43
7 Principios de E/S	44
7.1 Características de las señales.....	44
7.2 Señales del sistema.....	45
7.3 Conexiones enlazadas.....	45
7.4 Limitaciones	46
7.5 Información relacionada.....	47

1 Sistemas de Coordenadas

1.1 El punto central de la herramienta del robot (TCP)

Tanto la posición del robot como todos sus movimientos se refieren siempre al punto central de la herramienta. Este punto suele estar definido como estando situado en alguna parte de la herramienta, por ejemplo, en la boca de una pistola de aplicación de adhesivo, en el centro de una pinza o en el extremo de una varilla fija terminada en punta.

Se podrán definir varios TCPs (herramientas), pero sólo podrá haber un TCP activado a la vez. Cuando se registra una posición, lo que se registra es la posición del TCP. Es también el punto que se mueve a lo largo de una trayectoria específica, a una velocidad determinada.

Si el robot está sujetando una pieza y que está trabajando en una herramienta estacionaria, se deberá utilizar un TCP estacionario. Si esa herramienta está activa, la trayectoria y la velocidad programadas se referirán al objeto de trabajo. Véase el apartado sobre *TCPs estacionarios* de la página 10.

1.2 Sistemas de coordenadas utilizados para determinar la posición del TCP

La posición de la herramienta (TCP) puede ser especificada en distintos sistemas de coordenadas para facilitar la programación y el reajuste de los programas.

El sistema de coordenadas definido dependerá de la tarea que debe realizar el robot. Cuando ningún sistema de coordenadas ha sido definido, las posiciones del robot serán definidas en el sistema de coordenadas de la base.

1.2.1 Sistema de coordenadas de la base

En una aplicación simple, la programación puede realizarse en el sistema de coordenadas de la base; en este ejemplo, el eje z- coincide con el eje 1 del robot (véase la Figura 1).

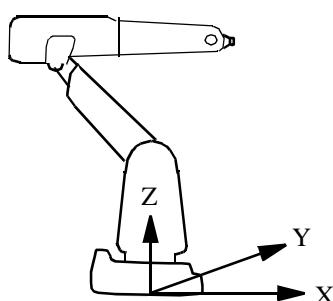


Figura 1 El sistema de coordenadas de la base.

El sistema de coordenadas de la base se sitúa en la base del robot:

- *El origen* está situado en la intersección del eje 1 y la superficie de montaje de la base.
- *El plano xy* es el mismo que la superficie de montaje de la base.
- *El eje x-* apunta hacia adelante.
- *El eje y-* apunta hacia la izquierda (desde la perspectiva del robot).
- *El eje z-* apunta hacia arriba.

1.2.2 Sistema de coordenadas mundo

Si el robot ha sido montado en el suelo, la programación en el sistema de coordenadas de la base resultará fácil. Si, de lo contrario, se ha realizado un montaje suspendido del robot, la programación en el sistema de coordenadas de la base resultará más difícil debido a que las direcciones de sus ejes no son las mismas que las direcciones principales del área de trabajo. En estos casos, se recomienda la definición de un sistema de coordenadas mundo. El sistema de coordenadas mundo coincidirá con el sistema de coordenadas de la base si no se ha definido específicamente.

En ciertas ocasiones, varios robots pueden estar trabajando dentro de la misma área de trabajo en una planta. En estos casos, se deberá utilizar un sistema de coordenadas mundo común para permitir que los programas de robot puedan comunicarse entre sí. También puede resultar ventajoso utilizar este tipo de coordenadas cuando las posiciones deban referirse a un punto fijo en el taller. Véase el ejemplo de la Figura 2.

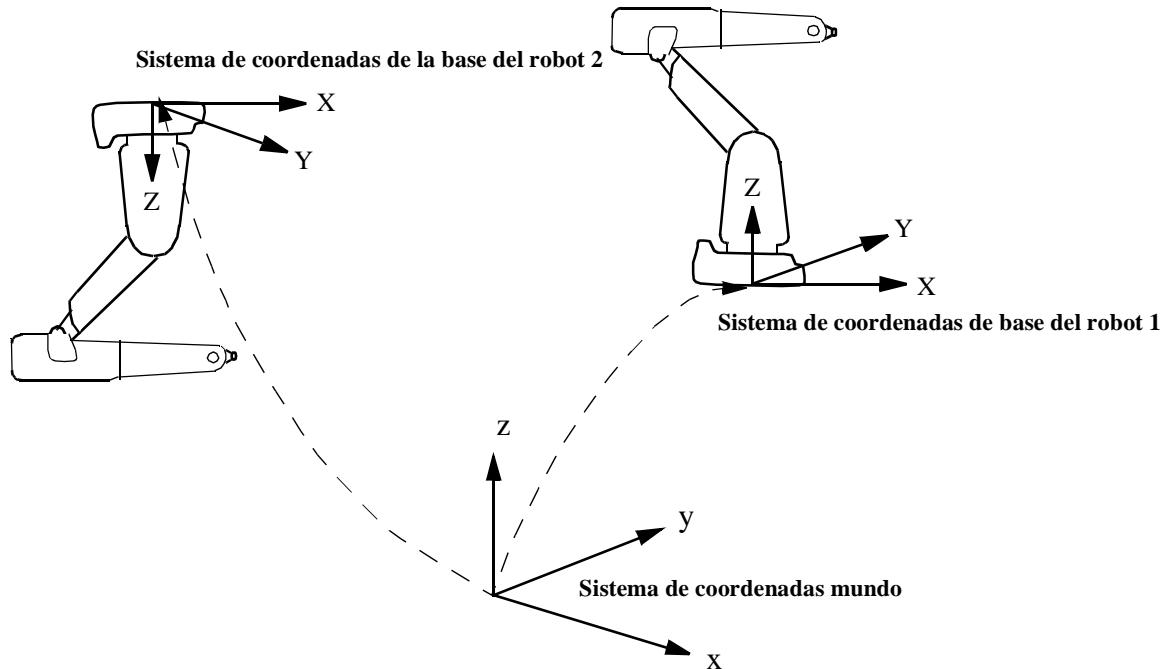


Figura 2 Dos robots (uno de ellos es de montaje suspendido) con un sistema de coordenadas mundo común.

1.2.3 Sistema de coordenadas del usuario

Un robot puede trabajar con diferentes útiles o superficies de trabajo que tienen distintas posiciones y distintas orientaciones. Se podrá definir para cada útil un sistema de coordenadas del usuario. En el caso en que todas las posiciones estén almacenadas en coordenadas del objeto, el usuario no tendrá que volver a programar si el útil debe ser movido o girado. Sólo con mover/girar el sistema de coordenadas del usuario en la misma medida en que el útil ha sido movido/girado, todas las posiciones programadas seguirán el útil y no se requerirá ninguna programación nueva.

El sistema de coordenadas del usuario se definirá basándose en el sistema de coordenadas mundo (véase la Figura 3).

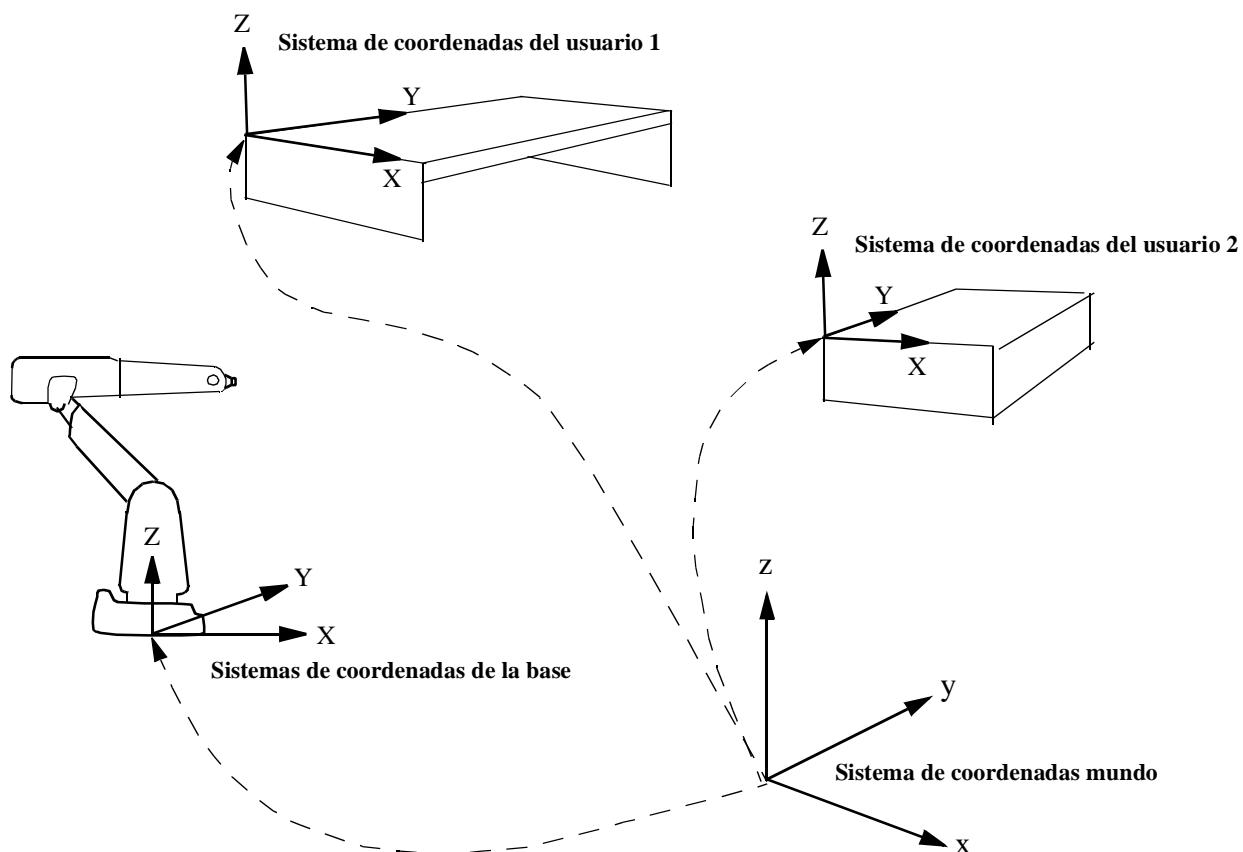


Figura 3 Dos sistemas de coordenadas del usuario que describen la posición de dos útiles diferentes.

1.2.4 Sistema de coordenadas del objeto

El sistema de coordenadas del usuario sirve para utilizar distintos sistemas de coordenadas según las diferentes útiles o superficies de trabajo. Un útil, no obstante, podrá incluir varios objetos de trabajo que deben ser procesados o manipulados por el robot. De esta manera, a menudo resultará útil definir un sistema de coordenadas para cada objeto, para facilitar el ajuste del programa si se ha movido el objeto o si se desea programar un objeto nuevo, el mismo que el anterior, pero en un lugar diferente. Un sistema de coordenadas referido a un objeto se denomina un sistema de coordenadas del objeto. Este sistema de coordenadas es muy adecuado también para la programación off-line, ya que las posiciones especificadas pueden sacarse directamente de un dibujo del objeto de trabajo. El sistema de coordenadas del objeto también podrá utilizarse para mover el robot.

El sistema de coordenadas del objeto será definido basándose en el sistema de coordenadas del usuario (véase la Figura 4).

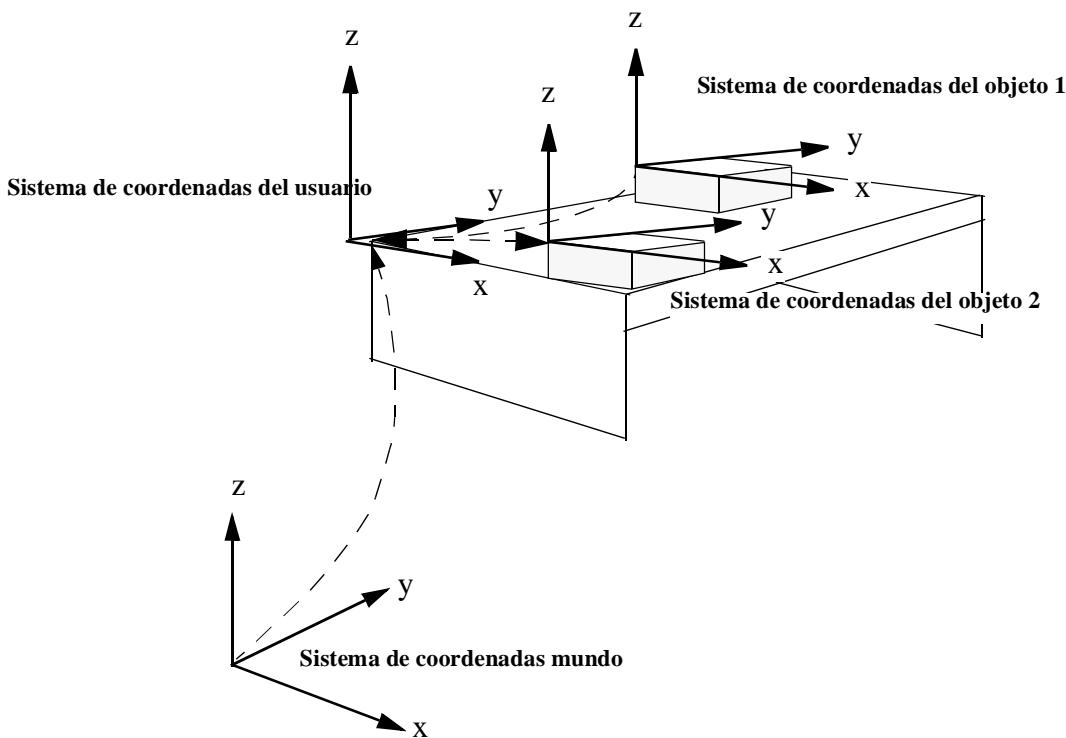


Figura 4 Dos sistemas de coordenadas del objeto que describen la posición de dos objetos de trabajo diferentes situados en un mismo soporte.

Las posiciones programadas están definidas siempre respecto a un sistema de coordenadas del objeto. El hecho de mover/girar el soporte, podrá ser compensado moviendo/girando el sistema de coordenadas del usuario. Así, no será necesario cambiar las posiciones programadas ni los sistemas de coordenadas definidos del objeto. Por lo tanto si se mueve/gira el objeto de trabajo, esto podrá ser compensado moviendo/girando el sistema de coordenadas del objeto.

En el caso en que el sistema de coordenadas del usuario sea móvil, es decir, que se utilicen ejes externos coordinados, entonces el sistema de coordenadas del objeto se moverá junto con el sistema de coordenadas del usuario. Esto hace posible que se pueda mover el robot respecto al objeto incluso cuando se está manipulando el banco de trabajo.

1.2.5 Sistema de coordenadas de desplazamiento

En algunas ocasiones, se deberá realizar la misma trayectoria en lugares diferentes del mismo objeto. Para evitar el volver a programar todas las posiciones cada vez, se definirá un sistema de coordenadas, conocido bajo el nombre de sistema de coordenadas de desplazamiento. Este sistema de coordenadas podrá utilizarse también conjuntamente con las búsquedas, para compensar las diferencias en las posiciones de las piezas individuales.

El sistema de coordenadas de desplazamiento se definirá basándose en el sistema de coordenadas del objeto (véase la Figura 5).

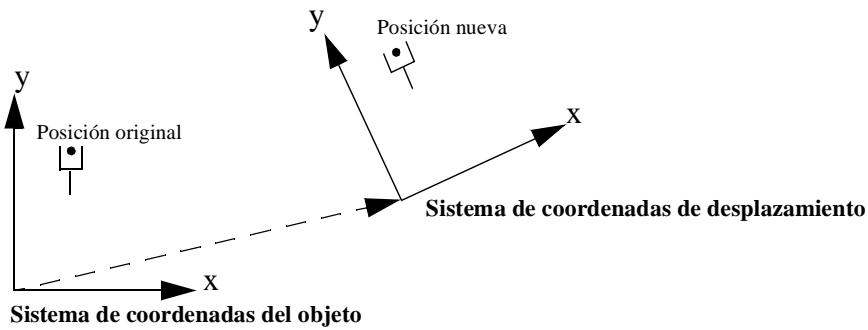


Figura 5 Si el desplazamiento de programa está activado, todas las posiciones serán desplazadas.

1.2.6 Ejes externos coordinados

Coordinación del sistema de coordenadas del usuario

Cuando un objeto de trabajo está situado en una unidad mecánica externa que se mueve mientras el robot está ejecutando una trayectoria definida en el sistema de coordenadas del objeto, se podrá definir un sistema de coordenadas móvil del usuario. La posición y la orientación del sistema de coordenadas del usuario, en este caso, dependerá de las rotaciones del eje de la unidad externa. La trayectoria y la velocidad programadas se referirán entonces al objeto de trabajo (véase la Figura 6) y no habrá ninguna necesidad de considerar el hecho de que el objeto es movido por la unidad externa.

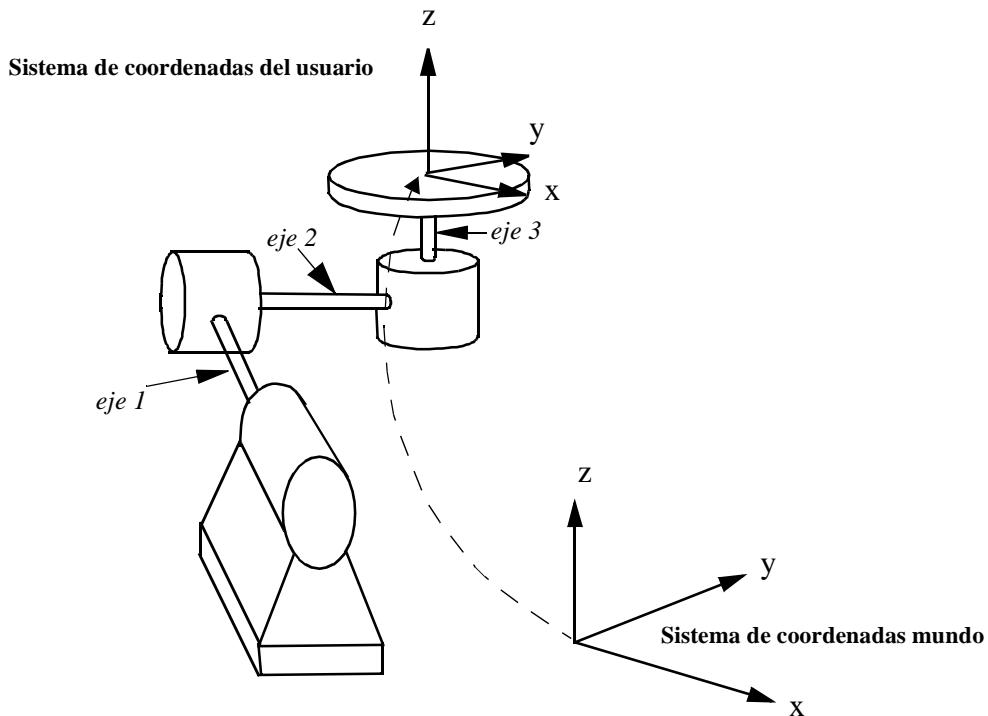


Figura 6 Un sistema de coordenadas del usuario definido para seguir los movimientos de la unidad mecánica externa del eje 3.

Coordinación del sistema de coordenadas en la base

También se podrá definir un sistema de coordenadas móvil para la base del robot. Esto ofrece especial interés en el momento de la instalación cuando se lleva a cabo el montaje del robot en un sistema de desplazamiento lineal o en un montaje en pórtico, por ejemplo. La posición y la orientación del sistema de coordenadas de la base, como para el sistema de coordenadas móvil del usuario, dependerán de los movimientos de la unidad externa. La trayectoria y la velocidad programadas se referirán al sistema de coordenadas del objeto (Figura 7) y por tanto no habrá que considerar el hecho de que la base del robot está movida por una unidad externa. Se podrá definir al mismo tiempo un sistema de coordenadas del usuario coordinado y un sistema de coordenadas de la base coordinado.

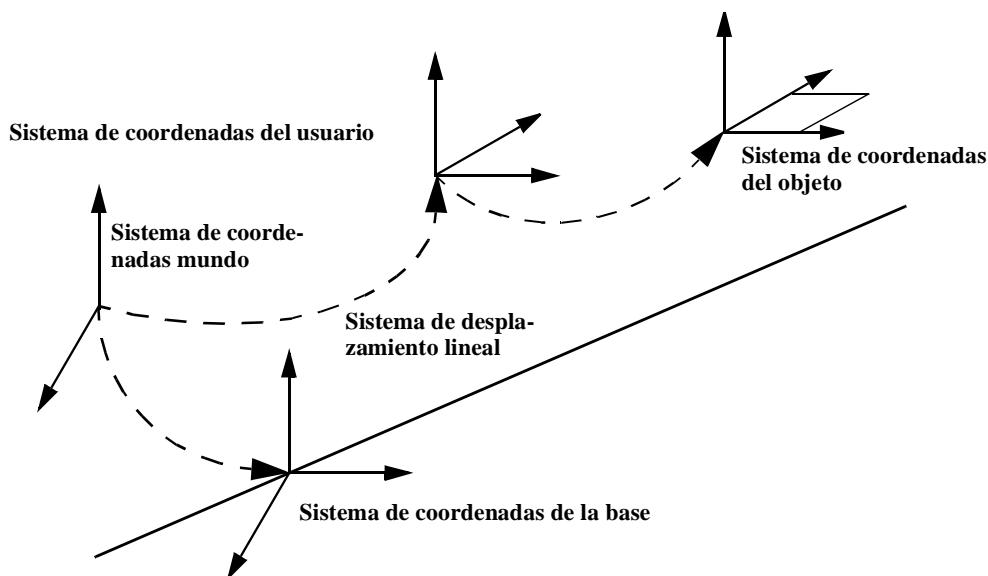


Figura 7 Interpolación coordinada con un sistema de desplazamiento lineal que mueve el sistema de coordenadas en la base del robot.

Para ser capaz de calcular el sistema de coordenadas del usuario y el sistema de coordenadas de la base, cuando las unidades implicadas son movidas por una unidad externa, el robot deberá conocer:

- Las posiciones de calibración del sistema de coordenadas del usuario, y del sistema de coordenadas de la base.
- Las relaciones entre los ángulos de los ejes externos y la traslación/rotación del sistema de coordenadas del usuario y del sistema de coordenadas de la base.

Estas relaciones están definidas en los parámetros del sistema.

1.3 Sistemas de coordenadas utilizados para determinar la dirección de la herramienta

La orientación de una herramienta en una posición programada es dada por la orientación del sistema de coordenadas de la herramienta. El sistema de coordenadas de la herramienta se refiere al sistema de coordenadas de la muñeca, definido en la brida de montaje de la muñeca del robot.

1.3.1 Sistema de coordenadas de la muñeca

En una aplicación sencilla, se utilizará el sistema de coordenadas de la muñeca para definir la orientación de la herramienta; en este caso, el eje z- coincidirá con el eje 6 del robot (véase la Figura 8).

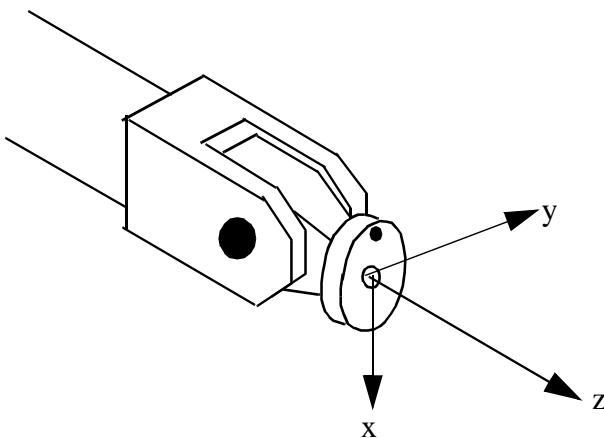


Figura 8 El sistema de coordenadas de la muñeca.

El sistema de coordenadas de la muñeca no podrá cambiarse y será siempre el mismo que la brida de montaje del robot, con las siguientes características:

- *El origen* está situado en el centro de la brida de montaje (en la superficie de montaje).
- *El eje x-* apunta hacia la dirección opuesta, hacia el orificio de control para el pasador de la brida de montaje.
- *El eje z-* apunta hacia afuera, perpendicularmente a la brida de montaje.

1.3.2 Sistema de coordenadas de la herramienta

La herramienta que ha sido montada en la brida de montaje del robot requiere a menudo su propio sistema de coordenadas para permitir la definición de su TCP, que es el origen del sistema de coordenadas de la herramienta. El sistema de coordenadas de la herramienta podrá utilizarse también para conseguir las direcciones de movimiento apropiadas al mover el robot.

En el caso en que se reemplace o se averíe una herramienta, lo que se deberá hacer es volver a definir el sistema de coordenadas de la herramienta. Normalmente no se deberá cambiar el programa.

El TCP (origen) será seleccionado como el punto en la herramienta que debe ser posicionado correctamente, por ejemplo, la boquilla de una pistola de aplicación de adhesivo. Los ejes de coordenadas de la herramienta serán definidos como siendo los ejes correspondientes a dicha herramienta.

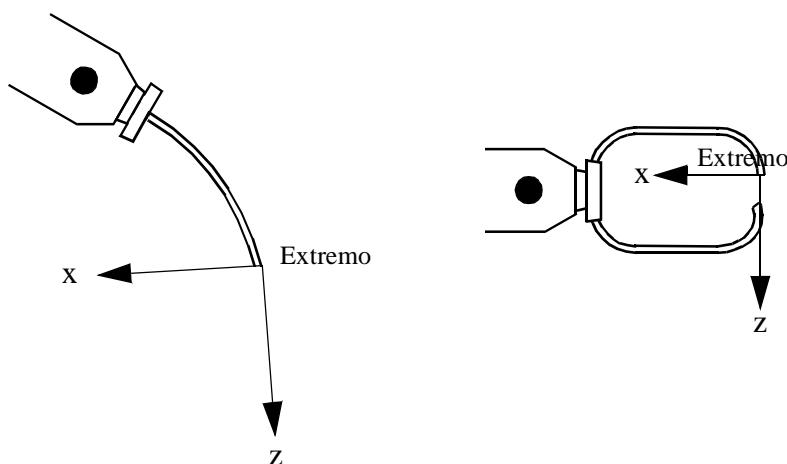


Figura 9 Sistema de coordenadas de la herramienta, tal como se suele definir para una pistola de soldadura al arco (izquierda) y para una pistola de soldadura por puntos (derecha).

El sistema de coordenadas de la herramienta se definirá basándose en el sistema de coordenadas de la muñeca (véase la Figura 10).

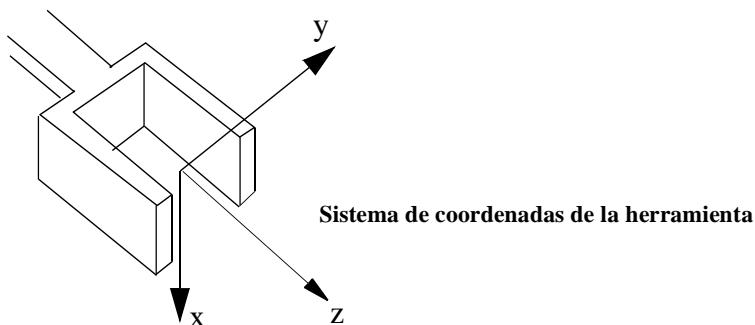


Figura 10 El sistema de coordenadas de la herramienta será definido respecto al sistema de coordenadas de la muñeca.

1.3.3 TCPs estacionarios

Cuando el robot está sujetando una pieza y trabaja con una herramienta estacionaria, se deberá utilizar un TCP estacionario. Si esta herramienta está activada, la trayectoria y la velocidad programadas se referirán al objeto de trabajo sujetado por el robot.

Esto significa que los sistemas de coordenadas serán invertidos, como se indica en la Figura 11.

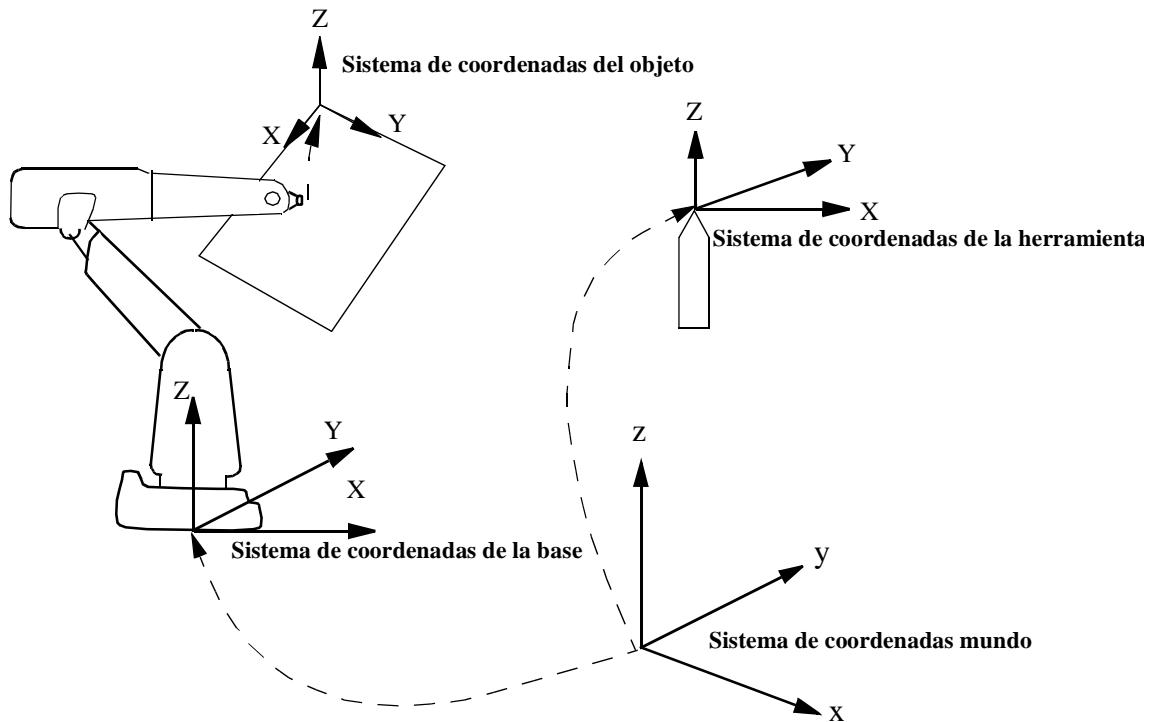


Figura 11 Cuando se utiliza un TCP estacionario, el sistema de coordenadas del objeto suele estar basado en el sistema de coordenadas de la muñeca.

En el ejemplo de la Figura 11, no se utiliza ningún sistema de coordenadas del usuario ni ningún sistema de coordenadas de desplazamiento de programa. No obstante, es posible utilizarlos y, en el caso en que lo sean, se referirán el uno al otro, como se indica en la Figura 12.

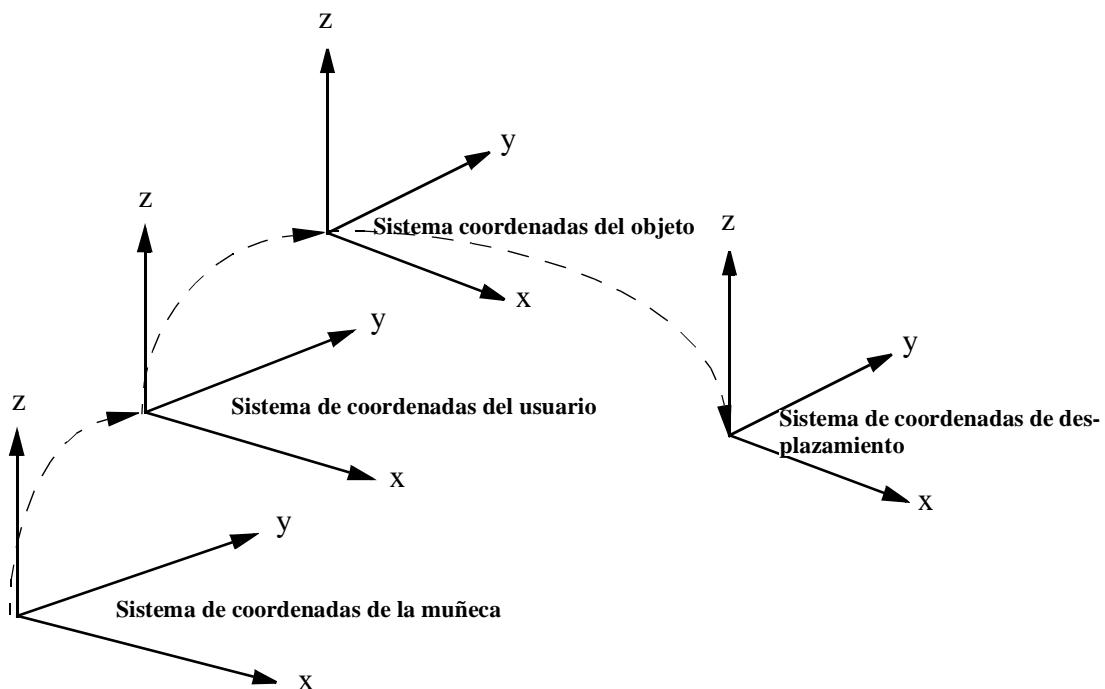


Figura 12 El sistema de coordenadas de desplazamiento de programa también podrá utilizarse con un TCP estacionario.

1.4 Información relacionada

	<u>Descripción:</u>
Definición del sistema de coordenadas mundo	Guía del Usuario - Parámetros del Sistema
Definición del sistema de coordenadas del usuario	Guía del Usuario - Calibración Tipos de Datos - <i>wobjdata</i>
Definición del sistema de coordenadas del objeto	Guía del Usuario - Calibración Tipos de Datos - <i>wobjdata</i>
Definición del sistema de coordenadas de la herramienta	Guía del Usuario - Calibración Tipos de Datos - <i>tooldata</i>
Definición del punto central de la herramienta	Guía del Usuario - Calibración Tipos de Datos - <i>tooldata</i>
Definición del sistema de coordenadas de desplazamiento de programa	Guía del Usuario - Calibración Resumen RAPID - <i>Características de Movimiento</i>
Movimiento en los diferentes sistemas de coordenadas	Guía del Usuario - Movimiento

2 Posicionamiento durante la Ejecución del Programa

2.1 Generalidades

Durante la ejecución del programa, las instrucciones de posicionamiento del programa robot controlan todos los movimientos. La tarea principal de las instrucciones de posicionamiento consiste en proporcionar la siguiente información sobre la manera de realizar los movimientos:

- El punto de destino del movimiento (definido como la posición del punto central de la herramienta, la orientación de la herramienta, la configuración del robot y la posición de los ejes externos).
- El método de interpolación utilizado para alcanzar el punto de destino, como por ejemplo, la interpolación eje a eje, la interpolación lineal o circular.
- La velocidad del robot y de los ejes externos.
- Los datos de zona (definen como el robot y los ejes externos deben alcanzar el punto de destino).
- Los sistemas de coordenadas (herramienta, usuario y objeto) utilizados para el movimiento.

Como una alternativa para la definición de la velocidad del robot y de los ejes externos, se podrá programar el tiempo para el movimiento. No obstante, esto deberá evitarse en el caso en que se utilice la función de oscilación. En vez de ello, se deberá usar las velocidades de la orientación y de los ejes externos para limitar la velocidad, cuando no se realizan movimientos con el TCP o cuando se realizan movimientos pequeños.



En la manipulación de materiales y en aplicaciones de paletización con movimientos frecuentes e intensivos, puede ocurrir que la supervisión del sistema de accionamiento se dispare y detenga el robot a fin de impedir que se produzca un sobrecalentamiento de los motores. Si esto ocurre, el tiempo de ciclo deberá ser aumentado ligeramente reduciendo la velocidad programada o la aceleración.

2.2 Interpolación de la posición y orientación de la herramienta

2.2.1 Interpolación eje a eje

Cuando la precisión de la trayectoria no es un factor demasiado importante, se utilizará este tipo de movimiento para mover la herramienta rápidamente de una posición a otra. La interpolación eje a eje permite también que un eje pueda moverse de una posición a otra dentro de su área de trabajo, en un único movimiento.

Todos los ejes se mueven del punto de arranque al punto de destino a una velocidad de ejes constante (véase la Figura 13).

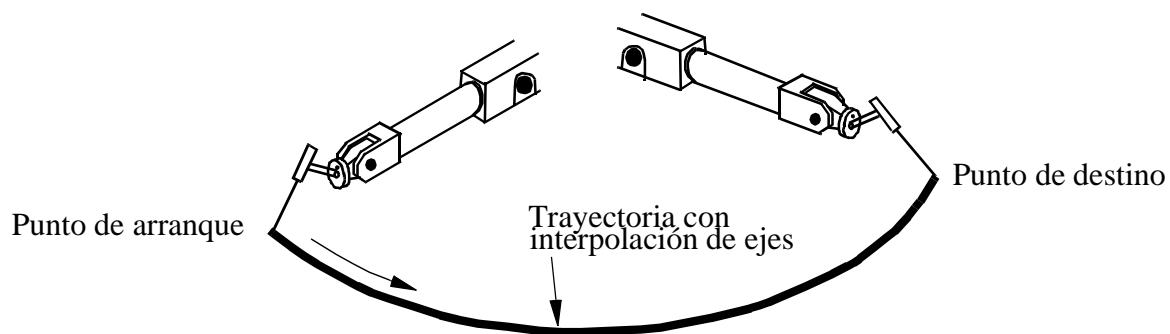


Figura 13 La interpolación eje a eje es con frecuencia la forma más rápida para moverse entre dos puntos ya que los ejes del robot siguen el camino más corto entre el punto de arranque y el punto de destino (desde la perspectiva de los ángulos de los ejes).

La velocidad del punto central de la herramienta está expresado en mm/s (en el sistema de coordenadas del objeto). Dado que se trata de una interpolación eje a eje, la velocidad no será exactamente igual que el valor programado.

Durante la interpolación, se determinará la velocidad del eje restringido, es decir, el eje que se desplaza más rápidamente respecto a su velocidad máxima para realizar el movimiento. A continuación, serán calculadas las velocidades de los ejes restantes de forma que todos los ejes alcancen el punto de destino al mismo tiempo.

Todos los ejes son coordinados de forma a obtener una trayectoria que sea independiente de la velocidad. La aceleración será automáticamente optimizada para conseguir la máxima capacidad del robot.

2.2.2 Interpolación lineal

En una interpolación lineal, el TCP se desplaza siguiendo una línea recta a partir del punto de arranque hasta el punto de destino (véase la Figura 14).

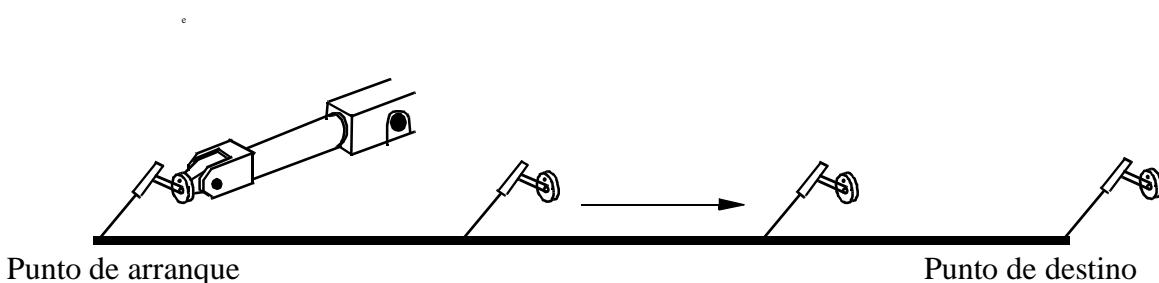


Figura 14 Interpolación lineal sin reorientación de la herramienta.

Para obtener una trayectoria lineal en el sistema de coordenadas del objeto, los ejes del robot deberán seguir una trayectoria que no es lineal dentro del área de trabajo. Cuanto menos lineal sea la configuración del robot, más aceleraciones y deceleraciones se requerirán para que el robot se mueva en línea recta y para obtener la orientación deseada de la herramienta. En el caso en que la configuración sea extremadamente no lineal (por ejemplo en la proximidad de puntos singulares del brazo y de la muñeca), uno o más ejes requerirán un par más elevado de lo que los motores pueden ofrecer. En estos casos, la velocidad de todos los ejes será automáticamente reducida.

La orientación de la herramienta permanece constante durante todo el movimiento a menos que se haya programado una reorientación. Si se reorienta la herramienta, la rotación se realizará a una velocidad constante.

Se podrá especificar una velocidad de rotación máxima (en grados por segundo) al realizar la rotación de la herramienta. Si la velocidad tiene un valor bajo, la reorientación será suave, independientemente de la velocidad definida para el punto central de la herramienta. Si la velocidad tiene un valor elevado, la velocidad de reorientación será limitada únicamente por las velocidades máximas de los motores. Mientras que ningún motor no exceda el límite del par, se mantendrá la velocidad definida. Si, de lo contrario, uno de los motores excediera el límite de corriente, la velocidad de todo el movimiento (con respecto a la posición y la orientación) sería reducida.

Todos los ejes están coordinados de forma a obtener una trayectoria independiente de la velocidad. La aceleración es optimizada automáticamente.

2.2.3 Interpolación circular

Para definir una trayectoria circular se deberán utilizar tres posiciones programadas que definen un segmento circular. El primer punto que se deberá programar es el punto de arranque del segmento circular. El punto siguiente es un punto auxiliar (punto de círculo) que sirve para definir la curvatura del círculo, y el tercer punto indicará el final del círculo (véase la Figura 15).

Los tres puntos programados deberán ser lo más equidistantes posibles en el arco circular para conseguir una trayectoria lo más precisa posible.

La orientación definida para el punto auxiliar sirve para seleccionar entre una curvatura pequeña y una curvatura grande para la orientación del punto de arranque al punto de destino.

En el caso en que la orientación programada sea la misma que la del círculo en el punto de arranque y de destino y que la orientación en el punto auxiliar sea prácticamente la misma orientación que la del círculo, la orientación de la herramienta permanecerá constante respecto a la trayectoria.

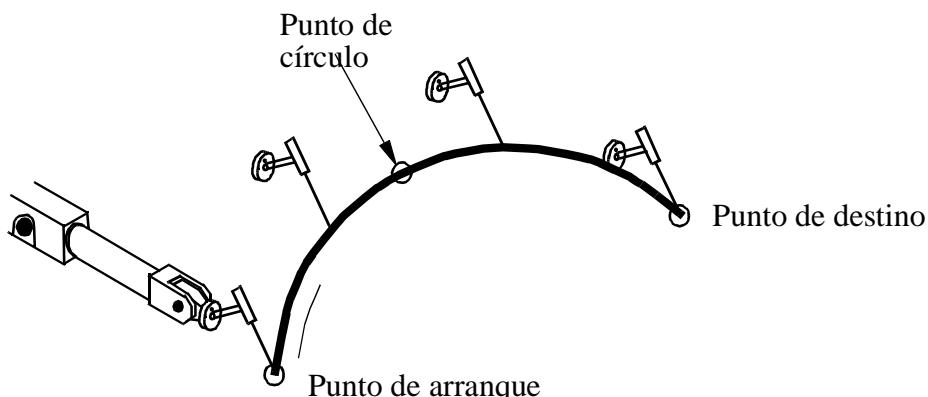


Figura 15 Interpolación circular con una curvatura pequeña para una parte del círculo (arco circular) con un punto de arranque, un punto de círculo y un punto de destino.

No obstante, si la orientación en el punto auxiliar está programada más cerca de la orientación girada de 180°, se seleccionará la curvatura alternativa (véase la Figura 16).

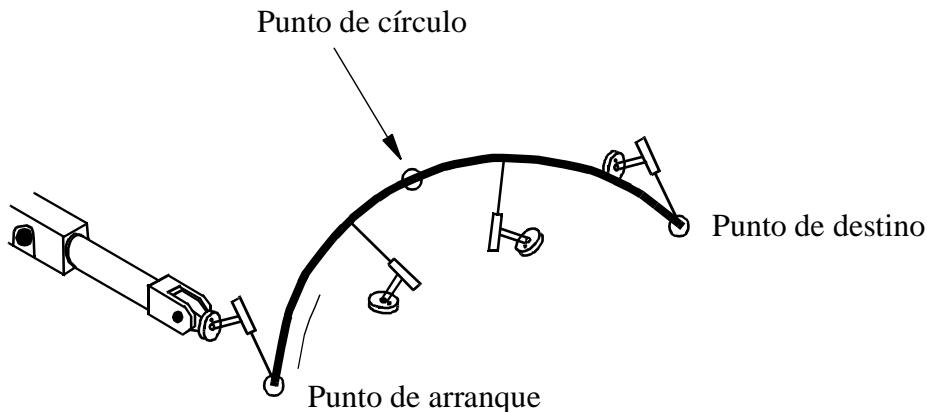


Figura 16 Una Interpolación circular con una curvatura grande se obtendrá definiendo la orientación del punto de círculo en la dirección opuesta al punto de arranque.

Mientras que ningún par del motor exceda los límites máximos permitidos, la herramienta se moverá a la velocidad programada en el arco circular. En el caso en que el par de algún motor sea insuficiente, la velocidad será automáticamente reducida en aquellos lugares de la trayectoria circular donde la capacidad del motor sea insuficiente.

Todos los ejes están coordinados de modo a obtener una trayectoria que sea independiente de la velocidad. La aceleración está optimizada automáticamente.

2.2.4 Sing Area\Wrist

Durante una ejecución en proximidad de un punto singular, la interpolación lineal o circular puede resultar problemática. En este caso, se recomienda utilizar una interpolación modificada, lo cual significa que los ejes de la muñeca son interpolados eje a eje, y el TCP sigue una trayectoria lineal o circular. La orientación de la herramienta, no obstante, diferirá de algo de la orientación programada.

En el caso de *Sing Area\Wrist* la orientación en el punto auxiliar de círculo será la misma que la programada. No obstante, la herramienta no tendrá una dirección constante respecto al plano circular, como en la interpolación circular normal. Cuando la trayectoria circular pasa por un punto singular, la orientación de las posiciones programadas algunas veces deberán ser modificadas a fin de evitar grandes movimientos de la muñeca, que pueden ocurrir si se genera una nueva configuración completa de la muñeca cuando se ejecuta el círculo (ejes 4 y 6 movidos de 180 grados cada uno).

2.3 Interpolación de las trayectorias esquina

El punto de destino se definirá como un punto de paro en vistas a obtener un movimiento punto a punto. Esto significa que el robot y todos los ejes externos se pararán y que no se podrá continuar con el posicionamiento hasta que la velocidad de todos los ejes sea cero y que los ejes estén cerca de su destino.

Los puntos de paso se utilizan para conseguir movimientos continuos junto a las posiciones programadas. De esta forma, el robot evitará las posiciones pasando a una velocidad elevada sin tener la necesidad de reducir la velocidad de forma innecesaria. Un punto de paso generará una trayectoria esquina (trayectoria parabólica) más allá del punto programado, lo que normalmente significará que la posición programada nunca será alcanzada. El principio y el final de esta trayectoria esquina será definida determinando una zona alrededor de la posición programada (véase la Figura 17).

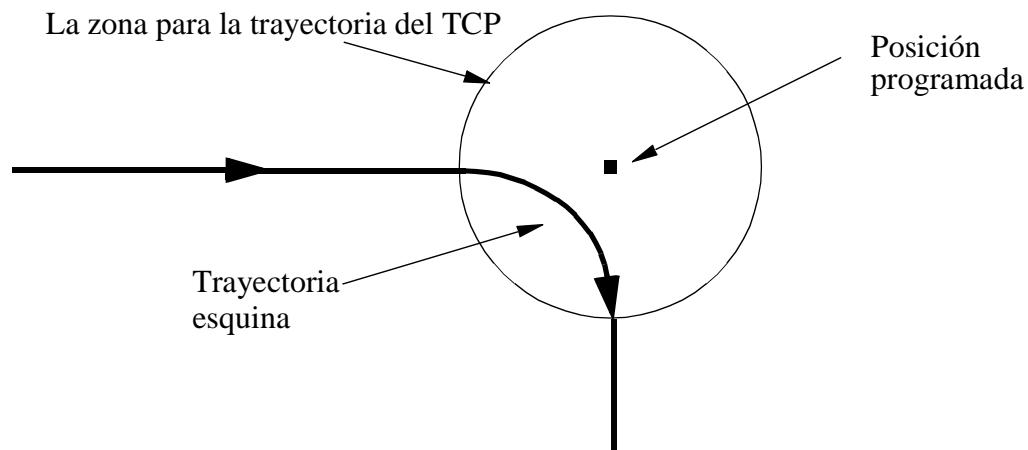


Figura 17 Un punto de paso genera una trayectoria esquina para evitar el punto programado.

Todos los ejes están coordinados de forma a obtener una trayectoria que es independiente de la velocidad. La aceleración está optimizada automáticamente.

2.3.1 Interpolación eje a eje en las trayectorias esquina

El tamaño de las trayectorias esquina (zonas) para el movimiento del TCP se expresa en mm (véase la Figura 18). Puesto que la interpolación se realiza eje a eje, el tamaño de las zonas (en mm) deberá volverse a calcular en ángulos de los ejes (radianes). Este cálculo tiene un factor de error (normalmente del 10% como máximo), lo que significa que la zona verdadera se desviará de algo respecto a la programada.

Si se han programado diferentes velocidades antes o después de la posición, el cambio de una velocidad a otra será suave y tendrá lugar dentro de la trayectoria esquina sin afectar a la trayectoria actual.

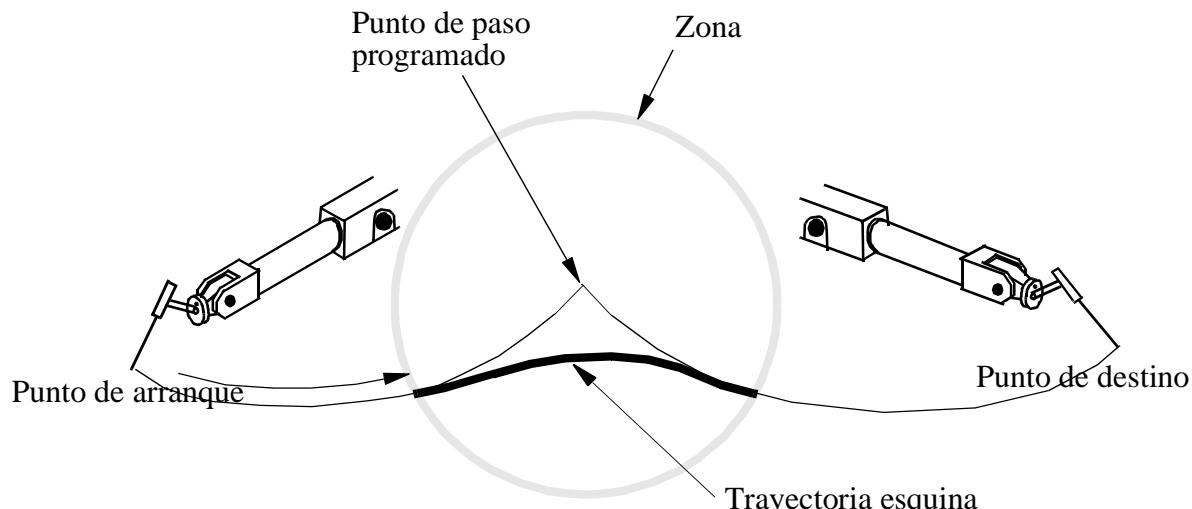


Figura 18 Durante la interpolación eje a eje, se generará una trayectoria para evitar el punto de paso.

2.3.2 Interpolación lineal de una posición en las trayectorias esquina

El tamaño de las trayectorias esquina (zonas) para el movimiento del TCP se expresa en mm (véase la Figura 19).

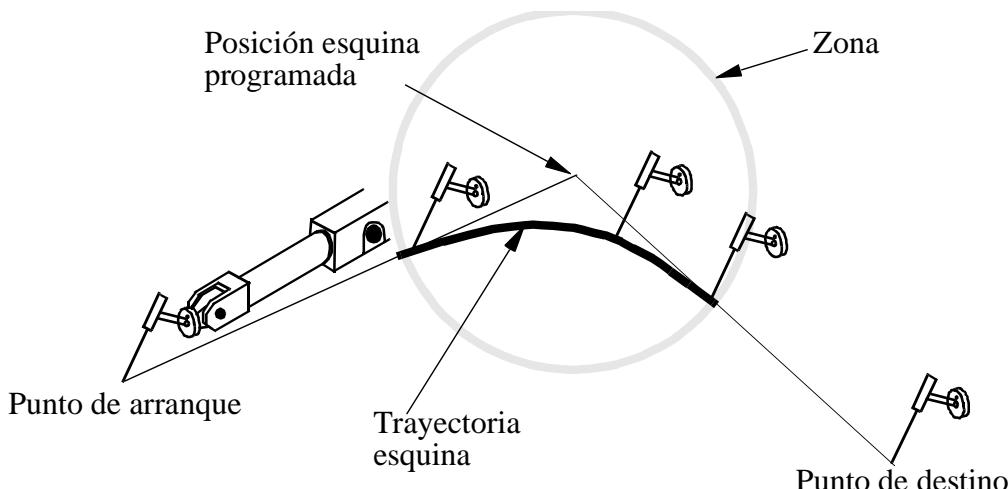


Figura 19 Durante la interpolación lineal, se genera una trayectoria esquina para evitar un punto de paso.

Si se han programado diferentes velocidades antes o después de la posición esquina, el cambio de una a otra será suave y tendrá lugar dentro de la zona esquina sin afectar la trayectoria actual.

En el caso en que la herramienta deba llevar a cabo un proceso (como de soldadura al arco, de aplicación de adhesivo o de corte por chorro de agua) en la trayectoria esquina, el tamaño de la zona debería ser ajustado para conseguir la trayectoria deseada. En el caso en que la forma de la trayectoria esquina parabólica no corresponda con la geometría del objeto, las posiciones programadas podrán colocarse más cerca entre sí, permitiendo así la aproximación a la trayectoria deseada utilizando dos o más trayectorias parabólicas más pequeñas.

2.3.3 Interpolación lineal de la orientación en las trayectorias esquina

Se podrán definir zonas para las orientaciones de la herramienta, exactamente de la misma manera que se pueden definir zonas para las posiciones de las herramientas. La zona de la orientación será normalmente mayor que la zona de posición. En este caso, la reorientación iniciará la interpolación hacia la orientación de la posición siguiente antes de que empiece la trayectoria esquina. La reorientación será entonces más suave y probablemente no será necesario reducir la velocidad para llevar a cabo la reorientación.

La herramienta será reorientada de forma que la orientación al final de la zona sea la misma que si se hubiera programado un punto de paro (véase la Figura 20, 21, 22).

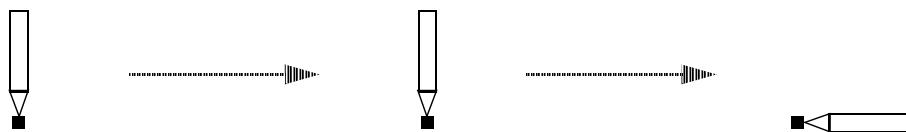


Figura 20 Se han programado tres posiciones con diferentes orientaciones de la herramienta

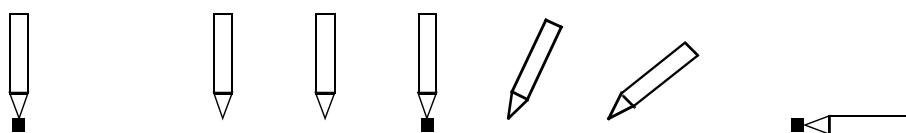


Figura 21 Si todas las posiciones fueran puntos de paro, la ejecución del programa tendría este aspecto.

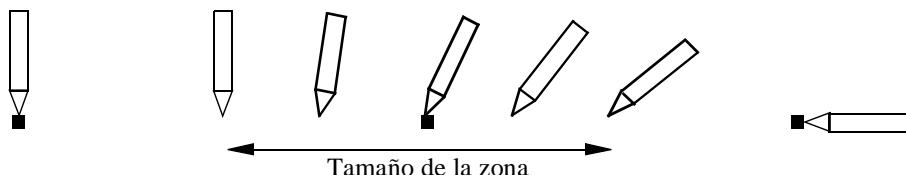


Figura 22 Si la posición central fuera un punto de paso, la ejecución del programa tendría este aspecto.

La zona de orientación para el movimiento de la herramienta suele ser expresada en mm. De esta forma, el usuario podrá determinar directamente el lugar en la trayectoria donde la zona de orientación empieza y acaba. Si no se mueve la herramienta, el tamaño de la zona será expresada en grados del ángulo de rotación en lugar de mm de TCP.

En el caso en que se hayan programado diferentes velocidades de reorientación antes o después del punto de paso, y si las velocidades de reorientación limitan el movimiento, el cambio de una velocidad a otra tendrá lugar de forma suave dentro de la trayectoria esquina.

2.3.4 Interpolación de los ejes externos en trayectorias esquina

También se podrán definir zonas para los ejes externos, de la misma forma que para la orientación. En el caso en que la zona de ejes externos es mayor que la zona del TCP, la interpolación de los ejes externos hacia el destino de la posición programada siguiente, empezará antes de que empiece la trayectoria esquina del TCP. Esto puede

utilizarse para suavizar los movimientos de los ejes externos de la misma forma que se utiliza la zona de orientación para suavizar los movimientos de la muñeca.

2.3.5 Trayectorias esquina cuando se cambia el método de interpolación

Las trayectorias esquina son también generadas cuando se cambia de método de interpolación. El método de interpolación utilizado en las trayectorias esquinas actuales será determinado de forma que el cambio de un método a otro sea lo más suave posible. Si las zonas de trayectoria esquina para la orientación y para la posición no son del mismo tamaño, se podrá usar más de un método de interpolación en la trayectoria esquina (véase la Figura 23).

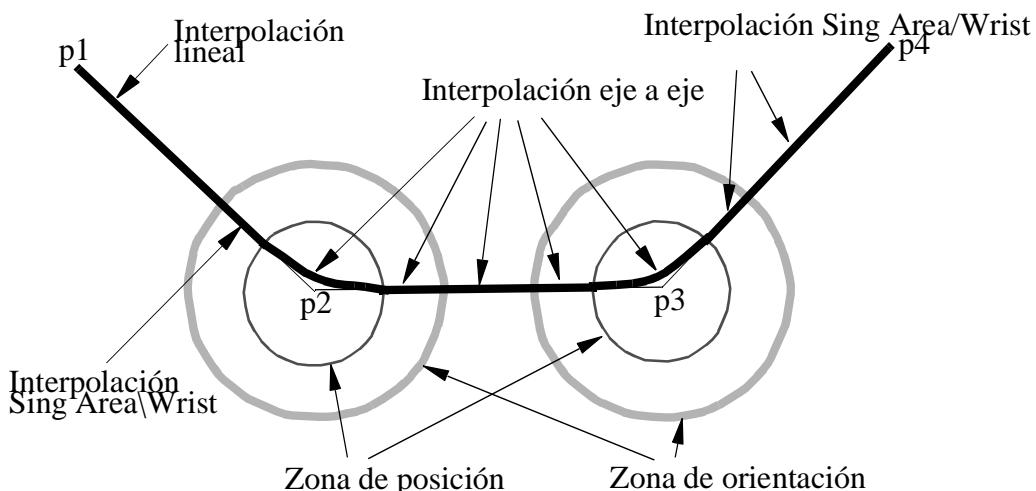


Figura 23 Interpolación durante el cambio de método de interpolación. La interpolación lineal ha sido programada entre p1 y p2; la interpolación eje a eje entre p2 y p3; y la interpolación Sing Area/Wrist entre p3 y p4.

En el caso en que se cambie de interpolación, de un movimiento de TCP normal a una reorientación sin un movimiento de TCP o viceversa, no se generará ninguna zona esquina. Sería el mismo caso si la interpolación se cambiara a o desde un movimiento de eje externo sin un movimiento de TCP.

2.3.6 Interpolación al cambiar de sistema de coordenadas

Cuando hay un cambio de sistema de coordenadas en una trayectoria esquina, por ejemplo, un TCP nuevo o un objeto de trabajo nuevo, se utilizará la interpolación eje a eje de la trayectoria esquina. Esto es aplicable también cuando se pasa de un funcionamiento coordinado a un funcionamiento no coordinado o viceversa.

2.3.7 Trayectorias esquina con zonas que se superponen

Si las posiciones programadas se encuentran cerca unas de otras, puede ocurrir que las zonas programadas se superpongan. Para conseguir una trayectoria bien definida y una velocidad óptima en todas circunstancias, el robot reducirá el tamaño de la zona a la mitad de la distancia de una posición programada superpuesta a otra (véase la Figura

24). Siempre se usará el mismo radio de zona para entradas o salidas de una posición programada, con el objetivo de obtener trayectorias esquina simétricas.

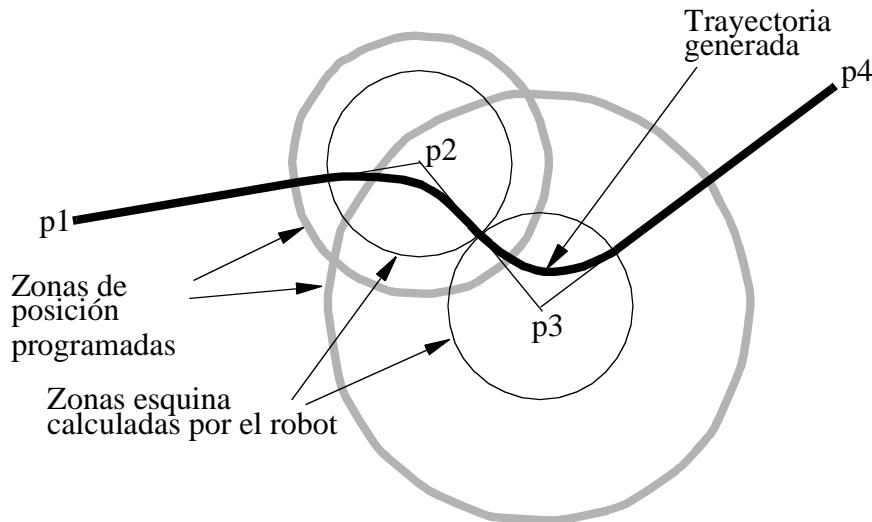


Figura 24 Interpolación con zonas de posición que se superponen. Las zonas alrededor de p2 y p3 son mayores que la mitad de la distancia entre p2 y p3. Así, el robot reducirá el tamaño de las zonas para que sean iguales a la mitad de la distancia entre p2 y p3, generando de esta forma trayectorias de esquina simétricas dentro de las zonas.

Puede ocurrir que las zonas de trayectorias esquina de la posición y de la orientación se superpongan. En cuanto una de estas zonas de trayectoria esquina se superponen, el tamaño de la zona será reducido (véase la Figura 25).

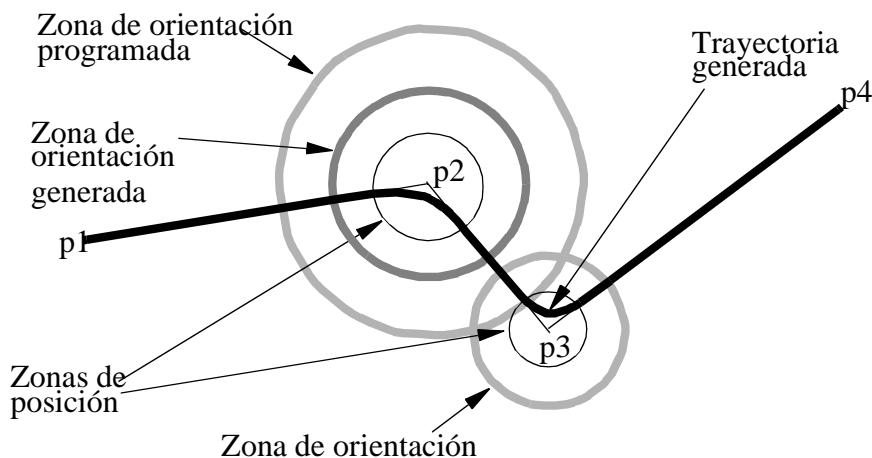


Figura 25 Interpolación con zonas de orientación que se superponen. La zona de orientación en p2 es mayor que la mitad de la distancia entre p2 y p3 y será así reducida a la mitad de la distancia entre p2 y p3. Las zonas de posición no se superponen y por consiguiente no serán reducidas; la zona de orientación en p3 tampoco será reducida.

2.3.8 Especificación del tiempo en la programación de los puntos de paso

Ocasionalmente, puede ocurrir que si el movimiento siguiente no ha sido programado en el tiempo, los puntos de paso programados pueden originar un punto de paro. Esto

podría ocurrir cuando:

- Una serie de instrucciones lógicas con tiempos de ejecución largos han sido programados entre movimientos cortos.
- Los puntos se encuentran muy juntos entre sí y se utilizan altas velocidades.

En el caso en que los puntos de paro constituyan un problema, entonces se deberá utilizar una ejecución concurrente del programa.

2.4 Ejes independientes

Un eje independiente es un eje que se mueve de forma independiente respecto a los demás ejes del sistema robot. Se puede, si se desea, activar un eje en el modo independiente y más adelante, volver a activarlo en el modo normal.

Existe un conjunto especial de instrucciones que sirve para manipular los ejes independientes. El movimiento del eje se encuentra especificado por cuatro instrucciones diferentes de movimiento. Por ejemplo, la instrucción *IndCMove* provocará el arranque del eje en un movimiento continuo. Entonces, el eje continuará moviéndose a una velocidad constante (independientemente de lo que está realizando el robot) hasta que se ejecute otra instrucción nueva independiente.

Para volver a pasar en el modo normal, se deberá usar una instrucción de reinicialización, *IndReset*. La instrucción de reinicialización puede también accionar una referencia nueva para el sistema de medida -un tipo de sincronización nueva para el eje. Cuando se haya vuelto a cambiar el eje en el modo normal, se podrá ejecutarlo como si fuera un eje normal.

2.4.1 Ejecución del programa

La ejecución de una instrucción *Ind_Move*, tendrá por resultado el cambio inmediato del modo de un eje, pasándolo al modo independiente. Esto se produce incluso si el eje se está moviendo en ese momento, por ejemplo como cuando se ha programado un punto anterior como un punto de paso o cuando se está llevando a cabo simultáneamente la ejecución del programa.

En el caso en que se ejecute una instrucción *Ind_Move* nueva antes de que haya terminado la última instrucción, inmediatamente la instrucción nueva se superpondrá sobre la instrucción antigua.

Si se interrumpe la ejecución del programa cuando un eje independiente se está moviendo, éste se detendrá. Cuando se reanude la ejecución del programa, el eje independiente volverá a arrancar inmediatamente. No se produce ninguna coordinación activa entre un eje independiente y los demás ejes en el modo normal.

Si se produce una pérdida de tensión mientras un eje está en el modo independiente, no se podrá rearrancar el programa. Aparecerá entonces un mensaje de error y se deberá arrancar el programa desde el principio.

Observar que no se podrá desactivar una unidad mecánica si uno de sus ejes está en el modo independiente.

2.4.2 Ejecución paso a paso

Durante la ejecución paso a paso, un eje independiente será ejecutado únicamente cuando otra instrucción sea ejecutada. El movimiento del eje también se ejecutará paso a paso de acuerdo con la ejecución de otras instrucciones, véase la Figura 26.

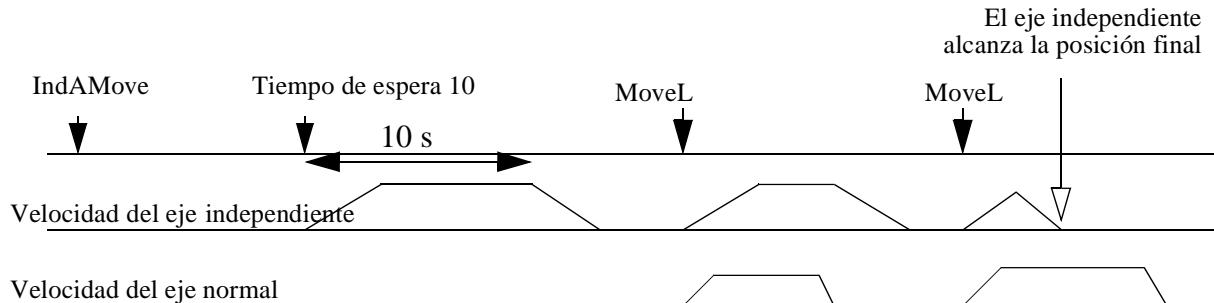


Figura 26 Ejecución paso a paso de un eje independiente.

2.4.3 Movimiento

Cuando los ejes están en el modo independiente no podrán ser movidos. Si se realiza un intento de hacer funcionar un eje manualmente, éste no se moverá y aparecerá visualizado un mensaje de error. Para salir del modo independiente, se deberá ejecutar una instrucción *IndReset* o mover el puntero del programa al menú principal.

2.4.4 Área de trabajo

El área de trabajo física abarca el movimiento total del eje.

El área de trabajo lógica es el área utilizada por las instrucciones RAPID y que aparece en la ventana de movimiento. Esta área de trabajo puede diferir según se esté trabajando con el modo normal o con el modo independiente.

Después de haber llevado la sincronización del sistema (actualizado el cuenta revoluciones), el área de trabajo física y lógica deben coincidir. Mediante la instrucción *IndReset* se podrá mover el área de trabajo lógica, véase la Figura 27.

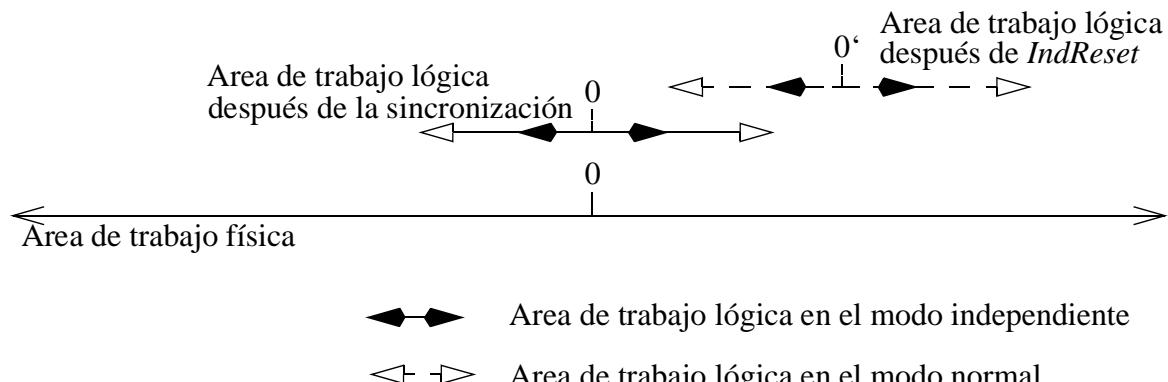


Figura 27 El área de trabajo lógica puede ser movida, utilizando la instrucción *IndReset*.

La resolución de las posiciones disminuye conforme uno se aparta de la posición lógica 0. Una baja resolución junto con un ajuste rígido del controlador puede provocar un par inaceptable, ruido y una inestabilidad de controlador. Comprobar el ajuste del controlador y la capacidad del eje junto al límite del área de trabajo en el momento de la instalación. Comprobar también si la resolución de la posición y la eficacia de la trayectoria son aceptables.

2.4.5 Velocidad y aceleración

En el modo manual a velocidad reducida, la velocidad queda reducida al mismo nivel que si el eje estuviera funcionando como si no fuera independiente. Observar que la función *IndSpeed\InSpeed* no será VERDADERA (TRUE) si la velocidad del eje es reducida.

La instrucción *VelSet* y la corrección de velocidad expresados en porcentajes en la ventana de producción serán activas para el movimiento independiente. Observar que la corrección mediante la ventana de producción inhabilita el valor TRUE de la función *IndSpeed\InSpeed*.

En el modo independiente, el valor más bajo de aceleración y deceleración, especificados en el archivo de configuración serán utilizados para la aceleración y la deceleración. Este valor podrá ser reducido por el valor rampa en la instrucción (1-100%). La instrucción *AccSet* no afecta a los ejes en el modo independiente.

2.4.6 Ejes del robot

Unicamente el eje robot 6 podrá usarse como eje independiente. Normalmente sólo se usa la instrucción *IndReset* para este eje. No obstante, la instrucción *IndReset* también podrá utilizarse para el eje 4 en los modelos IRB 2400 y 4400.

Si se usa el eje 6 como eje independiente, pueden ocurrir problemas de singularidad debido a que se sigue usando la función de transformación de la coordenada normal de 6 ejes. En el caso en que surja un problema, se deberá ejecutar el mismo programa con el eje 6 en el modo normal. Modificar los puntos o utilizar las instrucciones *SingArea\Wrist* o *MoveJ*.

El eje 6 está también activo internamente en el cálculo de la capacidad de la trayectoria. De ello resulta que un movimiento interno del eje 6 puede reducir la velocidad de los demás ejes del sistema.

El área de trabajo independiente del eje 6 está definida con los ejes 4 y 5 en posición inicial. Si el eje 4 o 5 está fuera de la posición inicial, el área de trabajo del eje 6 se moverá debido al acoplamiento del engranaje. No obstante, la lectura de la posición del eje 6 en la unidad de programación está compensada con las posiciones de los ejes 4 y 5 mediante el acoplamiento del engranaje.

2.5 Servo Suave

En ciertas aplicaciones se requiere un servo que actúa como un muelle mecánico. Esto significa que la fuerza del robot en el objeto de trabajo aumentará en función de la distancia entre la posición programada (detrás del objeto de trabajo) y la posición de contacto (herramienta del robot - objeto de trabajo).

La relación entre la desviación de la posición y la fuerza, será definida mediante un parámetro denominado **suavidad**. Cuanto más elevado sea el parámetro de suavidad, mayor será la desviación de la posición para obtener la misma fuerza.

El parámetro de suavidad está activado en el programa y es posible cambiar los valores de suavidad en cualquier sitio en el programa. Se podrá determinar diferentes valores de suavidad para los distintos ejes. También se podrá mezclar ejes que tengan un servo normal con ejes que tengan un servo suave.

La activación y desactivación del servo suave así como el cambio de los valores de suavidad podrá realizarse cuando el robot está en movimiento. Cuando esto ocurre, se realizará un ajuste entre los distintos modos de servo y entre los distintos valores de suavidad para conseguir transiciones suaves. El tiempo de ajuste podrá determinarse desde el programa con la rampa de parámetros. Con el parámetro *ramp = 1*, las transiciones tardarán 0,5 segundos, y de forma general el tiempo de transición será *ramp x 0,5* en segundos.

Obsérvese que la desactivación del servo suave no deberá realizarse cuando haya una fuerza ejercida entre el robot y el objeto de trabajo.

Con valores de suavidad altos, existe un riesgo de que las desviaciones de la posición servo sea tan alto, que los ejes se moverán fuera del área de trabajo del robot.

2.6 Paro y rearranque

Un movimiento puede ser detenido de tres maneras diferentes:

1. *Con un paro normal* el robot se parará en la trayectoria, lo que facilitará el rearranque.
2. *Con un paro rápido* el robot se parará en menos tiempo que en un paro normal, pero la trayectoria de deceleración no seguirá la trayectoria programada. Este método de paro se utilizará, por ejemplo, para un paro de búsqueda, cuando es importante parar

Posicionamiento durante la Ejecución del Programa

Principios de Movimiento y de E/S

el movimiento en cuanto antes.

3. *Con un paro instantáneo* los frenos mecánicos se utilizan para obtener una menor distancia de deceleración, que es lo más corta posible según se ha especificado por razones de seguridad. La desviación de la trayectoria será normalmente mayor para un paro instantáneo que para un paro rápido.

Después de un paro (cualquiera de los tipos anteriores) siempre se podrá realizar un rearranque en la trayectoria interrumpida. En el caso en que el robot se haya detenido fuera de la trayectoria programada, el rearranque empezará con un regreso a la posición en la trayectoria, donde el robot debería haber parado.

Un rearranque después de un corte de potencia es equivalente a un rearranque después de un paro instantáneo. Deberá tenerse en cuenta que el robot siempre regresará a la trayectoria antes de que el funcionamiento del programa interrumpido sea rearancado, incluso en los casos en que el corte de potencia ha ocurrido mientras se estaba ejecutando una instrucción lógica. Al rearranque, todos los tiempos son contados desde el principio; por ejemplo, el posicionamiento en el tiempo o una interrupción en la instrucción *WaitTime*.

2.7 Información relacionada

	<u>Descripción en:</u>
Definición de la velocidad	Tipos de Datos - <i>speeddata</i>
Definición de las zonas (trayectorias esquina)	Tipos de Datos - <i>zonedata</i>
Instrucciones para la interpolación eje a eje	Instrucciones - <i>MoveJ</i>
Instrucciones para la interpolación lineal	Instrucciones - <i>MoveL</i>
Instrucciones para la interpolación circular	Instrucciones - <i>MoveC</i>
Instrucciones para la interpolación modificada	Instrucciones - <i>SingArea</i>
Singularidad	Principios de Movimiento y de E/S - <i>Singularidad</i>
Ejecución concurrente del programa	Principios de Movimiento y de E/S - <i>Sincronización con instrucciones lógicas</i>
Optimización de la CPU	Guía del Usuario - Parámetros del Sistema

3 Sincronización con Instrucciones Lógicas

Las instrucciones suelen ejecutarse de forma secuencial en el programa. No obstante, las instrucciones lógicas pueden también ser ejecutadas en posiciones específicas o durante el movimiento que se está ejecutando.

Una instrucción lógica es cualquier instrucción que no genera ningún movimiento del robot o de los ejes externos, como por ejemplo, una instrucción de E/S.

3.1 Ejecución secuencial del programa en los puntos de paro

Si una instrucción de posicionamiento ha sido programada como un punto de paro, la instrucción siguiente no será ejecutada hasta que el robot y los ejes externos se hayan parado, es decir, cuando la posición programada haya sido alcanzada (véase la Figura 28).

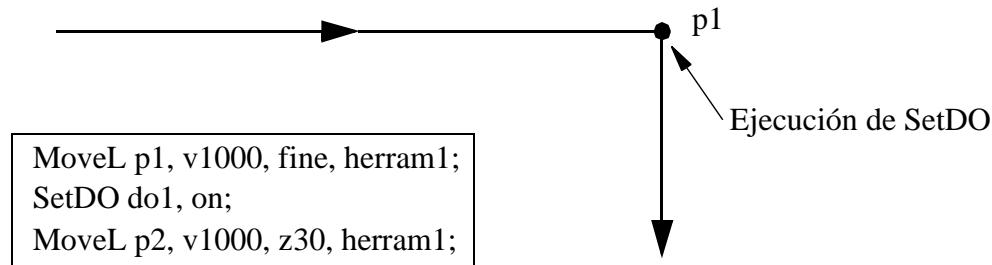


Figura 28 Una instrucción lógica después de un punto de paro no se ejecuta antes de haber alcanzado la posición de destino.

3.2 Ejecución secuencial del programa en los puntos de paso

Si una instrucción de posicionamiento ha sido programada como un punto de paso, las instrucciones lógicas siguientes se ejecutarán cierto tiempo antes de alcanzar la zona más grande (para posición, orientación o ejes externos). Véase la Figura 29. Estas instrucciones serán ejecutadas a continuación en orden.

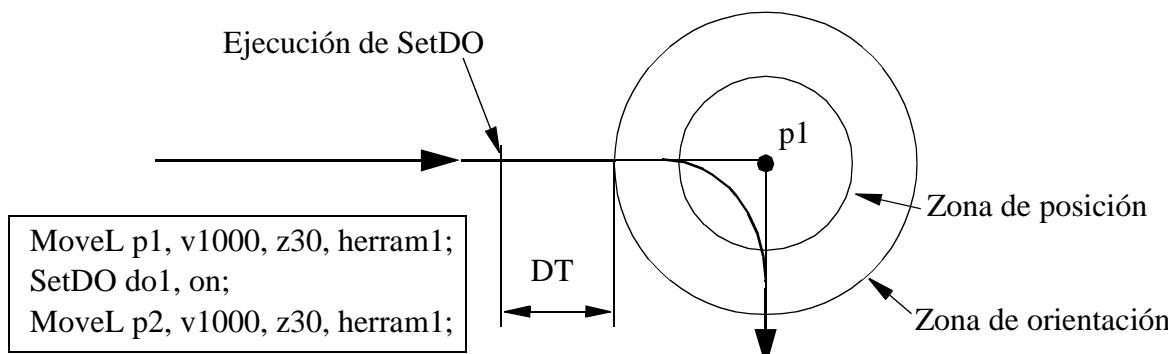


Figura 29 Una instrucción lógica que sigue un punto de paso se ejecutará antes de alcanzar la zona más grande.

El momento en que se ejecutan estas instrucciones (*DT*) comprende los siguientes componentes de tiempo:

- El tiempo necesario al robot para prever el siguiente movimiento: 0,1 segundos aproximadamente.
- El retraso del robot (retraso del servo) en segundos: de 0 a 1,0 segundos en función de la velocidad y de la capacidad de deceleración actual del robot.

3.3 Ejecución concurrente del programa

La ejecución concurrente del programa será programada utilizando el argumento *\Conc* en la instrucción de posicionamiento. Este argumento sirve para:

- Ejecutar una o más instrucciones lógicas al mismo tiempo que el robot se mueve, para reducir el tiempo de ciclo (por ejemplo, utilizado durante la comunicación a través de canales serie).

Cuando una instrucción de posicionamiento con el argumento *\Conc* es ejecutada, las siguientes instrucciones lógicas serán también ejecutadas (de forma secuencial) según lo siguiente:

- Si el robot no se mueve o si la instrucción de posicionamiento anterior termina en un punto de paro, las instrucciones lógicas serán ejecutadas en cuanto la instrucción de posicionamiento corriente arranque (al mismo tiempo que el movimiento). Véase la Figura 30.
- Si la instrucción de posicionamiento anterior termina en un punto de paso, las instrucciones lógicas se ejecutarán en un tiempo determinado (*DT*) antes de alcanzar la zona más grande (para la posición, orientación o ejes externos). Véase la Figura 31.

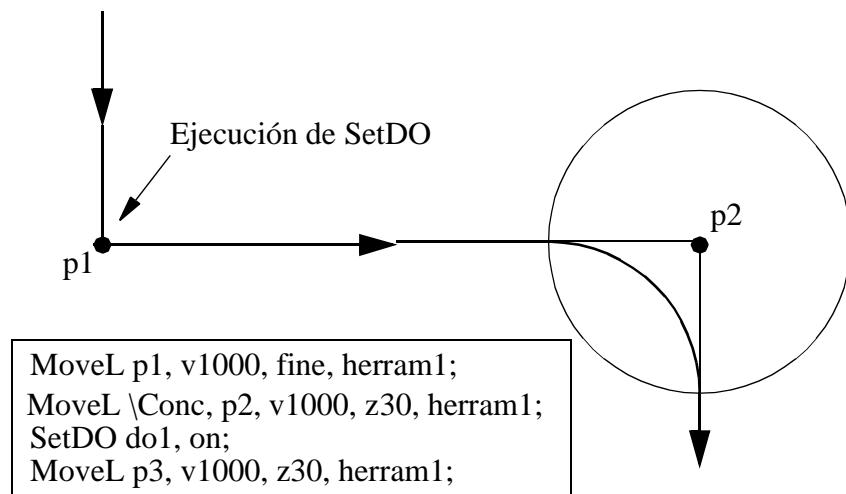


Figura 30 En el caso de una ejecución concurrente de programa después de un punto de paro, la instrucción de posicionamiento y las instrucciones lógicas siguientes se iniciarán en el mismo momento.

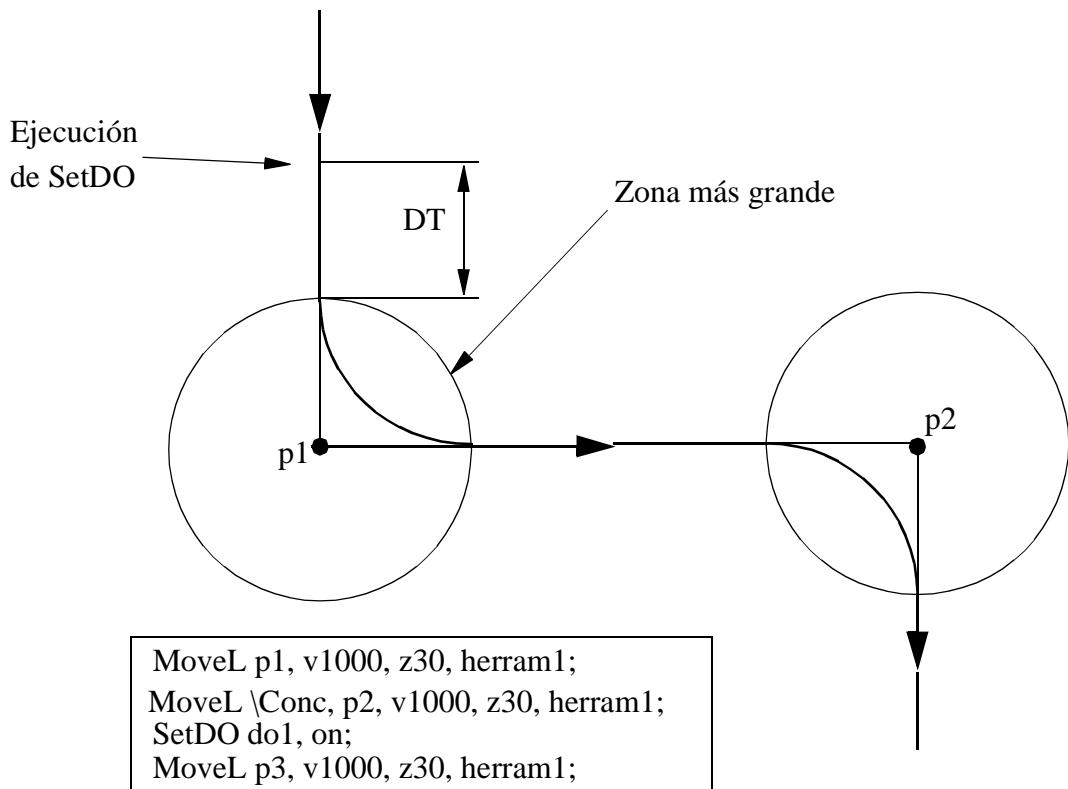


Figura 31 En el caso de una ejecución concurrente del programa después de un punto de paso, las instrucciones lógicas iniciarán su ejecución antes de las instrucciones de posicionamiento que comportan el argumento \Conc.

Las instrucciones que afectan de forma indirecta a los movimientos, como *ConfL* y *SingArea*, serán ejecutadas de la misma manera que otras instrucciones lógicas. No obstante, no afectarán a los movimientos ordenados por las instrucciones de posicionamiento anteriores.

En el caso en que varias instrucciones de posicionamiento con el argumento *\Conc* y varias instrucciones lógicas en una secuencia larga estén mezcladas, se aplicará lo siguiente:

- Las instrucciones de posicionamiento se ejecutarán después de la ejecución de las instrucciones de posicionamiento anteriores.
- Las instrucciones lógicas se ejecutarán directamente, según el orden en que han sido programadas. Esto ocurre al mismo tiempo que el movimiento (véase la Figura 32), lo que significa que las instrucciones lógicas se ejecutarán en una etapa anterior, a la trayectoria en que han sido programadas.
- Los movimientos no podrán ser rearrancados después de que el programa se haya parado, si hay dos o más instrucciones de posicionamiento en secuencia, en espera de ejecución.

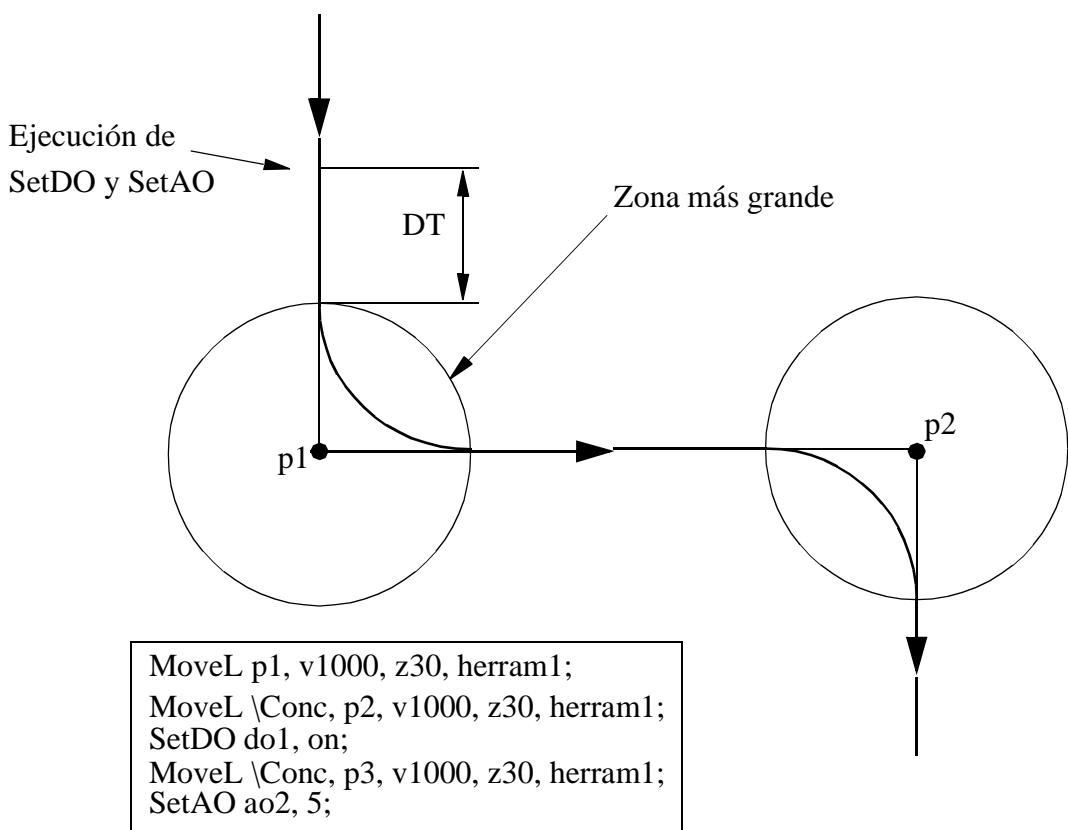


Figura 32 Si varias instrucciones de posicionamiento con el argumento \Conc han sido programadas de forma secuencial, todas las instrucciones lógicas conectadas se ejecutarán en el mismo momento que la primera posición.

Durante la ejecución concurrente del programa, se programarán las siguientes instrucciones para finalizar la secuencia y posteriormente para volver a sincronizar las instrucciones de posicionamiento y las instrucciones lógicas:

- una instrucción de posicionamiento sin el argumento \Conc,
- la instrucción *WaitTime* o *WaitUntil* con el argumento *Inpos*.

3.4 Sincronización de la trayectoria

Para sincronizar el equipo de proceso (como por ejemplo, en la aplicación de adhesivo, de pintura y de soldadura al arco) con los movimientos del robot, se podrán generar distintos tipos de señales de sincronización de trayectoria.

Con lo que llamamos un evento de posiciones, se generará una señal de disparo cuando el robot pasa por una posición predefinida en la trayectoria. Con un evento de tiempo, se generará una señal en un tiempo predefinido antes de que el robot se pare en una posición de paro. Además, el sistema de control también manipula eventos de oscilación, que generan pulsos en ángulos de fase predefinidos de un movimiento de oscilación.

Todas las señales sincronizadas de las posiciones pueden obtenerse tanto antes como

después (tiempo de retraso) del momento en que el robot ha pasado la posición predefinida. La posición es dada por una posición programada y podrá ser ajustada como una distancia de trayectoria antes de la posición programada.

La precisión es de ± 2 ms. Del lado de la trayectoria justo antes de la posición final (máx. 12 mm a 500 mm/s), la señal podrá ser no obstante retrasada de 24 ms.

En el caso de ocurrir un corte de potencia mientras se ejecuta una instrucción de disparo, la condición de disparo es activada a partir del nuevo punto de arranque. Esto a su vez, significa que la señal se activará de forma incorrecta si se usa el argumento `|Start`. En el caso en que la condición de disparo ha ocurrido antes del corte de potencia, entonces el evento asociado a posición será de nuevo activado.

3.5 Información relacionada

Descripción:

Instrucciones de posicionamiento

Resumen RAPID - *Movimiento*

Definición del tamaño de zona

Tipos de Datos - *zonedata*

4 Configuración del Robot

Normalmente se puede alcanzar la misma posición y orientación de la herramienta del robot de varias formas diferentes, utilizando distintos conjuntos de ángulo de ejes. En esto consisten las configuraciones distintas del robot. Si, por ejemplo, una posición se encuentra aproximadamente en el medio del área de trabajo, ciertos robots podrán alcanzar esta posición desde arriba o desde abajo (véase la Figura 33). Esto podrá conseguirse también girando la parte delantera del brazo superior del robot (eje 4) boca abajo mientras se gira los ejes 5 y 6 a la posición y orientación deseadas (véase la Figura 34).

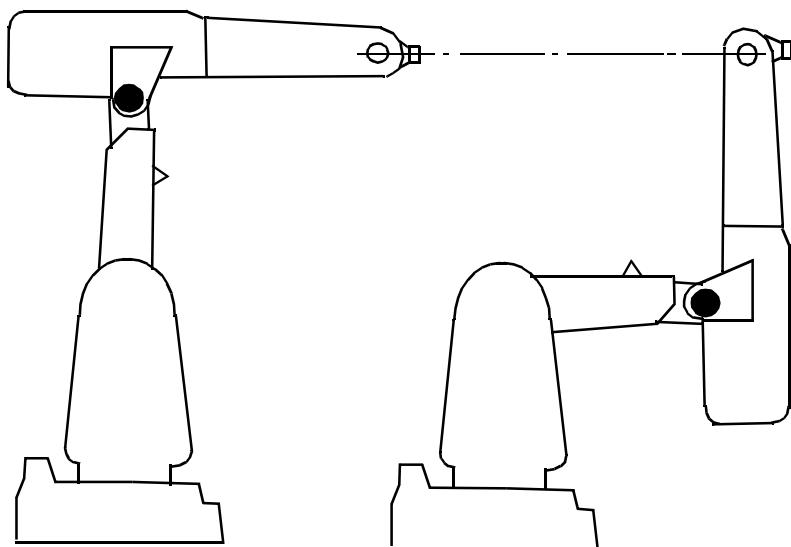


Figura 33 Dos configuraciones de brazo diferentes utilizadas para alcanzar la misma posición y orientación. En una de las configuraciones, los brazos apuntan hacia arriba y para alcanzar la otra configuración, el eje 1 deberá ser girado de 180 grados.

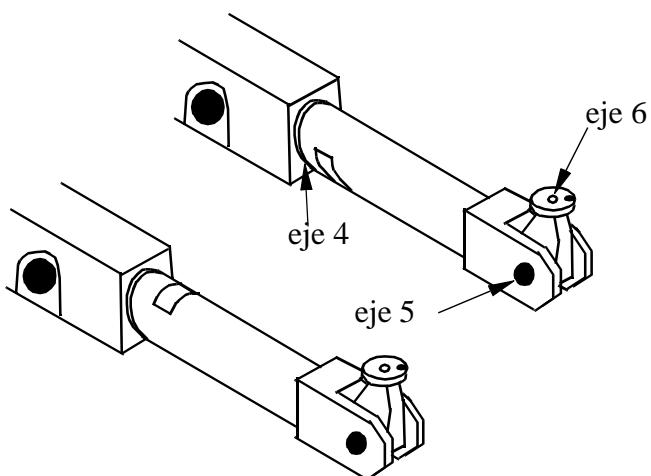


Figura 34 Dos configuraciones de muñeca distintas utilizadas para alcanzar la misma posición y orientación. En la configuración en la que la parte delantera del brazo superior apunta hacia arriba (la de abajo), el eje 4 ha sido girado de 180 grados, el eje 5 de 180 grados, y el eje 6 de 180 grados con el objetivo de alcanzar la configuración en la que la parte delantera del brazo superior apunta hacia abajo (la de arriba).

Lo normal es que se desee que el robot alcance la misma configuración durante la ejecución del programa que la que se ha programado previamente. Para conseguir esto, se puede pedir al robot que compruebe la configuración y, en el caso en que no se obtenga la configuración correcta, la ejecución del programa se parará. Si no se realiza ninguna comprobación de la configuración, el robot podrá empezar a mover sus brazos y muñeca de forma inesperada, lo que podría provocar una colisión con el equipo periférico.

La configuración del robot se especifica definiendo los cuadrantes apropiados para los ejes 1, 4 y 6. Si tanto el robot como la posición programada tienen el mismo cuadrante para estos ejes, significa que la configuración del robot es correcta.

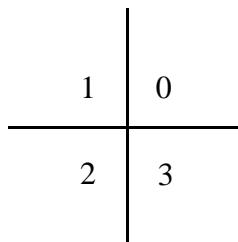


Figura 35 Cuadrante para un ángulo positivo de un eje: $\text{int}\left(\text{eje} - \frac{\text{angulo}}{\pi/2}\right)$.

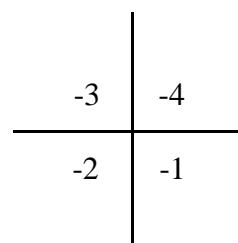


Figura 36 Cuadrante para un ángulo negativo de un eje: $\text{int}\left(\text{eje} - \frac{\text{angulo}}{\pi/2} - 1\right)$.

La comprobación de la configuración implica la comparación de la configuración de la posición programada con la del robot.

Durante el movimiento lineal, el robot siempre se moverá a la configuración más cercana posible a la programada. Si, además, la comprobación de la configuración está activada, la ejecución del programa se detendrá en cuanto alguno de los ejes se desvíe más que el número especificado de grados.

Si se utiliza la comprobación de la configuración durante el movimiento eje a eje, o durante un movimiento lineal modificado, el robot siempre se moverá a la configuración de ejes programada. Si no se consigue la posición y orientación programadas, la ejecución del programa se detiene antes de iniciar el movimiento. En el caso en que la comprobación de la configuración no esté activada, el robot se moverá a la posición y orientación especificadas adoptando la configuración más cercana a la programada.

Cuando la ejecución de una posición programada se para debido a un error de

configuración, el motivo causante podrá ser alguno de los siguientes:

- La posición ha sido programada off-line con una configuración incorrecta.
- Se ha cambiado la herramienta del robot y ello ha obligado el robot a tomar otra configuración que la programada.
- La posición ha sido sometida a una operación de estructura activa (desplazamiento, usuario, objeto, base).

La configuración correcta en la posición de destino podrá encontrarse posicionando el robot cerca de ella y leyendo la configuración en la unidad de programación.

En el caso en que los parámetros de configuración cambien debido a una operación de estructura activa, se podrá desactivar la comprobación de configuración.

4.1 Datos para la configuración del robot 6400C

El IRB 6400C es ligeramente diferente en cuanto a su configuración. La diferencia radica en la interpretación del dato de configuración confdata *cf1*.

cf1 sirve para seleccionar una de las dos posibles configuraciones de los ejes principales (ejes 1, 2 y 3):

- *cf1* = 0 es la configuración hacia adelante
- *cf1* = 1 es la configuración hacia atrás.

La Figura 37 muestra un ejemplo de una configuración hacia adelante y de una configuración hacia atrás que dan como resultado la misma posición y la misma orientación.

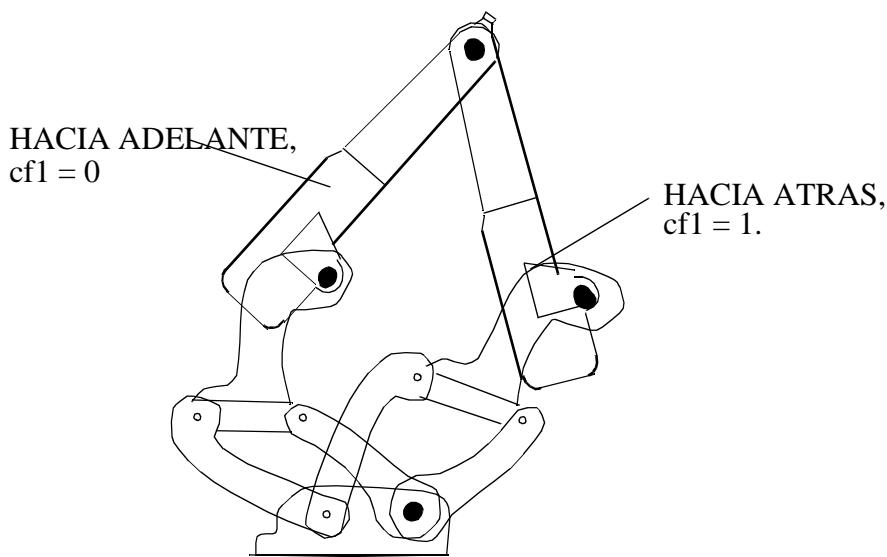


Figura 37 Misma posición y orientación obtenida con dos configuraciones distintas de ejes principales.

La configuración hacia adelante es la parte delantera del área de trabajo del robot con el brazo dirigido hacia adelante. La configuración hacia atrás es la parte trasera del área de trabajo con el brazo dirigido hacia atrás.

4.2 Información relacionada

Descripción en:

Definición de la configuración del robot	Tipos de Datos - <i>confdata</i>
Activación/desactivación de la comprobación de la configuración	Resumen RAPID - <i>Características de Movimiento</i>

5 Puntos Singulares

Ciertas posiciones del área de trabajo del robot podrán alcanzarse mediante un número infinito de configuraciones del robot para el posicionamiento y la orientación de la herramienta. Estas posiciones, conocidas bajo el nombre de puntos singulares (singularidades), constituyen un problema en el momento de calcular los ángulos del brazo del robot basándose en la posición y orientación de la herramienta.

De forma general, un robot puede tener dos tipos de singularidades: las singularidades de brazo y las singularidades de muñeca. Las singularidades de brazo son siempre configuraciones en que el centro de la muñeca (la intersección de los ejes 4, 5 y 6) termina directamente por encima del eje 1 (véase la Figura 38). Las singularidades de muñeca son configuraciones en que el eje 4 y el eje 6 se encuentran en la misma línea, es decir, que el eje 5 forma un ángulo igual a 0 (véase la Figura 39).

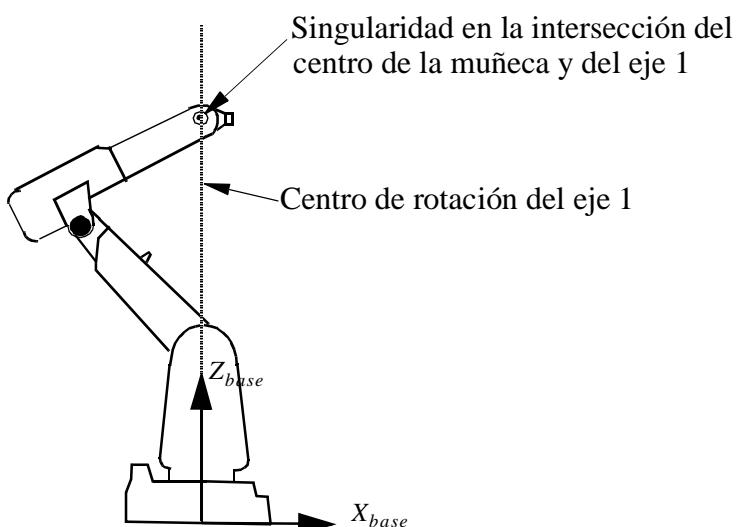


Figura 38 La singularidad del brazo se produce cuando el centro de la muñeca y el eje 1 se encuentran en un punto de intersección.

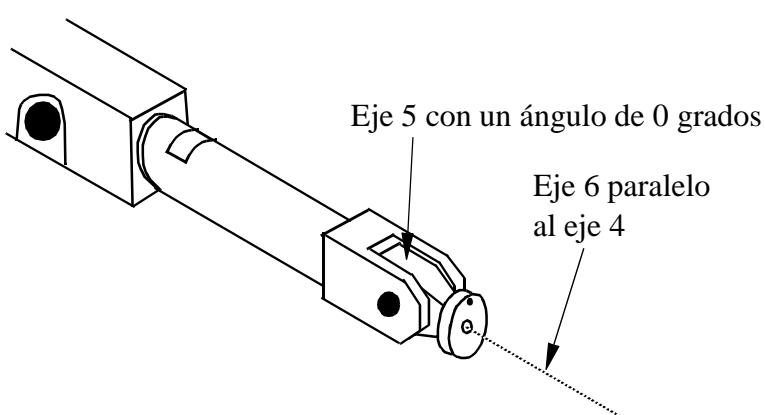


Figura 39 La singularidad de la muñeca se produce cuando el eje 5 es igual a 0 grados.

Puntos singulares/IRB 6400C

Entre el área de trabajo del robot, con el brazo dirigido hacia adelante y el brazo dirigido hacia atrás, hay un punto singular situado encima del robot. También hay un punto singular en los lados del robot. Estos puntos contienen una singularidad y tienen limitaciones cinemáticas. Una posición en estos puntos no podrá ser especificada como delantera/trasera y sólo podrá ser alcanzada con *MoveAbsJ*. Cuando el robot está en un punto singular:

- Sólo se podrá utilizar *MoveAbsJ* o mover el robot eje a eje.

5.1 Ejecución del programa con puntos singulares

Durante la interpolación eje a eje, el robot no tendrá nunca ningún problema para pasar por los puntos singulares.

Durante la ejecución de una trayectoria lineal o circular cerca de un punto singular, las velocidades de ciertos ejes (1 y 6/4 y 6) pueden ser muy elevadas. Para no exceder las velocidades máximas de los ejes, se reducirá la velocidad de la trayectoria lineal.

Las altas velocidades de los ejes pueden reducirse utilizando el modo (*Area Sing |Muñeca*) cuando los ejes de la muñeca están interpolados en ángulos de ejes mientras siguen manteniendo la trayectoria lineal de la herramienta del robot. Un error de orientación comparado con la interpolación lineal total será introducido.

Tener en cuenta que la configuración del robot cambia drásticamente cuando el robot pasa junto a un punto singular con una interpolación lineal o circular. Para evitar tener que realizar una nueva configuración, se deberá programar la primera posición en el otro lado de la singularidad con una orientación que hace que la reconfiguración es innecesaria.

Tener en cuenta también que el robot no debe estar en un punto singular cuando sólo se mueven los ejes externos, puesto que ello puede provocar que los ejes del robot realicen movimientos incontrolados.

5.2 Movimiento con puntos singulares

Durante la interpolación eje a eje, el robot no tendrá nunca ningún problema para pasar por puntos singulares.

Durante la interpolación lineal, el robot podrá pasar por puntos singulares pero a una velocidad reducida.

5.3 Información relacionada

Descripción:

Control de como debe actuar el robot en la ejecución de posiciones cerca de puntos singulares

Instrucciones - *SingArea*

7 Zonas Mundo

7.1 Utilización de las Zonas

Con esta función, el robot se detiene o una salida queda automáticamente activada cuando el robot se encuentra dentro de una área especialmente definida por el usuario. Esta función puede usarse en varios casos, por ejemplo:

- Cuando dos robots comparten una sección de sus áreas de trabajo. Con la supervisión de estas señales, los robots aseguran que no entrarán en colisión el uno con el otro.
- Cuando algún equipo externo se encuentra dentro del área de trabajo del robot. Se podrá crear una área “prohibida” para impedir que el robot colisione con este equipo.
- Cuando hay indicación de que el robot se encuentra en una posición donde se da la posibilidad de iniciar la ejecución del programa desde un PLC.

7.2 Utilidad de las Zonas Mundo

Sirven para:

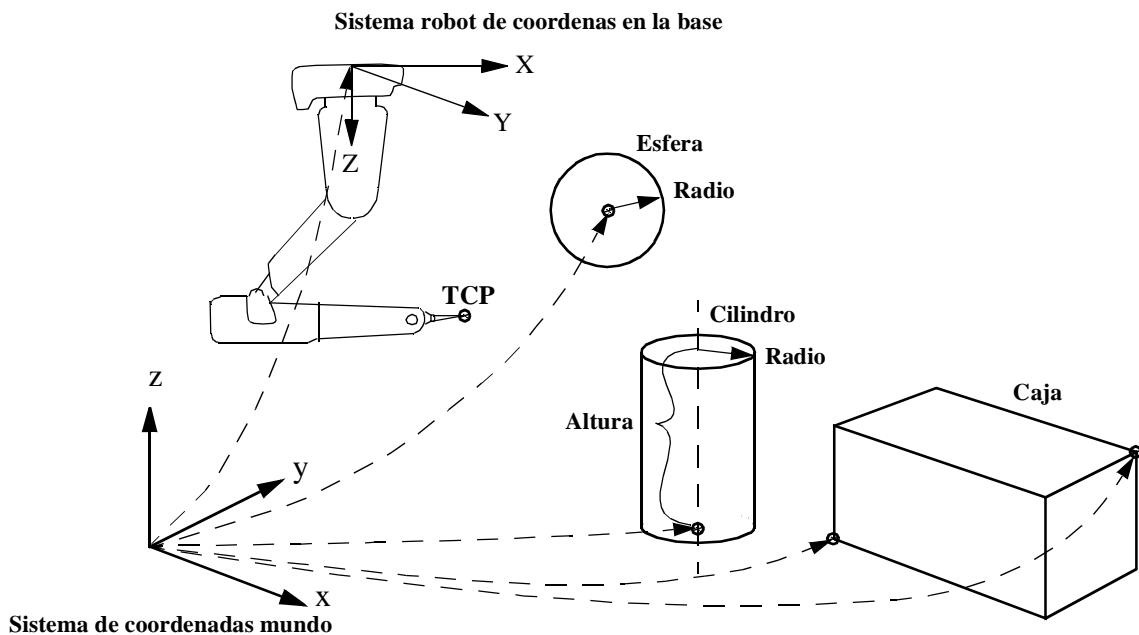
Indicar que el punto central de la herramienta se encuentra en un lugar específico del área de trabajo.

Limitar el área de trabajo del robot con el objeto de evitar colisiones con la herramienta.

Crear una área común para dos robots pero disponible para uno sólo a la vez.

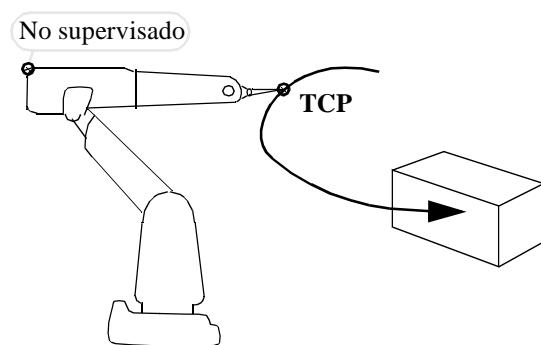
7.3 Definición de las Zonas Mundo en el sistema de coordenadas mundo

Todas las Zonas Mundo deben ser definidas en el sistema de coordenadas mundo. Los laterales de la Caja son paralelos a los ejes coordinados y el eje del Cilindro es paralelo al eje Z del sistema de coordenadas mundo.



Una Zona Mundo puede definirse como estando dentro o fuera de la forma de la Caja, de la Esfera o del Cilindro.

7.4 Supervisión del TCP del robot



Lo que se supervisa es el movimiento del punto central de la herramienta, y ningún otro punto del robot.

El TCP es siempre supervisado independientemente del modo de funcionamiento, por ejemplo, del funcionamiento del programa y del movimiento manual.

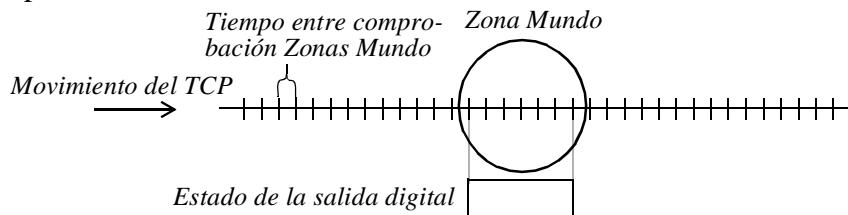
7.4.1 TCPs estacionarios

Si el robot está sujetando un objeto de trabajo y está trabajando con una herramienta estacionaria, se usará un TCP estacionario. Si esta herramienta está activada, la herramienta no se moverá y si está dentro de una Zona Mundo, entonces permanecerá siempre dentro.

7.5 Acciones

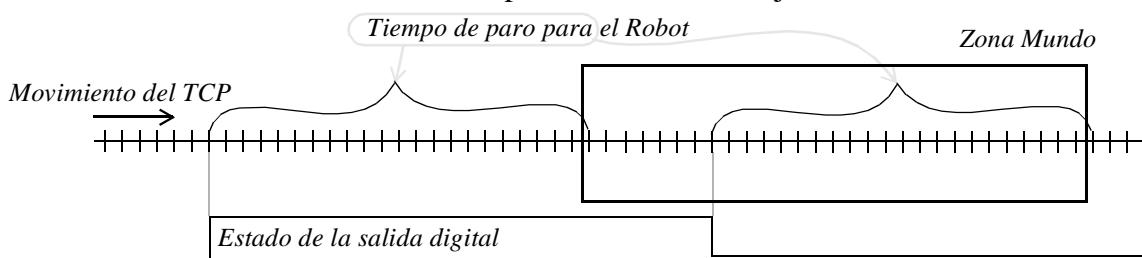
7.5.1 Activación de una salida digital cuando el TCP está dentro de una Zona Mundo

Esta acción tiene por objeto la activación de una salida digital cuando el TCP está dentro de una Zona Mundo. Sirve para indicar que el robot se ha detenido dentro de una área especificada.



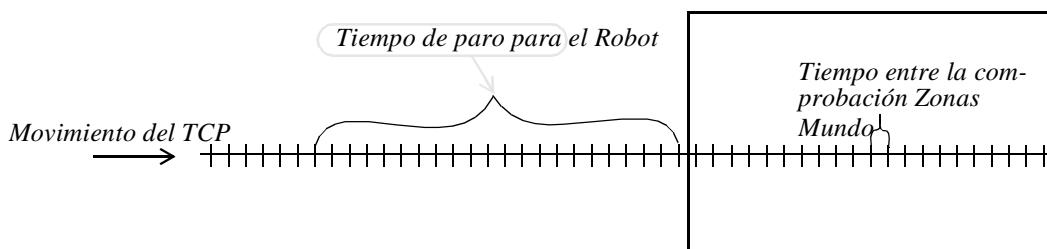
7.5.2 Activación de una salida digital antes de que el TCP alcance una Zona Mundo

Esta acción tiene por objeto la activación de una salida digital antes de que el TCP alcance una Zona Mundo. Sirve para detener el robot justo dentro una Zona Mundo.



7.5.3 Paro del robot antes de que el TCP alcance una Zona Mundo

Se podrá definir una Zona Mundo fuera del área de trabajo. En tal caso el robot se parará con el Punto Central de la Herramienta situado justo fuera de la Zona Mundo al dirigirse hacia la Zona.



En el caso en que el robot haya sido movido dentro de una Zona Mundo definida como estando fuera del área de trabajo, por ejemplo, al liberar los frenos y al mover el robot manualmente, la única forma de salir de dicha zona será moviendo o empujando manualmente con los frenos liberados.

7.6 Tamaño mínimo de las Zonas Mundo

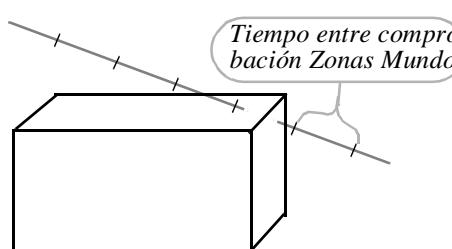
La supervisión del movimiento de los Puntos Centrales de la Herramienta se realiza en puntos concretos con un tiempo intermedio que dependerá de la resolución de la trayectoria.

Le incumbirá al usuario delimitar zonas suficientemente grandes de forma que el robot no pueda atravesar una zona sin ser sometido a una comprobación dentro de la

Tamaño mínimo de zona para una resolución de trayectoria y una vel. max.				
veloc. resol.	1000 mm/s	2000 mm/s	4000 mm/s	
1	25 mm	50 mm	100 mm	
2	50 mm	100 mm	200 mm	
3	75 mm	150 mm	300 mm	

Zona.

Si se usa la misma salida digital para más de una Zona Mundo, la distancia entre las Zonas deberá exceder el tamaño mínimo, de acuerdo con la tabla anterior, a fin de evitar un estado incorrecto para la salida.



Si el tiempo dentro de la zona es demasiado corto, es posible que el robot pase por la esquina de una zona sin verla. Por ello, se deberá asegurar que la zona sea mayor que el área peligrosa.

7.7 Reinicio después de un corte del suministro de alimentación

Las Zonas Mundo Estacionarias quedarán borradas cuando se desactive la alimentación y deberán ser reinsertadas cuando se active la alimentación mediante una rutina de evento conectada al evento POWER ON.

Las Zonas Mundo Temporales permanecerán después de una desactivación de la alimentación pero quedarán borradas cuando se cargue un programa nuevo o cuando se inicie un programa a partir de la rutina principal main.

Las salidas digitales de las Zonas Mundo serán actualizadas cuando se activen los motores (Motors On).

7.8 Información relacionada

Guía de Referencia RAPID

Principios Movimiento y de E/S:

Tipos de Datos:

Instrucciones:

Sistemas de Coordenadas

wztemporary

wzstationary

shapedata

WZBoxDef

WZSphDef

WZCylDef

WZLimSup

WZDOSet

WZDisable

WZEnable

WZFree

6 Principios de E/S

El robot suele tener una o más tarjetas de E/S. Cada una de las tarjetas tiene varios canales digitales y/o analógicos que deberán ser conectados a señales lógicas antes de poder ser utilizados. Esta operación se lleva a cabo en los parámetros del sistema y normalmente ya se ha realizado utilizando nombres estándar antes de la entrega del sistema. Los nombres lógicos deberán utilizarse siempre durante la programación.

Se podrá conectar un canal físico a varios nombres lógicos, aunque también puede ocurrir que un canal físico no tenga ninguna conexión lógica (véase la Figura 40).

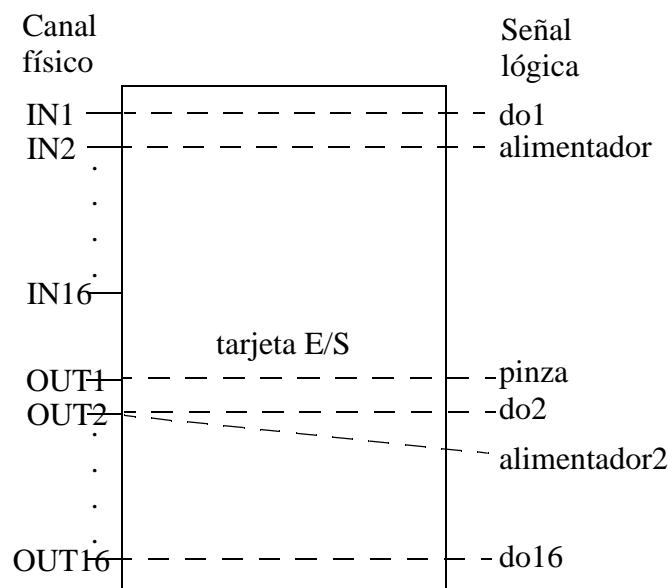


Figura 40 Para poder utilizar una tarjeta de E/S, se deberá atribuir nombres lógicos a sus canales. En el ejemplo de la figura, la salida física 2 ha sido conectada a dos nombres lógicos diferentes. Por otra parte, IN16, no tiene ningún nombre lógico atribuido y por lo tanto no podrá ser utilizado.

6.1 Características de las señales

Las características de una señal dependen del canal físico utilizado así como de la definición del canal en los parámetros del sistema. El canal físico determina los retardos y los niveles de tensión (referirse al documento Especificación del Producto). Las características, el filtrado y el escalado entre los valores físicos y los valores programados están definidos en los parámetros del sistema.

Cuando se conecta la alimentación al robot, todas las señales pasan a 0, y no se verán afectadas, por los paros de emergencia o eventos similares.

Una salida puede activarse en 1 o 0 desde dentro del programa. Esto también puede lle-

varse a cabo utilizando un retardo en el cambio o un pulso. En el caso en que se ordene para una salida un pulso o un cambio con retardo, la ejecución del programa continua. El cambio se realizará sin afectar el resto de la ejecución del programa. Por otra parte, en el caso en que se ordene un nuevo cambio para la misma salida antes de que haya transcurrido el tiempo especificado, el primer cambio será anulado y no se llevará a cabo (véase la Figura 41).

```
SetDO \SDelay:=1, do1;
WaitTime 0.5;
PulseDO do1;
```

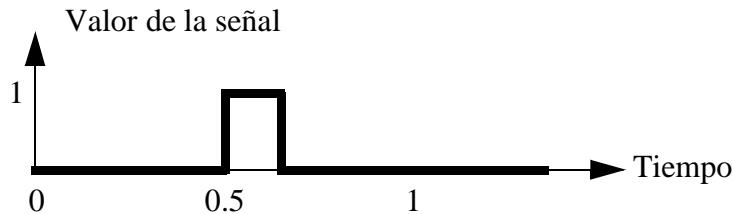


Figura 41 La instrucción SetDO no se llevará a cabo porque se ha introducido una nueva orden antes de que el tiempo de espera haya transcurrido.

6.2 Señales del sistema

Las señales lógicas podrán ser conectadas entre ellas mediante la utilización de unas funciones especiales del sistema. En el caso, por ejemplo, en que se haya conectado una entrada a la función del sistema *Iniciar*, se generará automáticamente un arranque de programa en cuanto esta entrada esté habilitada. Estas funciones del sistema suelen ser habilitadas únicamente en el modo automático. Para más información, referirse al Capítulo 9, Parámetros del Sistema o al capítulo referido a la *Instalación y Puesta en marcha - Comunicación PLC* del Manual de Producto.

6.3 Conexiones enlazadas

Las señales digitales pueden ser conectadas entre ellas de forma que automáticamente

se influyan mutuamente:

- Una señal de salida puede estar conectada a una o más señales de entrada o de salida.
- Una señal de entrada puede estar conectada a una o más señales de entrada o de salida.
- Si la misma señal ha sido utilizada en varias señales enlazadas, el valor de esta señal será el mismo que el último valor habilitado (cambiado).
- Las conexiones enlazables podrán ser interconectadas, en otras palabras, una señal enlazada puede afectar a otra. No deberán, por supuesto, estar conectadas de modo a formar un bucle sinfín; por ejemplo, no se deberá nunca enlazar la señal *di1* a *di2* mientras que la señal *di2* está enlazada a la señal *di1*.
- En el caso en que haya una señal enlazada a una señal de entrada, la conexión física correspondiente será automáticamente inhabilitada. Cualquier cambio en este canal físico no será detectado.
- Los pulsos o los retrasos no serán transmitidos a las señales enlazadas.
- Las condiciones lógicas podrán ser definidas utilizando NOT, AND y OR (Opción: *Funciones avanzadas*).

Ejemplos:

- *di2=di1*
- *di3=di2*
- *do4=di2*

Si *di1* cambia, *di2*, *di3* y *do4* serán cambiados al valor correspondiente.

- *do8=do7*
- *di8=di5*

Si *do7* se activa a 1, *do8* también lo será a 1. Si *di5* se activa posteriormente en 0, *do8* también cambiará (a pesar de que *do7* esté todavía en 1).

- *do5=di6 y do1*

Do5 está activado en 1 si *di6* y *do1* están en 1.

6.4 Limitaciones

Se podrán generar pulsos a un máximo de 10 señales a la vez y sólo se podrán retardar 20 señales a la vez.

6.5 Información relacionada

Descripción:

Definición de las tarjetas
y de las señales de E/S

Guía del Usuario - Parámetros del Sis-
tema

Instrucciones de manipulación de las E/S

Resumen RAPID - *Señales de Entrada*
y Salida

Manipulación manual de las E/S

Guía del Usuario - Entradas y Salidas

INDICE

bool	Valores lógicos
clock	Medida del tiempo
confdata	Datos de configuración del robot
corrdescr	Descriptor de generador de correcciones
dionum	Valores digitales 0 - 1
errnum	Número de error
extjoint	Posición de los ejes externos
intnum	Identificación de la interrupción
iodev	Canales y archivos de serie
jointtarget	Datos de posición de los ejes
loaddata	Datos de carga
mecunit	Unidad mecánica
motsetdata	Datos de movimiento
num	Valores numéricos (registros)
orient	Orientación
o_jointtarget	Dato de posición original de un eje
o_robtarget	Datos de posición original
pos	Posiciones (sólo X, Y y Z)
pose	Transformación de coordenadas
progdisp	Desplazamiento de programa
robjoint	Posición de los ejes del robot
robtarget	Datos de posición
shapedata	Datos de forma de una zona mundo
signalxx	Señales digitales y analógicas
speeddata	Datos de velocidad
string	Cadenas de Carácteres
symnum	Número simbólico
tooldata	Datos de herramienta
tpnum	Número de ventana de la Unidad de Programación
trigedata	Eventos de posicionamiento - disparo
tunetype	Tipo de ajuste servo
wobjdata	Datos del objeto de trabajo
wzstationary	Datos de zona mundo estacionaria
wztemporary	Datos de zona mundo temporal
zonedata	Datos de zona
Datos del Sistema	

Tipos de Datos

bool**Valores lógicos**

Bool se usa para los valores lógicos true/false (verdadero/falso).

Descripción

El valor del tipo de dato *bool* puede ser *TRUE (VERDADERO)* o *FALSE (FALSO)*.

Ejemplos

flag1 := TRUE;

flag tiene asignado el valor *TRUE*.

VAR bool valor1;
VAR num reg1;

.
valor1:= reg1 > 100;

valor1 tiene asignado el valor *TRUE* si *reg1* es mayor que 100; de lo contrario, le será asignado *FALSE*.

IF valor1 Set do1;

La señal *do1* será activada si *valor1* es *TRUE*.

valor1 := reg1 > 100;
valor2 := reg1 > 20 AND NOT valor1;

valor2 tiene asignado el valor *TRUE* si *reg1* se encuentra entre 20 y 100.

Información Relacionada

Expresiones lógicas

Operaciones que usan valores lógicos

Descrita en:

Características Básicas - *Expresiones*

Características Básicas - *Expresiones*

clock

Medida del tiempo

Clock se usa para la medida del tiempo. Las funciones *clock* se utilizan como un cronómetro.

Descripción

Los datos del tipo *clock* almacenan una medida del tiempo en segundos y tienen una resolución de 0,01 segundos.

Ejemplo

```
VAR clock reloj1;
```

```
ClkReset reloj1;
```

El reloj, *reloj1*, es declarado y puesto a cero. Antes de utilizar *ClkReset*, *ClkStart*, *ClkStop* y *ClkRead*, se deberá declarar una variable de tipo de dato *clock* en el programa.

Limitaciones

El tiempo máximo que puede ser almacenado en una variable de reloj es de aproximadamente 49 días (4.294.967 segundos). Las instrucciones *ClkStart*, *ClkStop* y *ClkRead* indican desbordamiento de reloj en el caso poco probable en que ocurra una.

Un reloj deberá declararse como un tipo de variable *VAR*, no como un tipo de variable *persistente*.

Características

Clock es un tipo de datos sin valor y no puede ser utilizado en operaciones orientadas a valores.

Información Relacionada

Descripción:

Resumen de Instrucciones de Fecha y Hora

Resumen RAPID - *Sistema y Hora*

Características de los datos sin valor

Características Básicas - *Tipos de Datos*

confdata**Datos de configuración del robot**

Confdato sirve para definir las configuraciones de los ejes del robot.

Descripción

Todas las posiciones del robot están definidas y almacenadas utilizando coordenadas rectangulares. Al calcular las posiciones de los ejes correspondientes, habrá a menudo dos o más soluciones posibles. Ello significa que el robot es capaz de alcanzar la misma posición, es decir, que la herramienta es capaz de llegar a la misma posición y con la misma orientación, mediante varias posiciones o configuraciones diferentes de los ejes del robot.

Para determinar sin ambigüedad una de estas configuraciones posibles, será especificada utilizando cuatro valores de ejes. Estos valores definen el cuadrante utilizado de los cuatro ejes del robot. Los cuadrantes están numerados según el orden 0, 1, 2, etc. (también pueden ser negativos). El número del cuadrante está conectado al ángulo de unión utilizado del eje. Para cada eje, el cuadrante 0 es el primer cuarto, de 0 a 90° , en una dirección positiva partiendo desde la posición 0; el cuadrante 1 es el cuarto siguiente, de 90 a 180° , etc. El cuadrante -1 es el cuarto 0° a (-90°) , etc. (véase la Figura 1).

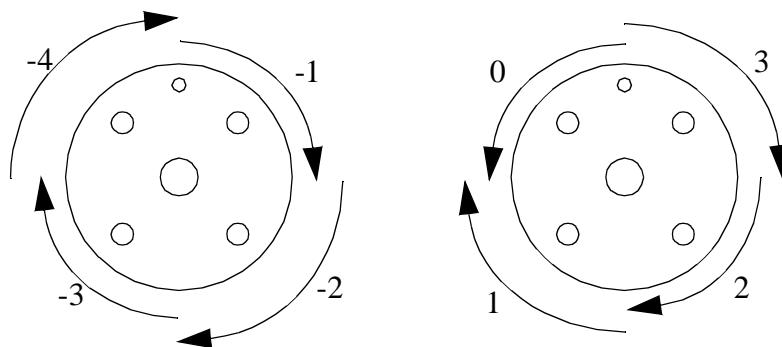


Figura 1 Los cuadrantes de configuración del eje 6.

Datos de configuración del robot para los modelos IRB1400, 2400, 3400, 4400, 6400

Sólo se usan los parámetros de configuración cf1, cf4 y cf6.

Datos de configuración del robot para el modelo IRB5400

Se usan los cuatro parámetros de configuración. cf1, cf4, cf6 para los ejes 1, 4 y 6 respectivamente y cfx para el eje 5.

Datos de configuración del robot para el modelo IRB6400C

El modelo IRB 6400C requiere una configuración ligeramente diferente. La diferencia radica en la interpretación de los datos de configuración *confdata cf1*.

cf1 sirve para seleccionar una de las dos posibles configuraciones de ejes principales (ejes 1, 2 y 3):

- *cf1 = 0* es la configuración hacia adelante
- *cf1 = 1* es la configuración hacia atrás

La Figura 2 muestra un ejemplo de una configuración hacia adelante y de una configuración hacia atrás que dan como resultado la misma posición y orientación.

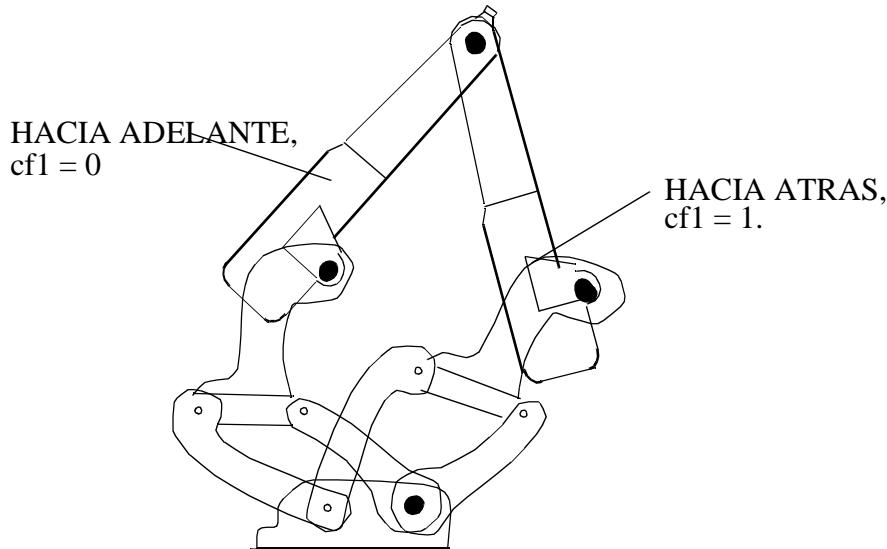


Figura 2 Se obtiene la misma posición y orientación con dos configuraciones de ejes principales diferentes.

La configuración hacia adelante está formada por la parte delantera del área de trabajo del robot con el brazo dirigido hacia adelante. La configuración hacia atrás está formada por la parte posterior del área de trabajo del robot con el brazo dirigido hacia atrás.

Componentes

cf1

Tipo de dato: *num*

El cuadrante utilizado del eje 1, expresado en un número entero negativo o positivo.

cf4

Tipo de dato: *num*

El cuadrante utilizado del eje 4, expresado en un número entero negativo o posi-

tivo.

cf6

Tipo de dato: *num*

El cuadrante utilizado del eje 6, expresado en un número entero negativo o positivo.

cfx

Tipo de dato: *num*

El cuadrante utilizado del eje 5 para el robot IRB 5400, expresado en un número entero negativo o positivo.

Ejemplo

```
VAR confdata conf15 := [1, -1, 0, 0]
```

La configuración *conf15* del robot está definida de la siguiente manera:

- La configuración del eje 1 del robot es el cuadrante *1*, es decir, 90-180°.
- La configuración del eje 4 del robot es el cuadrante *-1*, es decir, 0-(-90°).
- La configuración del eje 6 del robot es el cuadrante *0*, es decir, 0 - 90°.
- La configuración del eje 5 del robot es el cuadrante *0*, es decir, 0 - 90°.

Estructura

```
< estructura de confdata >
  < cf1 de tipo num >
  < cf4 de tipo num >
  < cf6 de tipo num >
  < cfx de tipo num >
```

Información relacionada

Descripción:

Sistemas de coordenadas

Principios de Movimiento y de E/S -
Sistemas de Coordenadas

Manipulación de los datos de configuración

Principios de Movimiento y de E/S -
Configuración del Robot

corrdescr Descriptor de generador de correcciones

Corrdescr (Correction generator descriptor) es utilizado por los generadores de correcciones. Un generador de correcciones añade offsets geométricos en el sistema de coordenadas de la trayectoria.

Descripción

Los datos del tipo *corrdescr* contienen una referencia a un generador de correcciones.

La conexión a un generador de correcciones se realiza mediante la instrucción *CorrCon* y el descriptor (la referencia al generador de correcciones) puede usarse para proporcionar offsets geométricos en el sistema de coordenadas de la trayectoria con la instrucción *CorrWrite*.

Los offsets proporcionados anteriormente pueden ser eliminados desconectando un generador de correcciones mediante la instrucción *CorrDiscon*. Todos los generadores de corrección conectados podrán ser eliminados mediante la instrucción *CorrClear*.

La función *CorrRead* retorna la suma de todos los offsets proporcionados hasta el momento (incluyendo todos los generadores de corrección conectados).

Ejemplo

```
VAR corrdescr id;  
VAR pos offset;  
...  
CorrCon id;  
offset := [1, 2 ,3];  
CorrWrite id, offset;
```

Para conectar un generador de corrección se utiliza la instrucción *CorrCon* y su referencia viene indicada por el descriptor *id*. Luego, los offsets son proporcionados al generador de corrección (con la referencia *id*) mediante la instrucción *CorrWrite*.

Características

Corrdescr es un tipo de dato sin valor.

Información relacionada

Descripción:

Conexión a un generador de correcciones	Instrucciones - <i>CorrCon</i>
Desconexión de un generador de correcciones	Instrucciones - <i>CorrDiscon</i>
Escritura en un generador de correcciones	Instrucciones - <i>CorrWrite</i>
Lectura de todos los offsets actuales	Funciones - <i>CorrRead</i>
Eliminación de todos los generadores de correcciones	Instrucciones - <i>CorrClear</i>
Características de un tipo de dato sin valor	Características - <i>Tipos de Datos</i>

dionum

Valores digitales 0 - 1

Dionum (entradas y salidas digitales numéricas) se usa para los valores digitales (0 o 1).

Este tipo de datos se utiliza en combinación con instrucciones y funciones que manipulan señales de entrada y salida digitales.

Descripción

El tipo de datos *dionum* representa un valor digital 0 o 1.

Ejemplos

CONST dionum cerrado := 1;

Definición de una constante *cerrado* con un valor igual a 1.

SetDO pinza1, cerrado;

La señal *pinza1* está activada en *cerrado*, es decir, en 1.

Manipulación de errores

Si un argumento del tipo *dionum* tiene un valor que no es igual ni a 0 ni a 1, se genera un error en la ejecución del programa.

Características

Dionum es otro nombre de un tipo de datos *num* y por consiguiente hereda sus características.

Información Relacionada

Descripción:

Instrucciones resumidas de entradas/salidas

Resumen RAPID -
Señales de Entradas y Salidas

Configuración de las E/S

Guía del Usuario - *Parámetros del Sistema*

Tipos de datos en general

Características Básicas - *Tipos de Datos*

errnum**Número de error**

Errnum sirve para describir todos los errores recuperables (que no son fatales) que ocurren durante la ejecución del programa, como por ejemplo, la división por cero.

Descripción

Cuando el robot detecta un error durante la ejecución del programa, éste puede ser procesado por el gestor de errores de la rutina. Ejemplos típicos de este tipo de errores son los valores que son demasiado elevados y la división por cero. La variable del sistema *ERRNO*, del tipo *errnum*, tiene asignados diferentes valores que dependen de la naturaleza del error. El gestor de errores es capaz de corregir un error leyendo esta variable, para que la ejecución del programa pueda proseguir de forma adecuada.

Utilizando la instrucción RAISE puede crearse un error desde dentro del programa. Este tipo específico de error puede ser detectado en el gestor de errores especificando un número de error (entre 1-90 o utilizado con la instrucción *BookErrNo*) como un argumento de RAISE.

Ejemplos

```
reg1 := reg2 / reg3;
```

```
. ERROR
```

```
  IF ERRNO = ERR_DIVZERO THEN
    reg3 := 1;
    RETRY;
  ENDIF
```

Si *reg3* = 0, el robot detecta un error cuando la división tiene lugar. Este error, no obstante, será detectado y corregido asignando a *reg3* el valor 1. Después de ésto, la división puede volver a realizarse y la ejecución del programa puede continuar.

```
CONST errnum error_maq := 1;
```

```
. IF di1=0 RAISE error_maq;
```

```
. ERROR
```

```
  IF ERRNO=error_maq RAISE;
```

Se produce un error en una máquina (detectado mediante una señal de entrada *di1*). Se produce un salto al gestor de error de la rutina que, a su vez, llama al gestor de error de la rutina que ha efectuado la llamada a la rutina actual donde el error podrá posiblemente ser corregido. La constante, *error_maq*, sirve para hacer saber al gestor de error qué tipo de error ha ocurrido.

Datos predefinidos

La variable del sistema ERRNO se utiliza para leer el último error que ha ocurrido. Se pueden utilizar una serie de constantes predefinidas para determinar el tipo de error que ha ocurrido.

<u>Nombre</u>	<u>Causa del error</u>
ERR_ALRDYCNT	La variable de interrupción sigue estando conectada a una rutina de tratamiento de interrupciones
ERR_ARGDUPCND	Hay más de un argumento condicional presente para el mismo parámetro
ERR_ARGNNAME	El argumento es una expresión, no está presente o es del tipo switch cuando se ejecuta ArgName
ERR_ARGNOTPER	El argumento no es una referencia persistente
ERR_ARGNOTVAR	El argumento no es una referencia variable
ERR_AXIS_ACT	El eje no está activado
ERR_AXIS_IND	El eje no es independiente
ERR_AXIS_MOVING	El eje se está moviendo
ERR_AXIS_PAR	El parámetro eje en la instrucción TestSign y SetCurrRef es incorrecto.
ERR_CALLIO_INTER	Cuando una demanda IOEnable o IODisable es interrumpida por otra demanda en la misma unidad
ERR_CALLPROC	Ha ocurrido un error de llamada de procedimiento durante el funcionamiento (late binding)
ERR_CNTOTVAR	El objetivo CONNECT no es una referencia variable
ERR_CNV_NOT_ACT	El transportador no está activado.
ERR_CNV_CONNECT	La instrucción <i>WaitWobj</i> está ya activada.
ERR_CNV_DROPPED	El objeto que la instrucción <i>WaitWobj</i> estaba esperando ha sido soltado.
ERR_DEV_MAXTIME	Tiempo excedido cuando se ejecuta una instrucción ReadBin, ReadNum o ReadStr
ERR_DIVZERO	División por cero
ERR_EXCRTYMAX	Número máximo de reintentos excedido
ERR_EXECPHR	Se ha realizado un intento de ejecutar una instrucción utilizando un comodín
ERR_FILEACC	Acceso incorrecto a un archivo
ERR_FILENOFND	No se ha encontrado el archivo

ERR_FILEOPEN	No se puede abrir un archivo
ERR_FNCNORET	Ningún valor de retorno
ERR_ILLDIM	Dimensión incorrecta de matriz
ERR_ILLQUAT	Intento de utilización de una válvula de orientación incorrecta (cuaternion)
ERR_ILLRAISE	El número de error en RAISE está fuera de alcance
ERR_INOMAX	No hay más números de interrupciones disponibles
ERR_IOENABLE	Tiempo excedido cuando se ejecuta IOEnable
ERR_IODISABLE	Tiempo excedido cuando se ejecuta IODisable
ERR_MAXINTVAL	El valor entero es demasiado elevado
ERR_NAME_INVALID	Si el nombre de la unidad no existe o si no se permite que la unidad sea inhabilitada
ERR_NEGARG	El argumento negativo no está permitido
ERR_NOTARR	El dato no es una matriz
ERR_NOTEQDIM	La dimensión de matriz utilizada al llamar la rutina no coincide con sus parámetros
ERR_NOTINTVAL	No es un valor entero
ERR_NOTPRES	Se ha utilizado un parámetro a pesar del hecho de que el argumento correspondiente no ha sido utilizado a la llamada de rutina
ERR_OUTOFBND	El índice de matriz se encuentra fuera de los límites permitidos
ERR_REFUNKDAT	Referencia a datos de objetos enteros desconocidos
ERR_REFUNKFUN	Referencia a una función desconocida
ERR_REFUNKPRC	Referencia a un procedimiento desconocido en el momento del enlace o durante el funcionamiento (late binding)
ERR_REFUNKTRP	Referencia a una rutina de tratamiento de interrupciones desconocida
ERR_PATHDIST	Distancia de regreso demasiado grande para la instrucción StartMove
ERR_RCVDATA	Se ha realizado un intento de leer un dato no numérico xxx con ReadNum
ERR_SC_WRITE	Error producido cuando se ha realizado un envío al computador externo
ERR_SIGSUPSEARCH	La señal ya tiene un valor positivo al principio del proceso de búsqueda
ERR_STEP_PAR	Paso del parámetro en SetCurrRef es incorrecto
ERR_STRTOOLNG	La cadena es demasiado larga
ERR_SYM_ACCESS	Error de símbolo de acceso lectura/escritura
ERR_TP_DIBREAK	Una instrucción TPRead ha sido interrumpida por una entrada digital
ERR_TP_MAXTIME	Tiempo excedido cuando se ejecuta una instrucción TPRead
ERR_UNIT_PAR	El parámetro Unidad_Mec en TestSign y SetCurrRef es incorrecto

ERR_UNKINO	Número de interrupción desconocido
ERR_UNLOAD	Error cargado
ERR_WAIT_MAXTIME	Tiempo excedido cuando se ejecuta una instrucción WaitDI o WaitUntil
ERR_WHLSEARCH	Sin paro de búsqueda

Características

Errnum es un nombre equivalente de un tipo de dato *num* y por consiguiente hereda sus características.

Información relacionada

Recuperación de errores

Descripción en:

Resumen RAPID - *Recuperación de errores*
Características Básicas - *Recuperación de errores*

Tipos de datos en general, y otros equivalentes

Características Básicas - *Tipos de Datos*

extjoint**Posición de los ejes externos**

Extjoint sirve para definir las posiciones de los ejes externos, de los posicionadores o de los manipuladores de piezas de trabajo.

Descripción

El robot puede controlar hasta seis ejes externos además de sus seis ejes internos, es decir, un total de doce ejes. Los seis ejes externos son denominados de forma lógica: a, b, c, d, e, f. Cada uno de estos ejes lógicos podrá ser conectado a un eje físico y, en este caso, la conexión será definida en los parámetros del sistema.

Los datos del tipo *extjoint* sirven para almacenar los valores de posición para cada uno de los ejes lógicos de a - f.

La posición de cada eje lógico conectado a un eje físico será definida según las siguientes instrucciones:

- Para los ejes rotativos – la posición será definida como la rotación en grados a partir de la posición de calibración.
- Para los ejes lineales – la posición será definida como la distancia en mm a partir de la posición de calibración.

En el caso en que un eje lógico no esté conectado a un eje físico, el valor 9E9 será utilizado como un valor de posición, indicando que el eje no está conectado. En el momento de la ejecución, los datos de posición de cada eje serán comprobados y también se comprobará si el eje correspondiente está conectado o no. Si el valor de posición almacenado no corresponde con la conexión del eje actual, se aplicará lo siguiente:

- Si la posición no ha sido definida en los datos de posición (el valor es 9E9), el valor será ignorado aunque el eje esté conectado y no activado. Pero si el eje está activado, se producirá un error.
- Si la posición ha sido definida en los datos de posición, aunque el eje no esté conectado, no se tendrá en cuenta el valor.

En el caso en que se utilice un offset de eje externo (instrucción *EOffsOn* o *EOffsSet*), las posiciones serán especificadas en el sistema de coordenadas ExtOffs.

Componentes

eax_a (*eje externo a*)

Tipo de dato: *num*

La posición del eje lógico externo “a”, expresado en grados o en mm (según el tipo de eje utilizado).

eax_b (<i>eje externo b</i>)	Tipo de dato: <i>num</i>
La posición del eje lógico externo “b”, expresado en grados o en mm (según el tipo de eje utilizado).	
...	
eax_f (<i>eje externo f</i>)	Tipo de dato: <i>num</i>
La posición del eje lógico externo “f”, expresado en grados o en mm (según el tipo de eje utilizado).	

Ejemplo

```
VAR extjoint posex10 := [ 11, 12.3, 9E9, 9E9, 9E9, 9E9 ] ;
```

La posición de un posicionador externo, *posex10*, será definida según lo siguiente:

- La posición del eje lógico externo “a” es 11, expresado en grados o mm (según el tipo de eje utilizado).
 - La posición del eje lógico externo “b” es 12.3, expresado en grados o en mm (según el tipo de eje utilizado).
 - Los ejes del c al f no han sido definidos.

Estructura

```
< dataobject de extjoint >
  < eax_a de num >
  < eax_b de num >
  < eax_c de num >
  < eax_d de num >
  < eax_e de num >
  < eax_f de num >
```

Información relacionada

	<u>Descripción:</u>
Datos de posición	Tipos de datos - <i>robtarget</i>
Sistema de coordenadas ExtOffs	Instrucciones - <i>EOffsOn</i>

intnum

Identificación de la interrupción

Intnum (interrupt numeric) sirve para identificar una interrupción.

Descripción

Cuando se relaciona una variable del tipo *intnum* a una rutina de tratamiento de interrupción, se le da un valor específico que identifica la interrupción. Esta variable es posteriormente utilizada cada vez que se procesa una interrupción, como cuando se la llama o cuando se la inhabilita.

Se puede relacionar más de una identificación de interrupción a la misma rutina de tratamiento de interrupción. La variable del sistema *INTNO* podrá así ser utilizada en una rutina de tratamiento de la interrupción para determinar el tipo de interrupción que se produce.

Ejemplos

```
VAR intnum error_transp;  
.  
CONNECT error_transp WITH correg_transp;  
ISignalDI di1, 1, error_transp;
```

Se generará una interrupción cuando la entrada *di1* se activa en 1. Cuando esto ocurre, se realiza una llamada a la rutina de tratamiento de interrupción *correg_transp*.

```
VAR intnum error_transp1;
VAR intnum error_transp2;

PROC init_interrupt();

    CONNECT error_transp1 WITH correg_transp;
    ISignalDI di1, 1, error_transp1;
    CONNECT error_transp2 WITH correg_transp;
    ISignalDI di2, 1, error_transp2;

ENDPROC

TRAP correg_transp
IF INTNO=error_transp1 THEN
ELSE
ENDIF

ENDTRAP
```

Se generará una interrupción cuando cualquier de las entradas *di1* o *di2* se activan en 1. Entonces se realiza una llamada a la rutina de tratamiento de interrupciones *correg_transp*. La variable del sistema INTNO es utilizada en la rutina de tratamiento de interrupciones para descubrir qué tipo de interrupción ha ocurrido.

Limitaciones

El número máximo de variables activas del tipo *intnum* al mismo tiempo (entre *CONNECT* y *IDelete*) está limitado a 40. El número máximo de interrupciones en la cola para la ejecución de una rutina *TRAP* al mismo está limitado a 30.

Características

Intnum es otro nombre de un tipo de datos *num* y por consiguiente se beneficia de las mismas características.

Información Relacionada

Descripción:

Resumen de interrupciones
Otros tipos de datos

Resumen RAPID - *Interrupciones*
Características Básicas -
Tipos de Datos

iodev

Canales y archivos de serie

Iodev (I/O device) se usa para los canales serie, como impresoras y archivos.

Descripción

El tipo de datos *iodev* contiene una referencia a un archivo o a un canal serie. Puede estar enlazado a la unidad física mediante la instrucción *Open* y luego ser utilizado para la lectura y escritura.

Ejemplo

```
VAR iodev archivo5;  
.  
Open "flp1:LOGDIR/INFIL.DOC", file\Read;  
input := ReadNum(archivo5);
```

Se abre el archivo *INFIL.DOC* en el directorio LOGDIR del disco para la lectura. Para la lectura del archivo, *archivo5* se utiliza como una referencia en lugar del nombre del archivo.

Características

Iodev es un tipo de dato sin valor.

Información Relacionada**Descrita en:**

Comunicación a través de canales serie

Resumen RAPID- *Comunicación*

Configuración de los canales serie

Guía del Usuario - *Parámetros del Sistema*

Características de los tipos de datos sin valor

Características Básicas - *Tipos de Datos*

jointtarget**Datos de posición de los ejes**

Jointtarget sirve para definir la posición a la que el robot y los ejes externos se moverán utilizando la instrucción *MoveAbsJ*.

Descripción

Jointtarget define la posición de cada eje individual, tanto para el robot como para los ejes externos.

Componentes**robax**

(ejes del robot)

Tipo de dato: *robjoint*

Son las posiciones de los ejes del robot expresadas en grados.

La posición de los ejes es definida como la rotación en grados del eje (brazo) correspondiente en una dirección positiva o negativa a partir de la posición de calibración del eje.

extax

(ejes externos)

Tipo de dato: *extjoint*

La posición de los ejes externos.

La posición es definida según las siguientes indicaciones para cada eje individual (*eax_a*, *eax_b* ... *eax_f*):

- Para los ejes de rotación, la posición es definida como la rotación, expresada en grados, a partir de la posición de calibración.
- Para los ejes lineales, la posición es definida como la distancia, expresada en mm, a partir de la posición de calibración.

Los ejes externos *eax_a* ... son ejes lógicos. La relación existente entre el número del eje lógico y el número del eje físico se encuentra definida en los parámetros del sistema.

El valor 9E9 será definido para los ejes que no estén conectados. En el caso en que los ejes definidos en los datos de posición difieran de los ejes que están realmente conectados durante la ejecución del programa, se aplicará lo siguiente:

- En el caso en que la posición no esté definida en los datos de posición (valor 9E9) no se tendrá en cuenta el valor, si el eje está conectado y no activado. Pero si el eje está activado, el sistema generará un error.
- En el caso en que la posición sea definida en los datos de posición y que el eje todavía no esté conectado, no se tendrá en cuenta el valor.

Ejemplos

```
CONST jointtarget calib_pos := [ [ 0, 0, 0, 0, 0, 0], [ 0, 9E9, 9E9, 9E9, 9E9, 9E9] ];
```

La posición de calibración normal para el IRB2400 se encuentra definida en *calib_pos* mediante el tipo de dato *jointtarget*. La posición de calibración normal 0 (expresada en grados o en mm) también estará definida para el eje lógico externo a. En cambio, los ejes externos b a f no están definidos.

Estructura

```
< dataobject de jointtarget >
  < robax de robjoint >
    < rax_1 de num >
    < rax_2 de num >
    < rax_3 de num >
    < rax_4 de num >
    < rax_5 de num >
    < rax_6 de num >
  < extax de extjoint >
    < eax_a de num >
    < eax_b de num >
    < eax_c de num >
    < eax_d de num >
    < eax_e de num >
    < eax_f de num >
```

Información relacionada

Movimiento a la posición de un eje
Instrucciones de posicionamiento
Configuración de los ejes externos

Descripción:

Instrucciones - *MoveAbsJ*
Resumen RAPID - *Movimiento*
Guía del Usuario - *Parámetros del Sistema*

loaddata**Datos de carga**

Loaddata sirve para describir las cargas vinculadas al interface mecánico del robot (brida de montaje del robot).

Los datos de carga suelen definir la carga útil (la carga de la pinza es definida por la instrucción *GripLoad*) del robot, es decir, la carga soportada por el útil del robot. La carga de la herramienta (peso) está especificada en los datos de la herramienta (*tool-data*) que incluyen los datos de carga.

Descripción

Las cargas especificadas sirven para determinar un modelo de dinámica del robot, de forma que los movimientos del robot puedan ser controlados de la mejor manera posible.



Es importante definir siempre la carga actual de la herramienta y cuando se use, la carga útil del robot. Definiciones incorrectas de los datos de carga pueden provocar una sobrecarga de la estructura mecánica del robot.

Si se especifican datos incorrectos de carga, se pueden generar las siguientes consecuencias:

- Si el valor de los datos de carga especificados es mayor que el valor real de la carga;
 - > El robot no será utilizado a su capacidad máxima
 - > La precisión de trayectoria es incorrecta incluyendo el riesgo de vibraciones.
 - > Existe un riesgo de sobrecarga de la estructura mecánica
- Si el valor de los datos de carga especificados es menor que el valor real de la carga;
 - > La precisión de trayectoria es incorrecta incluyendo el riesgo de vibraciones.
 - > Existe un riesgo de sobrecarga de la estructura mecánica

La carga útil será activada/desactivada utilizando la instrucción *GripLoad*.

Componentes**mass**

Tipo de dato: *num*

El peso de la carga en kg.

cog

(*centro de gravedad*)

Tipo de dato: *pos*

El centro de gravedad de la carga de la herramienta será expresado utilizando el sistema de coordenadas de la muñeca. Si se usa una herramienta estacionaria, se referirá al centro de gravedad de la herramienta que sujetá el objeto de trabajo.

El centro de gravedad de una carga útil será expresado utilizando el sistema de

coordenadas de la herramienta. Si se usa una herramienta estacionaria, entonces se utilizará un sistema de coordenadas del objeto.

aom

(ejes del momento)

Tipo de dato: *orient*

La orientación del sistema de coordenadas definido por los ejes inerciales de la carga de la herramienta. **Deberá estar siempre activado en 1, 0, 0, 0.** Expresado en el sistema de coordenadas de la muñeca como un quaternion (q1, q2, q3 y q4). Si se usa una herramienta estacionaria, se referirá a los ejes inerciales de la herramienta que sujetó el objeto de trabajo.

La orientación del sistema de coordenadas definido por los ejes inerciales de la carga útil. **Deberá estar siempre activado en 1, 0, 0, 0.** Expresado en el sistema de coordenadas de la herramienta como un quaternion (q1, q2, q3 y q4). Si se usa una herramienta estacionaria, se utilizará un sistema de coordenadas del objeto.

Restricción en la orientación del sistema de coordenadas de la carga de la herramienta y de la carga útil (ejes de momento -aom):

La orientación del sistema de coordenadas de la carga de la herramienta debe coincidir con la orientación del sistema de coordenadas de la muñeca.

La orientación del sistema de coordenadas de la carga útil debe coincidir con la orientación del sistema de coordenadas de la muñeca. Debido a ello, la mejor manera consiste en definir la orientación del sistema de coordenadas de la herramienta (tool frame) para que coincida con la orientación del sistema de coordenadas de la muñeca.

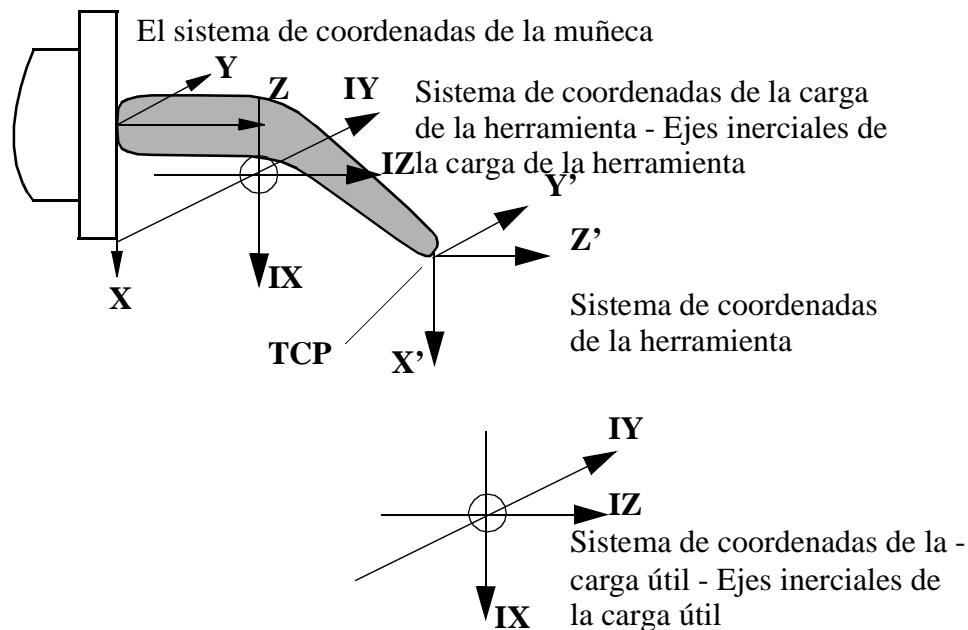


Figura 1 Restricción en la orientación del sistema de coordenadas de la carga de la herramienta y de la carga útil.

ix

(momento de inercia alrededor de x)

Tipo de dato: num

El momento de inercia de la carga alrededor del eje x', respecto a su centro de masa de los ejes del sistema de coordenadas de la carga útil y de la carga de la herramienta, expresado en kgm^2 .

Una definición correcta de los momentos de inercia permitirá una utilización óptima de la planificación de la trayectoria y del control de los ejes. Ello puede ser muy importante cuando por ejemplo se manipulan grandes láminas de metal, etc. Si el momento de inercia para todos los componentes ix , iy , iz es equivalente a 0 kgm^2 , implica una carga puntual.

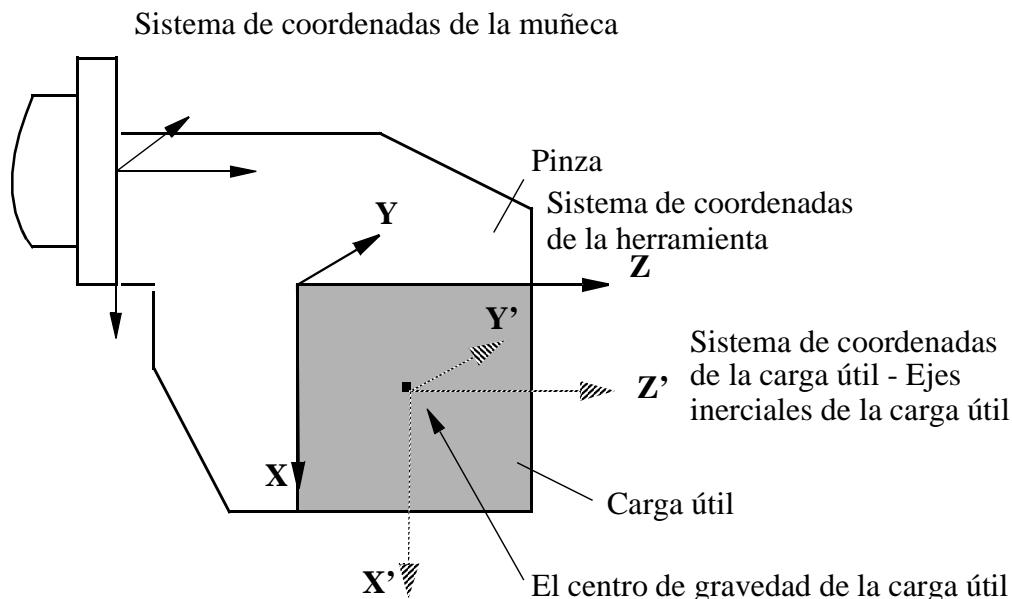


Figura 2 El centro de gravedad de la carga útil y los ejes iniciales.

Normalmente los momentos de inercia sólo deberán ser definidos cuando la distancia entre la brida de montaje y el centro de gravedad sea menor que la dimensión de la carga (véase la Figura 3).

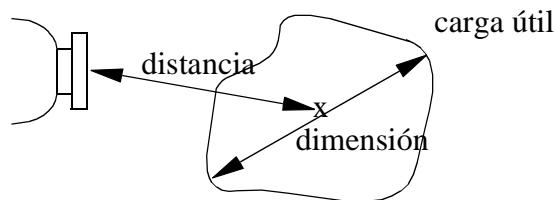


Figura 3 Normalmente, el momento de inercia deberá ser definido cuando la distancia sea menor que la dimensión de la carga.

iy

(momento de inercia alrededor de y)

Tipo de dato: num

El momento de inercia de la carga alrededor del eje y' expresado en kgm^2 .

Para más información, véase *ix*.

iz (*momento de inercia alrededor de z*) Tipo de dato: *num*

El momento de inercia de la carga alrededor del eje z' expresado en kgm^2 .

Para más información, véase *ix*.

Ejemplos

```
PERS loaddata pieza1 := [ 5, [50, 0, 50], [1, 0, 0, 0], 0, 0, 0];
```

Para definir la carga útil de la Figura 2 se deberá utilizar los siguientes valores:

- Peso de 5 kg.
- El centro de gravedad es $x = 50$, $y = 0$, $z = 50$ mm en el sistema de coordenadas de la herramienta.
- La carga útil es una carga puntual.

```
Set pinza;  
WaitTime 0.3;  
GripLoad pieza1;
```

La conexión de la carga útil, *pieza1*, se especifica en el mismo momento en que el robot coge la carga *pieza1*.

```
Reset pinza;  
WaitTime 0.3;  
GripLoad load0;
```

La desconexión de una carga útil, se especifica en el mismo momento en que el robot deja la carga.

Limitaciones

La carga útil deberá ser definida únicamente como una variable persistente (PERS) y no dentro de una rutina. Los valores utilizados serán entonces almacenados al guardar el programa en un disquete y volverán ser utilizados en el momento en que se carguen.

Los argumentos del tipo datos de carga (loaddata) en la instrucción *GripLoad* deberá ser un dato persistente entero (no un elemento matriz ni un componente de registro).

Datos predefinidos

La carga *load0* define una carga útil de un peso equivalente a 0 kg, es decir ninguna carga. Esta carga se utiliza como argumento de la instrucción *GripLoad* para desconectar una carga útil.

Siempre se podrá acceder a la carga *load0* desde el programa, sin embargo no podrá ser cambiada (se encuentra almacenada en el módulo del sistema *BASE*).

PERS loaddata load0 := [0.001, [0, 0, 0.001], [1, 0, 0, 0], 0, 0, 0];

Estructura

```
< dataobject de loaddata >
  < mass de num >
  < cog de pos >
    < x de num >
    < y de num >
    < z de num >
  < aom de orient >
    < q1 de num >
    < q2 de num >
    < q3 de num >
    < q4 de num >
  < ix de num >
  < iy de num >
  < iz de num >
```

Información relacionada

Descripción:

Sistemas de coordenadas

Principios de Movimiento y de E/S -
Sistemas de Coordenadas

Definición de cargas de la herramienta

Tipos de datos - *tooldata*

Activación de las cargas útiles

Instrucciones - *GripLoad*

mecunit**Unidad mecánica**

Mecunit sirve para definir las diferentes unidades mecánicas que pueden ser controladas y a las que se puede acceder desde el robot y desde el programa.

Los nombres de las unidades mecánicas están definidos en los parámetros del sistema y por lo tanto, no deberán ser definidos en el programa.

Descripción

Los tipos de dato *mecunit* contienen únicamente una referencia a la unidad mecánica.

Limitaciones

Los tipos de datos *mecunit* no deberán ser definidos en el programa. El tipo de dato podrá, no obstante, ser utilizado como un parámetro al declarar una rutina.

Datos predefinidos

Siempre se podrá acceder desde el programa (datos instalados) a las unidades mecánicas definidas en los parámetros del sistema.

Características

Mecunit es un tipo de dato *sin valor*. Esto significa que los datos de este tipo no permiten la realización de operaciones con valores.

Información relacionada**Descripción:**

Activación/Desactivación de las unidades mecánicas

Instrucciones - *ActUnit*, *DeactUnit*

Configuración de las unidades mecánicas

Guía del Usuario - *Parámetros del Sistema*

Características de los tipos de datos sin valor

Características Básicas - *Tipos de Datos*

motsetdata**Datos de movimiento**

Motsetdata sirve para definir una serie de características de movimiento que afectan a todas las instrucciones de posicionamiento del programa:

- Velocidad máxima y corrección de la velocidad
- Datos de aceleración
- Comportamiento del sistema cerca de un punto singular
- Manipulación de las diferentes configuraciones del robot
- Carga útil
- Ajuste de la resolución de la trayectoria

Normalmente este tipo de datos no debe ser usado ya que estas características deben ser determinadas únicamente mediante las instrucciones *VelSet*, *AccSet*, *SingArea*, *ConfJ*, *ConfL*, *GripLoad* y *PathResol*.

Se podrá acceder a los valores de estas características de movimiento utilizando la variable del sistema *C_MOTSET*.

Descripción

Las características de movimiento utilizadas (almacenadas en la variable del sistema *C_MOTSET*) afectan a todos los movimientos.

Componentes**vel.oride**Tipo de dato: *veldata/num*

La velocidad como un porcentaje de la velocidad programada.

vel.maxTipo de dato: *veldata/num*

Velocidad máxima en mm/s.

acc.accTipo de dato: *accdata/num*

Aceleración y deceleración como un porcentaje de los valores normales.

acc.rampTipo de dato: *accdata/num*

El nivel con el que la aceleración y la deceleración aumenta como un porcentaje de los valores normales.

sing.wrist	Tipo de dato: <i>singdata/bool</i>
Se permite variar la orientación de la herramienta para evitar una singularidad de la muñeca.	
sing.arm	Tipo de dato: <i>singdata/bool</i>
Se permite variar ligeramente la orientación de la herramienta para evitar una singularidad del brazo (no implementado).	
sing.base	Tipo de dato: <i>singdata/bool</i>
No se acepta ninguna desviación de la orientación de la herramienta.	
conf.jsup	Tipo de dato: <i>confsupdata/bool</i>
La supervisión de la configuración de los ejes está activa durante el movimiento eje a eje.	
conf.lsup	Tipo de dato: <i>confsupdata/bool</i>
La supervisión de la configuración de los ejes está activa durante el movimiento lineal y circular.	
conf.ax1	Tipo de dato: <i>confsupdata/num</i>
Desviación máxima permitida en grados para el eje 1 (no utilizado en esta versión).	
conf.ax4	Tipo de dato: <i>confsupdata/num</i>
Desviación máxima permitida en grados para el eje 4 (no utilizado en esta versión).	
conf.ax6	Tipo de dato: <i>confsupdata/num</i>
Desviación máxima permitida en grados para el eje 6 (no utilizado en esta versión).	
grip.load	Tipo de dato: <i>gripdata/loaddata</i>
La carga útil del robot (sin incluir la pinza).	
pathresol	Tipo de dato: <i>num</i>
Ajuste actual en un porcentaje de la resolución de la trayectoria configurada.	

Limitaciones

Sólo uno de los siguientes componentes *sing.wrist*, *sing.arm* o *sing.base* puede tener un valor igual a TRUE.

Ejemplo

```
IF C_MOTSET.vel.oride > 50 THEN
    ...
ELSE
    ...
ENDIF
```

Diferentes partes del programa son ejecutadas en función del ajuste de velocidad utilizado.

Datos predefinidos

C_MOTSET describe las características de movimiento del robot utilizadas y siempre se podrá acceder a ellas desde el programa (datos instalados). Por otra parte, *C_MOTSET*, sólo podrá cambiarse mediante una serie de instrucciones, y no por asignaciones.

Los siguientes valores por defecto de los parámetros de movimiento se activan:

- a la puesta en marcha
- cuando se carga un programa nuevo
- cuando se ejecuta la primera instrucción del programa.

PERS motsetdata C_MOTSET := [
[100, 500],	-> veldata
[100, 100],	-> accdata
[FALSE, FALSE, TRUE],	-> singdata
[TRUE, TRUE, 30, 45, 90],	-> confsupdata
[[0, [0, 0, 0], [1, 0, 0, 0], 0, 0, 0]],	-> gripdata
100]	-> path resolution

Estructura

```

<dataobject of motsetdata>
  <vel of veldata >      -> Es afectado por la instrucción VelSet
    < oride of num >
    < max of num >
  <acc of accdata >      -> Es afectado por la instrucción AccSet
    < acc of num >
    < ramp of num >
  <sing of singdata >      -> Es afectado por la instrucción SingArea
    < wrist of bool >
    < arm of bool >
    < base of bool >
  <conf of confsupdata >      -> Es afectado por las instrucciones ConfJ, ConfL y
                                LimConfL
    < jsup of bool >
    < lsup of bool >
    < ax1 of num >
    < ax4 of num >
    < ax6 of num >
  <grip of gripdata >      -> Es afectado por la instrucción GripLoad
    < load of loaddata >
      < mass of num>
      < cog of pos >
        < x of num >
        < y of num >
        < z of num >
      <aom of orient >
        < q1 of num >
        < q2 of num >
        < q3 of num >
        < q4 of num >
      < ix of num >
      < iy of num>
      < iz of num >
    <pathresol of num>      -> Es afectado por la instrucción PathResol

```

Información relacionada

Descripción:

Instrucciones para la activación
de los parámetros de movimiento

Resumen RAPID -
Características de movimiento

num**Valores numéricos (registros)**

Num se usa para los valores numéricos; por ejemplo, para los contadores.

Descripción

El valor de un tipo de dato *num* puede ser

- un número entero; por ejemplo, -5,
- un número decimal; por ejemplo, 3,45.

También puede estar escrito de forma exponencial; por ejemplo, 2E3 ($= 2 \cdot 10^3 = 2000$), 2,5E-2 ($= 0,025$).

Los números enteros entre -8388607 y +8388608 son siempre almacenados como números enteros exactos.

Los números decimales no son más que números aproximados y no deberán, por ello, ser utilizados en comparaciones del tipo *es igual a* o *no es igual a*. En el caso de divisiones y operaciones que utilizan números decimales, el resultado será también un número decimal; es decir, no un número entero exacto.

E.g.

```
a := 10;
b := 5;
IF a/b=2 THEN
...

```

Dado que el resultado de a/b no es un número entero, esta condición no será necesariamente satisfecha.

Ejemplo

VAR num reg1;

reg1 := 3;

reg1 tiene asignado el valor 3.

a := 10 DIV 3;
b := 10 MOD 3;

División entera donde *a* tiene asignado un número entero (=3) y *b* tiene asignado al resto (=1).

Datos Predefinidos

La constante pi (π) ya ha sido definida en el módulo del sistema *BASE*.

CONST num pi := 3.1415926;

Información Relacionada

Descripción:

Expresiones numéricas

Características Básicas - *Expresiones*

Operaciones que utilizan valores numéricos

Características Básicas - *Expresiones*

***o_jointtarget* Dato de posición original de un eje**

o_jointtarget (*original joint target*) se utiliza en combinación con la función *Absolute Limit Modpos*. Cuando se usa esta función para modificar una posición, la posición original es almacenada como un dato del tipo *o_jointtarget*.

Descripción

Si se ha activado la función *Absolute Limit Modpos* y que se ha modificado con la función *Modpos* una posición con nombre dentro de una instrucción de movimiento, entonces se salvará la posición programada original.

Ejemplo de un programa antes de *Modpos*:

```
CONST jointtarget jpos40 := [[0, 0, 0, 0, 0, 0],  
                           [0, 9E9, 9E9, 9E9, 9E9, 9E9]];  
...  
MoveAbsJ jpos40, v1000, z50, tool1;
```

El mismo programa después *ModPos* en el que el punto *jpos40* ha sido corregido de dos grados para el eje 1 del robot:

```
CONST jointtarget jpos40 := [[2, 0, 0, 0, 0, 0],  
                           [0, 9E9, 9E9, 9E9, 9E9, 9E9]];  
CONST o_jointtarget o_jpos40 := [[0, 0, 0, 0, 0, 0],  
                                 [0, 9E9, 9E9, 9E9, 9E9, 9E9]];  
...  
MoveAbsJ jpos40, v1000, z50, tool1;
```

El punto original programado ha sido salvado en *o_jpos40* (por el tipo de dato *o_jointtarget*) y el punto modificado ha sido salvado en *jpos40* (por el tipo de dato *jointtarget*).

Al salvar el punto original programado, el robot puede monitorizar que posteriores modificaciones *Modpos* del punto correspondiente, están dentro de los límites aceptables desde el punto original programado.

La convención fija del nombre significa que un punto original programado con el nombre *xxxxx* será salvado con el nombre *o_xxxxx* al utilizar *Absolute Limit Modpos*.

Componentes

robax	<i>(ejes del robot)</i>	Tipo de dato: <i>robjoint</i>
--------------	-------------------------	-------------------------------

Posiciones de los ejes del robot expresadas en grados.

extax*(ejes externos)*Tipo de dato: *extjoint*

La posición de los ejes externos.

Estructura

```
< dataobject de o_jointtarget >
  < robax de robjoint >
    < rax_1 de num >
    < rax_2 de num >
    < rax_3 de num >
    < rax_4 de num >
    < rax_5 de num >
    < rax_6 de num >
  < extax de extjoint >
    < eax_a de num >
    < eax_b de num >
    < eax_c de num >
    < eax_d de num >
    < eax_e de num >
    < eax_f de num >
```

Información relacionada

Datos de posición

Descripción en:

Configuración de Limit Modpos

Tipos de Datos - *Jointtarget*

Guía del Usuario - *Parámetros del Sistema*

orient**Orientación**

Orient se usa para las orientaciones (como la orientación de una herramienta) y las rotaciones (como la rotación de un sistema de coordenadas).

Descripción

La orientación se describe bajo la forma de un quaternion formado por cuatro elementos: *q1*, *q2*, *q3* y *q4*. Para más información sobre como se calculan, véase a continuación.

Componentes**q1**Tipo de dato: *num*

Quaternion 1.

q2Tipo de dato: *num*

Quaternion 2.

q3Tipo de dato: *num*

Quaternion 3.

q4Tipo de dato: *num*

Quaternion 4.

Ejemplo

```
VAR orient orient1;
.
orient1 := [1, 0, 0, 0];
```

La orientación *orient1* tiene asignado el valor *q1*=1, *q2*-*q4*=0; estos valores corresponden a ninguna rotación.

Limitaciones

La orientación debe ser normalizada; es decir que la suma de los cuatro debe ser igual a 1:

$$q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1$$

¿Qué es un Quaternion?

La orientación de un sistema de coordenadas (como la de una herramienta) puede ser descrita como una matriz rotacional que describe la dirección de los ejes del sistema de coordenadas respecto a un sistema de referencia (véase la Figura 1).

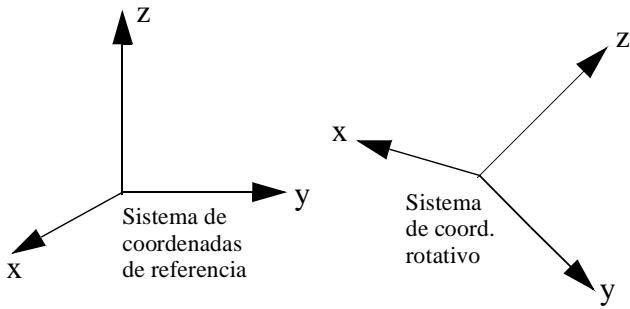


Figura 1 La rotación de un sistema de coordenadas está descrita mediante un quaternion.

Los ejes de los sistemas de coordenadas rotativos (**x**, **y**, **z**) son vectores que pueden ser expresados en el sistema de coordenadas de referencia según lo siguiente:

$$\mathbf{x} = (x_1, x_2, x_3)$$

$$\mathbf{y} = (y_1, y_2, y_3)$$

$$\mathbf{z} = (z_1, z_2, z_3)$$

Esto significa que el componente x- de vector x- en el sistema de coordenadas de referencia será x_1 , el componente y- será x_2 , etc.

Estos tres vectores pueden ser puestos juntos en una matriz rotacional, donde cada uno de los vectores forman una columna:

$$\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{bmatrix}$$

Un quaternion no es más que una forma más concisa de describir esta matriz rotacional; los quaternions se calculan basándose en los elementos de la matriz rotacional:

$$q1 = \frac{\sqrt{x_1 + y_2 + z_3 + 1}}{2}$$

$$q2 = \frac{\sqrt{x_1 - y_2 - z_3 + 1}}{2}$$

$$q3 = \frac{\sqrt{y_2 - x_1 - z_3 + 1}}{2}$$

$$q4 = \frac{\sqrt{z_3 - x_1 - y_2 + 1}}{2}$$

$$\text{signo } q2 = \text{signo } (y_3 - z_2)$$

$$\text{signo } q3 = \text{signo } (z_1 - x_3)$$

$$\text{signo } q4 = \text{signo } (x_2 - y_1)$$

Ejemplo 1

Una herramienta está orientada de forma que su eje Z' apunte recto hacia adelante (en la misma dirección que el eje X del sistema de coordenadas en la base). El eje Y' de la herramienta corresponde al eje Y del sistema de coordenadas en la base (véase la Figura 3). ¿Cómo está definida la orientación de la herramienta en el dato de posición (robtarget)?

La orientación de la herramienta en una posición programada está normalmente relacionada con el sistema de coordenadas del objeto de trabajo utilizado. En este ejemplo, no se utiliza ningún objeto de trabajo y el sistema de coordenadas en la base es igual al sistema de coordenadas global. Así, la orientación es relativa con el sistema de coordenadas en la base.

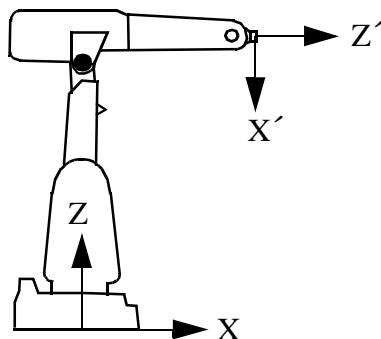


Figura 2 La dirección de una herramienta de acuerdo con el ejemplo 1.

Los ejes estarán relacionados de la siguiente forma:

$$\mathbf{x}' = -\mathbf{z} = (0, 0, -1)$$

$$\mathbf{y}' = \mathbf{y} = (0, 1, 0)$$

$$\mathbf{z}' = \mathbf{x} = (1, 0, 0)$$

Lo cual corresponde a la siguiente matriz rotacional:

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

La matriz rotacional proporciona al quaternion correspondiente:

$$q_1 = \frac{\sqrt{0+1+0+1}}{2} = \frac{\sqrt{2}}{2} = 0,707$$

$$q_2 = \frac{\sqrt{0-1-0+1}}{2} = 0$$

$$q_3 = \frac{\sqrt{1-0-0+1}}{2} = \frac{\sqrt{2}}{2} = 0,707 \quad \text{signo } q_3 = \text{signo } (1+1) = +$$

$$q_4 = \frac{\sqrt{0-0-1+1}}{2} = 0$$

Ejemplo 2

La dirección de la herramienta ha sido girada de 30° en los ejes X'- y Z'-respecto al sistema de coordenadas de la muñeca (véase la Figura 3). ¿Cómo estará definida la ori-

entación de la herramienta en los datos de la herramienta?

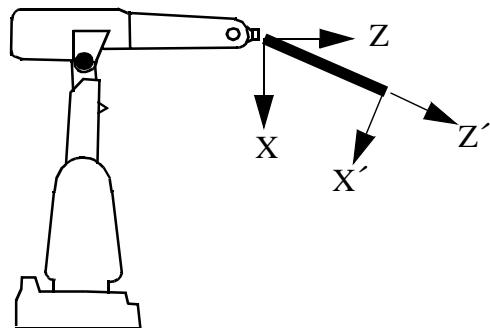


Figura 3 La dirección de la herramienta de acuerdo con el ejemplo2.

Los ejes estarán relacionados de la forma siguiente:

$$\mathbf{x}' = (\cos 30^\circ, 0, -\sin 30^\circ)$$

$$\mathbf{y}' = (0, 1, 0)$$

$$\mathbf{z}' = (\sin 30^\circ, 0, \cos 30^\circ)$$

Lo cual corresponde a la siguiente matriz rotacional: $\begin{bmatrix} \cos 30^\circ & 0 & \sin 30^\circ \\ 0 & 1 & 0 \\ -\sin 30^\circ & 0 & \cos 30^\circ \end{bmatrix}$

La matriz rotacional proporciona al quaternion correspondiente:

$$q_1 = \frac{\sqrt{\cos 30^\circ + 1 + \cos 30^\circ + 1}}{2} = 0,965926$$

$$q_2 = \frac{\sqrt{\cos 30^\circ - 1 - \cos 30^\circ + 1}}{2} = 0$$

$$q_3 = \frac{\sqrt{1 - \cos 30^\circ - \cos 30^\circ + 1}}{2} = 0,258819 \quad \text{signo } q_3 = \text{signo } (\sin 30^\circ + \sin 30^\circ) = +$$

$$q_4 = \frac{\sqrt{\cos 30^\circ - \cos 30^\circ - 1 + 1}}{2} = 0$$

Estructura

```
<dataobject of orient>
<q1 of num>
<q2 of num>
<q3 of num>
<q4 of num>
```

Información Relacionada

Operaciones sobre orientaciones

Descripción:

Características Básicas - *Expresiones*

o_robtarget**Datos de posición original**

o_robtarget (*original robot target*) se utiliza en combinación con la función *Absolute Limit Modpos*. Cuando esta función es utilizada para modificar una posición, la posición original queda almacenada como un dato del tipo *o_robtarget*.

Descripción

En el caso en que la función *Absolute Limit Modpos* ha sido activada y que una posición nombrada en una instrucción de movimiento es modificada con la función *Modpos*, entonces la posición programada original queda almacenada.

Ejemplo de un programa antes de *Modpos*:

```
CONST robtarget p50      := [[500, 500, 500], [1, 0, 0, 0], [1, 1, 0, 0],
                               [500, 9E9, 9E9, 9E9, 9E9, 9E9]];
...
MoveL p50, v1000, z50, tool1;
```

El mismo programa después de *ModPos* en el cual el punto *p50* ha sido corregido a 502 en la dirección -x:

```
CONST robtarget p50      := [[502, 500, 500], [1, 0, 0, 0], [1, 1, 0, 0],
                               [500, 9E9, 9E9, 9E9, 9E9, 9E9]];
CONST o_robtarget o_p50  := [[500, 500, 500], [1, 0, 0, 0], [1, 1, 0, 0],
                               [500, 9E9, 9E9, 9E9, 9E9, 9E9]];
...
MoveL p50, v1000, z50, tool1;
```

El punto original programado ha sido almacenado en *o_p50* (por el tipo de dato *o_robtarget*) y el punto modificado, ha sido almacenado en *p50* (por el tipo de dato *robtarget*).

Al guardar el punto original programado, el robot podrá monitorizar que posteriores modificaciones *Modpos* del punto en cuestión estén dentro de los límites aceptables del punto original programado.

La convención fija del nombre significa que un punto original programado con el nombre *xxxxx* será guardado con el nombre *o_xxxxx* al utilizar *Absolute Limit Modpos*.

Componentes**trans**

(traslación)

Tipo de dato: *pos*

La posición (x, y, z) del punto central de la herramienta expresada en mm.

rot	(<i>rotación</i>)	Tipo de dato: <i>orient</i>
	La orientación de la herramienta, expresada bajo la forma de un cuaternio (q1, q2, q3 y q4).	
robconf	(<i>configuración del robot</i>)	Tipo de dato: <i>confdata</i>
	La configuración de los ejes del robot (cf1, cf4, cf6 y cfx).	
extax	(<i>ejes externos</i>)	Tipo de dato: <i>extjoint</i>
	La posición de los ejes externos.	

Estructura

```
<dataobject de o_robtarget >
  <trans de pos >
    <x de num >
    <y de num >
    <z de num >
  <rot de orient >
    <q1 de num >
    <q2 de num >
    <q3 de num >
    <q4 de num >
  <robconf de confdata >
    <cf1 de num >
    <cf4 de num >
    <cf6 de num >
    <cfx de num >
  <extax de extjoint >
    <eax_a de num >
    <eax_b de num >
    <eax_c de num >
    <eax_d de num >
    <eax_e de num >
    <eax_f de num >
```

Información relacionada

Datos de posición
Configuración de Limit Modpos

Descripción en:

Tipos de Datos - *Robtarget*
Guía del Usuario - *Parámetros del Sistema*

pos**Posiciones (sólo X, Y y Z)**

Pos se usa para las posiciones (sólo X, Y y Z).

El tipo de dato *robtarget* se usa para determinar la posición del robot incluyendo la orientación de la herramienta y la configuración de los ejes.

Descripción

Los datos del tipo *pos* describen la posición de las coordenadas: X, Y y Z.

Componentes

x

Tipo de dato: *num*

El valor X de la posición.

y

Tipo de dato: *num*

El valor Y de la posición.

z

Tipo de dato: *num*

El valor Z de la posición.

Ejemplos

VAR pos pos1;

pos1 := [500, 0, 940];

La posición *pos1* tiene asignado el valor: X=500 mm, Y=0 mm, Z=940 mm.

pos1.x := pos1.x + 50;

La posición *pos1* es desplazada 50 mm en la dirección X.

Estructura

<dataobject of *pos*>
<x of *num*>
<y of *num*>
<z of *num*>

Información Relacionada

Descripción:

Operaciones sobre las posiciones

Características Básicas - *Expresiones*

Posición del robot incluyendo la orientación

Tipos de datos - *robtarget*

pose

Transformación de coordenadas

Pose se usa para cambiar de un sistema de coordenadas a otro.

Descripción

Los datos del tipo *pose* describen como se desplaza un sistema de coordenadas y se hace girar en torno a otro sistema de coordenadas. Los datos pueden, por ejemplo, describir como está posicionado el sistema de coordenadas de la herramienta respecto al sistema de coordenadas de la muñeca.

Componentes

trans	(<i>translación</i>)	Tipo de dato: <i>pos</i>
El desplazamiento en la posición (x, y, z) del sistema de coordenadas.		
rot	(<i>rotación</i>)	Tipo de dato: <i>orient</i>
La rotación del sistema de coordenadas.		

Ejemplo

```
VAR pose base1;  
.  
base1.trans := [50, 0, 40];  
base1.rot := [1, 0, 0, 0];
```

El cambio de coordenadas *base1* tiene asignado un valor que corresponde a un desplazamiento en la posición, donde X=50 mm, Y=0 mm, Z=40 mm; no hay, por lo tanto, ninguna rotación.

Estructura

```
<dataobject of pose>  
  <trans of pos>  
  <rot of orient>
```

Información Relacionada

¿Qué es un quaternion?

Descripción:

Tipos de Datos - *orient*

progdisp

Desplazamiento de programa

Progdisp sirve para almacenar el desplazamiento de programa utilizado del robot y de los ejes externos.

Este tipo de dato normalmente no debe ser utilizado ya que el dato se activará cuando se utilicen las instrucciones *PDispSet*, *PDispOn*, *PDispOff*, *EOffsSet*, *EOffsOn* y *EOffsOff*. Se utiliza únicamente para almacenar de forma temporal el valor utilizado para un uso posterior.

Descripción

Se podrá acceder a los valores utilizados para el desplazamiento del programa utilizando la variable del sistema *C_PROGDISP*.

Para más información, véanse las instrucciones *PDispSet*, *PDispOn*, *EOffsSet* y *EOffsOn*.

Componentes

pdisp (*desplazamiento de programa*)

Tipo de dato: *pose*

El desplazamiento del programa del robot expresado mediante la utilización de una translación y de una orientación. La translación está expresada en mm.

eoffs (*offset externo*)

Tipo de dato: *extjoint*

El offset de cada uno de los ejes externos. Si el eje es lineal, el valor será expresado en mm; si es de rotación, el valor será expresado en grados.

Ejemplo

```
VAR progdisp despl1;  
.  
SearchL sen1, pbusc, p10, v100, herram1;  
PDispOn \ExeP:=pbusc, *, herram1;  
EOffsOn \ExeP:=pbusc, *;  
.  
despl1:=C_PROGDISP;  
PDispOff;  
EOffsOff;  
.  
PDispSet despl1.pdisp;  
EOffsSet despl1.eoffs;
```

En primer lugar, un desplazamiento de programa será activado a partir de una posición buscada. Luego, será desactivado de forma temporal almacenando el valor en la variable *despl1* y, posteriormente, será activada de nuevo mediante las instrucciones *PDispSet* y *EOffsSet*.

Datos predefinidos

La variable del sistema *C_PROGDISP* describe el desplazamiento de programa utilizado del robot y de los ejes externos. Siempre se podrá acceder a ella desde el programa. Por otra parte, *C_PROGDISP*, sólo podrá cambiarse utilizando una serie de instrucciones y no por asignación.

Estructura

```
< dataobject de progdisp >  
<pdisp de pose>  
< trans de pos >  
< x de num >  
< y de num >  
< z de num >  
< rot de orient >  
< q1 de num >  
< q2 de num >  
< q3 de num >  
< q4 de num >  
< edefs de extjoint >  
< eax_a de num >  
< eax_b de num >  
< eax_c de num >  
< eax_d de num >  
< eax_e de num >  
< eax_f de num >
```

Información relacionada

Descripción:

Instrucciones para la definición de un desplazamiento de programa

Resumen RAPID - *Características de Movimiento*

Sistemas de coordenadas

Principios de Movimiento y de E/S - *Sistemas de Coordenadas*

robjoint

Posición de los ejes del robot

Robjoint sirve para definir la posición, expresada en grados, de los ejes del robot.

Descripción

Los datos del tipo *robjoint* sirven para almacenar las posiciones, en grados, de los ejes 1 a 6 del robot. La posición de eje se define como la rotación en grados del eje (brazo) respectivo en una dirección positiva o negativa a partir de la posición de calibración del eje.

Componentes

rax_1	<i>(eje 1 del robot)</i>	Tipo de dato: <i>num</i>
La posición del eje 1 del robot, en grados, a partir de la posición de calibración.		
...		
rax_6	<i>(eje 6 del robot)</i>	Tipo de dato: <i>num</i>
La posición del eje 6 del robot, en grados, a partir de la posición de calibración.		

Estructura

```
< dataobject de robjoint >
  < rax_1 de num >
  < rax_2 de num >
  < rax_3 de num >
  < rax_4 de num >
  < rax_5 de num >
  < rax_6 de num >
```

Información relacionada

	<u>Descripción:</u>
Datos de posición de los ejes	Tipos de Datos - <i>jointtarget</i>
Movimiento a una posición de los ejes	Instrucciones - <i>MoveAbsJ</i>

robtarget

Datos de posición

Robtarget sirve para definir la posición del robot y de los ejes externos.

Descripción

Los datos de posición sirven para definir la posición en las instrucciones de posicionamiento a las que el robot y los ejes externos deben moverse.

Dado que el robot es capaz de alcanzar una misma posición de varias formas diferentes, se deberá especificar la configuración de los ejes. Esto servirá para definir los valores de los ejes en el caso de que sean ambiguos, por ejemplo:

- si el robot está en una posición hacia adelante o hacia atrás,
- si el eje 4 apunta hacia abajo o hacia arriba,
- si el eje 6 tiene una vuelta positiva o negativa.



La posición será definida basándose en el sistema de coordenadas del objeto de trabajo, incluyendo cualquier desplazamiento de programa. Si la posición es programada con cualquier otro objeto de trabajo que el utilizado en la instrucción, el robot no se moverá de la forma esperada. Asegurarse de que se está utilizando el mismo objeto de trabajo que el que se ha utilizado al programar las instrucciones de posicionamiento. Una utilización incorrecta puede provocar daños al personal o al equipo.

Componentes

trans	<i>(traslación)</i>	Tipo de dato: <i>pos</i>
--------------	---------------------	--------------------------

La posición (x, y, z) del punto central de la herramienta expresada en mm.

La posición se especifica respecto al sistema de coordenadas del objeto utilizado, incluyendo el desplazamiento del programa. Si no se especifica ningún objeto de trabajo, se usará el sistema de coordenadas mundo.

rot	<i>(rotación)</i>	Tipo de dato: <i>orient</i>
------------	-------------------	-----------------------------

La orientación de la herramienta expresada bajo la forma de un quaternion (q1, q2, q3 y q4).

La orientación se especifica respecto al sistema de coordenadas del objeto utilizado, incluyendo el desplazamiento del programa. Si no se especifica ningún objeto de trabajo, se usará el sistema de coordenadas mundo.

robconf	(configuración del robot)	Tipo de dato: <i>confdata</i>
----------------	---------------------------	-------------------------------

La configuración de los ejes del robot (cf1, cf4, cf6 y cfx). Esto se define bajo la forma del cuarto utilizado del eje 1, del eje 4 y del eje 6. El primer cuarto positivo 0-90 ° está definido como 0. El componente cfx se utiliza únicamente para el modelo de robot IRB5400.

Para más información, véase el tipo de dato *confdata*.

extax	(ejes externos)	Tipo de dato: <i>extjoint</i>
--------------	-----------------	-------------------------------

La posición de los ejes externos.

La posición será definida como se indica a continuación para cada eje individual (*eax_a*, *eax_b* ... *eax_f*):

- Para los ejes rotativos, la posición será definida como la rotación en grados a partir de la posición de calibración.
- Para los ejes lineales, la posición será definida como la distancia en mm a partir de la posición de calibración.

Los ejes externos *eax_a* ... son ejes lógicos. La forma en que el número de los ejes lógicos y el número de los ejes físicos están relacionados entre sí se encuentra definida en los parámetros del sistema.

El valor 9E9 será definido para los ejes que no están conectados. Si los ejes definidos en los datos de posición difieren de los ejes que están conectados en realidad para la ejecución del programa, se aplicará lo siguiente:

- Si la posición no está definida en los datos de posición (el valor 9E9) no se tendrá en cuenta el valor, si el eje está conectado y no está activado. Pero si el eje está activado, el sistema producirá un error.
- Si la posición está definida en los datos de posición y que el eje todavía no está conectado, no se tendrá en cuenta el valor.

Ejemplos

```
CONST robtarget p15 := [ [600, 500, 225,3], [1, 0, 0, 0], [1, 1, 0, 0],
```

[11, 12.3, 9E9, 9E9, 9E9, 9E9]];

Una posición *p15* será definida de la siguiente manera:

- La posición del robot: $x = 600$, $y = 500$, $z = 225.3$ mm en el sistema de coordenadas del objeto.
- La orientación de la herramienta en la misma dirección que el sistema de coordenadas del objeto.
- La configuración del eje del robot: eje 1 y 4 en la posición 90-180°, el eje 6 en la posición 0-90°.
- La posición de los ejes externos lógicos, a y b, expresada en grados o en mm. (dependiendo del tipo de ejes). Los ejes c y f no están definidos.

```
VAR robtarget p20;
```

```
...
p20 := CRobT();
p20 := Offs(p20,10,0,0);
```

La posición *p20* será activada en la misma posición que la posición actual del robot llamando la función *CRobT*. La posición se moverá entonces de 10 mm en la dirección x-.

Limitaciones

Cuando se utiliza la función de edición configurable *Absolute Limit Modpos*, el número de caracteres del nombre del dato del tipo *robtarget*, está limitado a 14 (en otros casos, está limitado a 16).

Estructura

```
< dataobject de robtarget >
  < trans de pos >
    < x de num >
    < y de num >
    < z de num >
  < rot de orient >
    < q1 de num >
    < q2 de num >
    < q3 de num >
    < q4 de num >
  < robconf de confdata >
    < cf1 de num >
    < cf4 de num >
    < cf6 de num >
    < cfx de num >
  < extax de extjoint >
    < eax_a de num >
    < eax_b de num >
    < eax_c de num >
    < eax_d de num >
    < eax_e de num >
    < eax_f de num >
```

Información relacionada

Instrucciones de posicionamiento

Describa en:

Resumen RAPID- *Movimiento*

Sistemas de coordenadas

Principios de Movimiento y de E/S -
Sistemas de coordenadas

Manipulación de los datos de configuración

Principios de Movimiento y de E/S -
Configuración del Robot

Configuración de los ejes externos

Guía del Usuario - *Parámetros del sistema*

¿Qué es un quaternion?

Tipos de datos - *Orient*

shapedata

Datos de forma de una zona mundo

shapedata sirve para describir la geometría de una zona mundo.

Descripción

Las zonas mundo pueden clasificarse en tres formas geométricas diferentes:

- en forma de una caja rectangular, con todos sus lados paralelos al sistema de coordenadas mundo y definida por una instrucción *WZBoxDef*.
- en forma de una esfera, definida por una instrucción *WZSphDef*.
- en forma de un cilindro, paralelo al eje z del sistema de coordenadas mundo y definida por una instrucción *WZCylDef*.

La geometría de una zona mundo se define con una de las instrucciones anteriores y la acción de una zona mundo se define mediante las instrucciones *WZLimSup* o *WZDO-Set*.

Ejemplo

```
VAR wzstationary pole;  
VAR wzstationary conveyor;  
...  
PROC ...  
    VAR shapedata volume;  
    ...  
    WZBoxDef \Inside, volume, p_corner1, p_corner2;  
    WZLimSup \Stat, conveyor, volume;  
    WZCylDef \Inside, volume, p_center, 200, 2500;  
    WZLimSup \Stat, pole, volume;  
ENDPROC
```

Un *transportador* es definido como un volumen en forma de una caja rectangular y la supervisión para esta área es activada. Un *pole* es definido como un volumen en forma de cilindro y la supervisión para esta zona también es activada. Cuando el robot entra en una de estas áreas, el movimiento queda detenido.

Características

shapedata es un tipo de dato sin valor.

Información relacionada

Descripción:

Definición de una zona mundo en forma de caja rectangular	Instrucciones - <i>WZBoxDef</i>
Definición de una zona mundo en forma de esfera	Instrucciones - <i>WZSphDef</i>
Definición de una zona mundo en forma de cilindro	Instrucciones - <i>WZCylDef</i>
Activación de una área de trabajo restringida	Instrucciones - <i>WZLimSup</i>
Activación de una salida referida a una zona	Instrucciones - <i>WZDOSet</i>

signalxx**Señales digitales y analógicas**

Los tipos de datos incluidos en *signalxx* se usan para las señales digitales y analógicas de entrada y de salida.

Los nombres de las señales están definidos en los parámetros del sistema y por lo tanto no deberán ser definidos en el programa.

Descripción

<u>Tipo de dato</u>	<u>Sirve para</u>
signalai	las señales de entrada analógicas
signalao	las señales de salida analógicas
signaldi	las señales de entrada digitales
signaldo	las señales de salida digitales
signalgi	los grupos de señales de entrada digitales
signalgo	los grupos de señales de salida digitales

Las variables del tipo *signalxo* sólo contienen una referencia a la señal. El valor de la señal es determinado cuando se utiliza una instrucción, por ejemplo, *DOoutput*.

Las variables del tipo *signalxi* contienen una referencia a una señal así como un método para recuperar el valor. El valor de la señal de entrada es devuelto cuando se llama a una función, por ejemplo, *DInput*, o cuando se usa la variable en un contexto de valores, por ejemplo, *IF signal_y=1 THEN*.

Limitaciones

Los datos del tipo *signalxx* no deberán ser definidos en el programa. En el caso en que lo estén, aparecerá visualizado un mensaje de error en cuanto una instrucción o función que se refiere a esta señal, sea ejecutada. Por otro lado, el tipo de dato podrá ser utilizado como parámetro al declarar una rutina.

Datos predefinidos

Siempre se podrá acceder a las señales definidas en los parámetros del sistema desde el programa, utilizando las variables de las señales predefinidas (datos instalados). Obsérvese, sin embargo, que si se definen otros datos con el mismo nombre, estas señales no podrán ser utilizadas.

Características

Signalxo es un tipo de dato *sin valor*. Por lo tanto, los datos de este tipo no permiten la realización de operaciones orientadas a valores.

Signalxi es un tipo de dato *semi valor*.

Información relacionada

Descripción en:

Resumen de instrucciones de entradas/salidas	Resumen RAPID- <i>Señales de Entrada y Salida</i>
Funciones de las entradas/salidas en general	Principios de Movimiento y E/S- <i>Principios de E/S</i>
Configuración de las E/S	Guía del Usuario - <i>Parámetros del Sistema</i>
Características de tipos de datos sin valor	Características Básicas - <i>Tipos de datos</i>

speeddata

Datos de velocidad

Speeddata sirve para especificar la velocidad a la que se moverán tanto los ejes externos como el robot.

Descripción

Los datos de velocidad definen la velocidad:

- a la que se mueve el punto central de la herramienta,
- de la reorientación de la herramienta,
- a la que se mueven los ejes externos rotativos o lineales.

Cuando se combinan diferentes tipos de movimiento, a menudo hay una de las velocidades que limita todos los movimientos. La velocidad de los demás movimientos será reducida de tal manera que todos los movimientos acabarán su ejecución al mismo tiempo.

La velocidad será también restringida por la capacidad del robot. Esto difiere según el tipo de robot y la trayectoria utilizada.

Componentes

v_tcp (*velocidad del tcp*)

Tipo de dato: *num*

La velocidad del punto central de la herramienta (TCP) en mm/s.

Si se utilizan ejes externos coordinados o una herramienta estacionaria, la velocidad será especificada respecto al objeto de trabajo.

v_ori (*velocidad de orientación*)

Tipo de dato: *num*

La velocidad de reorientación del TCP expresado en grados/s.

Si se utilizan ejes externos coordinados o una herramienta estacionaria, la velocidad será especificada respecto al objeto de trabajo.

v_leax (*velocidad de los ejes externos lineales*)

Tipo de dato: *num*

La velocidad de los ejes externos lineales en mm/s.

v_reax (*velocidad de los ejes externos rotativos*)

Tipo de dato: *num*

La velocidad de los ejes externos rotativos en grados/s.

Ejemplo

`VAR speeddata vmedia := [1000, 30, 200, 15];`

El dato de velocidad *vmedia* será definido con las siguientes velocidades:

- 1000 mm/s para el TCP.
- 30 grados/s para la reorientación de la herramienta.
- 200 mm/s para los ejes externos lineales.
- 15 grados/s para los ejes externos rotativos.

`vmedia.v_tcp := 900;`

La velocidad del TCP será cambiada a 900 mm/s.

Datos predefinidos

Una serie de datos de velocidad están ya definidos en el módulo del sistema *BASE*.

<u>Nombre</u>	<u>Vel. TCP</u>	<u>Orientación</u>	<u>Ejes ext. lineal</u>	<u>Ejes ext. rotat.</u>
v5	5 mm/s	500°/s	5000 mm/s	1000°/s
v10	10 mm/s	500°/s	5000 mm/s	1000°/s
v20	20 mm/s	500°/s	5000 mm/s	1000°/s
v30	30 mm/s	500°/s	5000 mm/s	1000°/s
v40	40 mm/s	500°/s	5000 mm/s	1000°/s
v50	50 mm/s	500°/s	5000 mm/s	1000°/s
v60	60 mm/s	500°/s	5000 mm/s	1000°/s
v80	80 mm/s	500°/s	5000 mm/s	1000°/s
v100	100 mm/s	500°/s	5000 mm/s	1000°/s
v150	150 mm/s	500°/s	5000 mm/s	1000°/s
v200	200 mm/s	500°/s	5000 mm/s	1000°/s
v300	300 mm/s	500°/s	5000 mm/s	1000°/s
v400	400 mm/s	500°/s	5000 mm/s	1000°/s
v500	500 mm/s	500°/s	5000 mm/s	1000°/s
v600	600 mm/s	500°/s	5000 mm/s	1000°/s
v800	800 mm/s	500°/s	5000 mm/s	1000°/s
v1000	1000 mm/s	500°/s	5000 mm/s	1000°/s
v1500	1500 mm/s	500°/s	5000 mm/s	1000°/s
v2000	2000 mm/s	500°/s	5000 mm/s	1000°/s
v2500	2500 mm/s	500°/s	5000 mm/s	1000°/s
v3000	3000 mm/s	500°/s	5000 mm/s	1000°/s
vmax	5000 mm/s	500°/s	5000 mm/s	1000°/s

Estructura

```
< dato del tipo de speeddata >
  < v_tcp de num >
  < v_ori de num >
  < v_leax de num >
  < v_reax de num >
```

Información relacionada

Instrucciones de posicionamiento
Movimiento/Velocidad en general

Definición de la velocidad máxima
Configuración de los ejes externos

Capacidad de movimiento

Descripción:

Resumen RAPID - *Movimiento*
Principios de Movimiento y de E/S -
Posicionamiento durante la Ejecución del Programa
Instrucciones - *VelSet*
Guía del Usuario - *Parámetros del Sistema*
Especificación del Producto

string**Cadenas de Carácteres**

String se usa para las cadenas de caracteres.

Descripción

Una cadena de caracteres está formada por una serie de caracteres (como máximo 80) incluidos entre comillas ("),

ejemplo: "Esto es una cadena de caracteres".

En el caso en que se desee incluir comillas dentro de la cadena de caracteres, deberán repetirse dos veces,

ejemplo: "Esta cadena contiene un ""caracter".

Ejemplo

```
VAR string text;  
.  
text := "arranque boquilla1";  
TPWrite text;
```

El texto "*arranque boquilla1*" será introducido en la unidad de programación.

Limitaciones

Una cadena puede tener de 0 a 80 caracteres, incluyendo las comillas adicionales.

Una cadena puede contener cualquiera de los caracteres especificados en la norma ISO 8859-1 así como los caracteres de control (caracteres no contenidos en ISO 8859-1 con un código numérico entre 0-255).

Datos Predefinidos

Hay una serie de constantes de cadenas predefinidas (datos instalados) que pueden utilizarse junto con funciones de cadena.

<u>Nombre</u>	<u>Grupo de caracteres</u>
STR_DIGIT	<caracter> ::= 0 1 2 3 4 5 6 7 8 9
STR_UPPER	<letra mayúscula> ::= A B C D E F G H I J K L M N O P Q R S T U V W X Y Z Á Á Á Á Ä Å Æ Ç È É È È Í Í Î Í 1) Ñ Ò Ó Ô Õ Ö Ø Ù Ú Û Ü 2) 3)
STR_LOWER	<letra minúscula> ::= a b c d e f g h i j k l m n o p q r s t u v w x y z à á â ã ä å æ ç è é ê ë ì í î ï 1) ñ ò ó ô õ ö ø ù ú û ü 2) 3) ß ÿ
STR_WHITE	<en blanco> ::=

- 1) Letra eth islandesa.
- 2) Letra Y con acento agudo
- 3) Letra thorn islandesa.

Información Relacionada

	<u>Describa en:</u>
Operaciones que utilizan cadenas	Características Básicas - <i>Expresiones</i>
Valores de las cadenas	Características Básicas - <i>Elementos de Base</i>

symnum**Número simbólico**

Symnum sirve para representar un número entero con una constante simbólica.

Descripción

Una constante *symnum* está prevista para ser utilizada cuando se comprueba el valor de retorno de las funciones *OpMode* y *RunMode*. Véase el ejemplo siguiente.

Ejemplo

```
IF RunMode() = RUN_CONT_CYCLE THEN
    .
    .
    ELSE
    .
    .
ENDIF
```

Datos Predefinidos

A continuación se indican las constantes simbólicas del tipo de dato *symnum* que están predefinidas y que pueden ser utilizadas cuando se comprueba los valores de retorno de las funciones *OpMode* y *RunMode*.

Valor	Constante simbólica	Comentario
0	RUN_UNDEF	Modo de ejecución no definido
1	RUN_CONT_CYCLE	Modo de ejecución continuo o cíclico
2	RUN_INSTR_FWD	Modo de ejecución hacia adelante
3	RUN_INSTR_BWD	Modo de ejecución hacia atrás
4	RUN_SIM	Modo de ejecución simulada

Valor	Constante simbólica	Comentario
0	OP_UNDEF	Modo de funcionamiento no definido
1	OP_AUTO	Modo de funcionamiento automático
2	OP_MAN_PROG	Modo de funcionamiento manual max. 250 mm/s
3	OP_MAN_TEST	Modo de funcionamiento manual Velocidad total, 100%

Características

Symnum es un tipo de dato equivalente de *num* y por consiguiente hereda sus características.

Información relacionada

Descripción:

Tipos de datos en general, tipos de datos equivalentes

Características BASIC - *Tipos de Datos*

tooldata**Datos de herramienta**

Tooldata se usa para describir las características de una herramienta, como por ejemplo, una boquilla de soldadura o una pinza.

Si la herramienta está fija en el espacio (si se trata de una herramienta estacionaria), se definirán los datos de herramienta normales correspondientes así como la pinza que sujetla el objeto de trabajo.

Descripción

Los datos de herramienta afectarán a los movimientos del robot en la siguiente medida:

- El punto central de la herramienta (TCP) se refiere a un punto que cumplirá con la trayectoria especificada y con la exigencia de velocidad. En el caso en que se reorienta la herramienta o si se usan ejes externos coordinados, solamente este punto seguirá la trayectoria deseada a la velocidad programada.
- En el caso en que se use una herramienta estacionaria, tanto la velocidad como la trayectoria programadas se referirán al objeto de trabajo.
- La carga de la herramienta sirve para controlar los movimientos del robot de la mejor manera posible. La definición de una carga incorrecta puede provocar oscilaciones, por ejemplo.
- Las posiciones programadas se refieren a la posición del TCP utilizado y a la orientación respecto al sistema de coordenadas de la herramienta. Esto significa que si, por ejemplo, se ha reemplazado una herramienta porque está dañada, se podrá todavía utilizar el programa antiguo si se vuelve a definir el sistema de coordenadas de la herramienta.

Los datos de herramienta también se usan al mover el robot para:

- Definir el TCP que no debe moverse cuando se reorienta la muñeca.
- Definir el sistema de coordenadas de la herramienta con vistas a facilitar el movimiento o rotación de las direcciones de la herramienta.

Componentes**robhold**

(en el robot)

Tipo de datos: *bool*

Define si el robot está sujetando la herramienta o no:

- *TRUE* -> El robot está sujetando la herramienta.
- *FALSE* -> El robot no está sujetando la herramienta, es decir, se trata de una herramienta estacionaria.

tframe (*coordenadas de la herramienta*)Tipo de dato: *pose*

El sistema de coordenadas de la herramienta, es decir:

- La posición del TCP (x, y, z) en mm.
- Las direcciones de la herramienta expresadas bajo la forma de un cuaternio (q1, q2, q3 y q4).

Tanto la posición como la rotación se definen utilizando el sistema de coordenadas de la muñeca (véase la Figura 1.) Si se desea utilizar una herramienta estacionaria, la definición se realizará respecto al sistema de coordenadas del mundo.

Si la dirección de la herramienta no está especificada, el sistema de coordenadas de la herramienta y el sistema de coordenadas de la muñeca coincidirán.

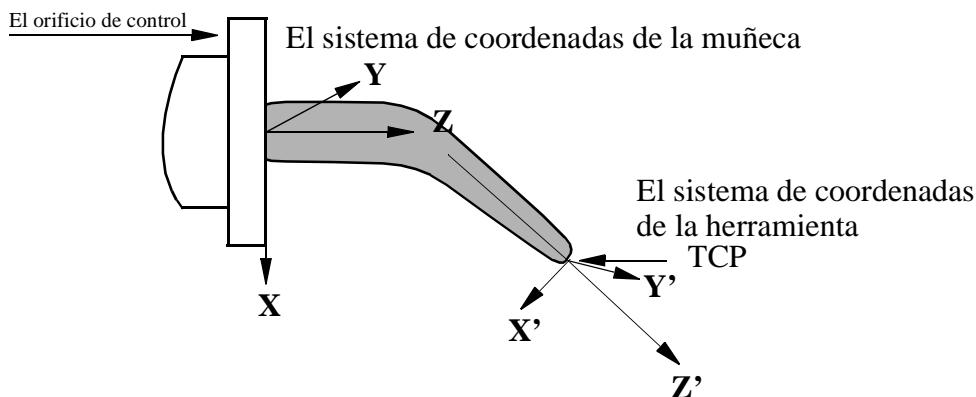


Figura 1 Definición del sistema de coordenadas de la herramienta.

tload (*carga de la herramienta*)Tipo de dato: *loaddata*

La carga de la herramienta, es decir:

- El peso de la herramienta en kg.
- El centro de gravedad de la herramienta (x, y, z) en mm.
- Los ejes de momento de la herramienta, expresados como un cuaternio (q1, q2, q3, q4).
- El momento de inercia de la herramienta de los ejes de momento x, y, z, expresado en kgm^2 .

Si todos los componentes han sido definidos como 0 kgm^2 , la herramienta será considerada como si se tratara de una carga puntual.

Para más información, véase el tipo de dato *loaddata*.

Si se utiliza una herramienta estacionaria, se deberá definir la carga de la pinza que sujetla el objeto de trabajo.

Obsérvese que sólo se deberá especificar la carga de la herramienta. La carga útil manipulada por una pinza se activa/desactiva mediante la instrucción *GripLoad*.

Examples

```
PERS tooldata pinza := [ TRUE, [[97.4, 0, 223.1], [0.924, 0, 0.383 ,0]],  
[5, [23, 0, 75], [1, 0, 0, 0], 0, 0, 0]];
```

La herramienta de la Figura 1 deberá ser descrita utilizando los siguientes valores:

- El robot está sujetando la herramienta.
- El TCP está situado en el punto 223,1 mm en línea recta a partir del eje 6 y 97,4 mm a lo largo del eje X del sistema de coordenadas de la muñeca.
- Las direcciones X y Z de la herramienta deberán girarse de 45° respecto al sistema de coordenadas de la muñeca.
- La herramienta pesa 5 kg.
- El centro de gravedad está situado en el punto 75 mm en línea recta a partir del eje 6 y 23 mm a lo largo del eje X del sistema de coordenadas de la muñeca.
- La carga puede ser considerada como una carga puntual, es decir, sin ningún momento de inercia.

```
pinza.tframe.trans.z := 225.2;
```

El TCP de la herramienta, *gripper*, será ajustado a 225,2 en la dirección z.

Limitaciones

Los datos de herramienta (tooldata) deberán ser definidos únicamente como variables persistentes (*PERS*) y no deberán ser definidos dentro de una rutina. Los valores utilizados serán almacenados luego, cuando el programa sea almacenado en un disquete y se recuperaran al cargar el disquete.

El argumento del tipo datos de herramienta en cualquier instrucción de movimiento deberá ser un dato persistente entero (no un elemento matriz ni un componente de registro).

Datos predefinidos

La herramienta *tool0* define el sistema de coordenadas de la muñeca y tiene como punto de origen el centro de la brida de montaje. Se podrá siempre acceder a *tool0* desde el programa, aunque no puedan ser cambiados (están almacenados en el módulo del sistema *BASE*).

```
PERS tooldata tool0 := [ TRUE, [ [0, 0, 0], [1, 0, 0 ,0] ],  
[0, [0, 0, 0], [1, 0, 0, 0], 0, 0, 0 ]];
```

Estructura

```
< dataobject of tooldata >
  < robold of bool >
  < tframe of pose >
    < trans of pos >
      < x of num >
      < y of num >
      < z of num >
    < rot of orient >
      < q1 of num >
      < q2 of num >
      < q3 of num >
      < q4 of num >
  < tload of loaddata >
    < mass of num >
    < cog of pos >
      < x of num >
      < y of num >
      < z of num >
    < aom of orient >
      < q1 of num >
      < q2 of num >
      < q3 of num >
      < q4 of num >
    < ix of num >
    < iy of num >
    < iz of num >
```

Información Relacionada

Instrucciones de posicionamiento
Sistemas de coordenadas
Definición de la carga útil

Descripción en:
Resumen RAPID- *Movimiento*
Principios de Movimiento y de E/S -
Sistemas de coordenadas
Instrucciones - *Gripload*

tpnum**Número de ventana de la Unidad de Programación**

tpnum sirve para representar el número de ventana de la unidad de programación con una constante simbólica.

Descripción

Una constante *tpnum* está destinada a ser utilizada en la instrucción *TPShow*. Véase el ejemplo que se indica a continuación.

Ejemplo

TPShow TP_PROGRAM;

La *Ventana de Producción* estará activa si el sistema se encuentra en el modo *AUTO* y la *Ventana de Programa* estará activa si el sistema se encuentra en el modo *MAN*, después de la ejecución de esta instrucción.

Datos predefinidos

Las constantes simbólicas que se indican a continuación del tipo de dato *tpnum* están predefinidas y pueden ser utilizadas en la instrucción *TPShow*:

Valor	Constante simbólica	Comentario
1	TP_PROGRAM	AUTO: Ventana de Producción MAN: Ventana de Programa
2	TP_LATEST	Ultima ventana utilizada en la unidad de programación

Características

tpnum es un tipo de dato similar a *num* y por consiguiente tiene sus mismas características.

Información relacionada

Descripción:

Tipos de datos den general,
tipos de datos similares

Características Básicas - *Tipos de Datos*

Comunicación mediante la unidad
de programación

Resumen RAPID - *Comunicación*

Cambiar de ventana en la unidad de
programación

Instrucciones - *TPShow*

triggdata Eventos de posicionamiento - disparo

Triggdata sirve para almacenar datos referentes a un evento de posicionamiento durante un movimiento del robot.

Un evento de posicionamiento puede tomar la forma de la activación de una señal de salida o de la ejecución de una rutina de interrupción en una posición específica a lo largo de la trayectoria de movimiento del robot.

Descripción

Para definir las condiciones de las medidas respectivas en un evento de posicionamiento, se utilizan las variables del tipo *triggdata*. El contenido de los datos de la variable será formado en el programa utilizando una de las instrucciones *TriggIO* o *TriggInt*, y es utilizado por una de las instrucciones *TriggL*, *TriggC* o *TriggJ*.

Ejemplo

```
VAR triggdata gunoff;  
TriggIO gunoff, 5 \DOp:=gun, off;  
TriggL p1, v500, gunoff, fine, gun1;
```

La señal de salida digital *gun* está activada en el valor *off* cuando el TCP está en una posición de 5 mm antes del punto *p1*.

Características

Triggdata es un tipo de dato sin valor.

Información relacionada

Descripción:

Definición de los disparos

Instrucciones - *TriggIO*, *TriggInt*

Uso de los disparos

Instrucciones - *TriggL*, *TriggC*,
TriggJ

Características de tipos de datos sin valor

Características Básicas - *Tipos de Datos*

tunetype**Tipo de ajuste servo**

Tunetype sirve para representar un entero con una constante simbólica.

Descripción

La constante *tunetype* ha sido concebida para ser utilizada como un argumento para la instrucción *TuneServo*. Véase el ejemplo siguiente.

Ejemplo

TuneServo MHA160R1, 1, 110 \Type:= TUNE_KP;

Datos predefinidos

Las siguientes constantes simbólicas del tipo de dato *tunetype* son predefinidas y se utilizarán como argumentos para la instrucción *TuneServo*.

Valor	Constante simbólica	Comentarios
0	TUNE_DF	Reduce las oscilaciones
1	TUNE_KP	Afecta la ganancia de control de posición
2	TUNE_KV	Afecta la ganancia de control de velocidad
3	TUNE_TI	Afecta el tiempo de integración de control de la velocidad

Las siguientes constantes simbólicas del tipo de dato *tunetype* están predefinidas y pueden usarse como argumentos para la instrucción *SpeedPrioAct* (disponible únicamente bajo demanda).

Valor	Constante simbólica	Comment
1	SP_MODE1	Prioridad de velocidad modo de interpolación 1
2	SP_MODE2	Prioridad de velocidad modo de interpolación 2

Características

Tunetype es un tipo de dato equivalente a *num* y por consiguiente hereda sus mismas características.

Información relacionada

Descripción:

Tipos de datos en general, tipos de datos equivalentes

Características Básicas - *Tipos de Datos*

wobjdata**Datos del objeto de trabajo**

Wobjdata se usa para describir el objeto de trabajo que el robot está soldando, procesando, moviendo, etc.

Descripción

Si los objetos de trabajo han sido definidos en una instrucción de posicionamiento, la posición estará basada en las coordenadas del objeto de trabajo. Las ventajas de ello son las siguientes:

- Si los datos de posición son introducidos de forma manual, como en la programación off-line, los valores podrán sacarse de un gráfico.
- Los programas podrán ser reutilizados rápidamente siguiendo los cambios realizados en la instalación del robot. Así, por ejemplo, si se mueve una fijación, sólo se deberá volver a definir el sistema de coordenadas del usuario.
- Las variaciones respecto a la forma en que el objeto de trabajo está fijado podrán ser compensadas. Para ello, sin embargo, se necesitará algún tipo de sensor para posicionar el objeto de trabajo.

Si se utiliza una herramienta estacionaria o ejes externos coordinados, se deberá definir el objeto de trabajo, ya que la trayectoria y la velocidad se referirán al objeto de trabajo y no al TCP.

Los datos del objeto de trabajo podrán ser utilizados también para el movimiento:

- El robot podrá ser movido en las direcciones del objeto de trabajo.
- La posición utilizada que se visualiza está basada en el sistema de coordenadas del objeto de trabajo.

Componentes

robhold	<i>(en el robot)</i>	Tipo de dato: <i>bool</i>
----------------	----------------------	---------------------------

Define si el robot está sujetando o no el objeto de trabajo:

- *TRUE* -> El robot está sujetando el objeto de trabajo, es decir, que está utilizando una herramienta estacionaria.
- *FALSE* -> El robot no está sujetando el objeto de trabajo, es decir, que el robot está sujetando la herramienta.

ufprog (*sistema de coordenadas del usuario/programada*) Tipo de dato: *bool*

Define si se usa o no un sistema de coordenadas fijo del usuario:

- *TRUE* -> Sistema de coordenadas fijo del usuario.
- *FALSE* -> Sistema de coordenadas móvil del usuario, es decir, que se están usando ejes externos coordinados.

ufmec (*sistema de coordenadas de la unidad mecánica*) Tipo de dato: *string*

La unidad mecánica con la que los movimientos del robot están coordinados. Sólo se especifica en el caso de sistemas de coordenadas móvil del usuario (*ufprog* es *FALSE*).

Está especificado con el nombre con el que está definido en los parámetros del sistema, por ejemplo, "orbit_a".

uframe (*sistema de coordenadas del usuario*) Tipo de dato: *pose*

El sistema de coordenadas del usuario, es decir, la posición de la superficie de trabajo o el utilaje utilizados (véase la Figura 1):

- La posición del origen del sistema de coordenadas (x, y, z) en mm.
- La rotación del sistema de coordenadas, expresado como un cuaternio (q1, q2, q3, q4).

Si el robot está sujetando la herramienta, el sistema de coordenadas del usuario será definido como sistema de coordenadas globales (como sistema de coordenadas de la muñeca si se usa una herramienta estacionaria).

Cuando se usan los ejes externos coordinados (*ufprog* es *FALSE*), el sistema de coordenadas del usuario será definido en los parámetros del sistema.

oframe (*sistema de coordenadas del objeto*) Tipo de dato: *pose*

El sistema de coordenadas del objeto, es decir, la posición del objeto de trabajo utilizado (véase la Figura 1):

- La posición del origen del sistema de coordenadas (x, y, z) en mm.
- La rotación del sistema de coordenadas, expresado como un cuaternio (q1, q2, q3, q4).

El sistema de coordenadas del objeto está definido en el sistema de coordenadas del usuario.

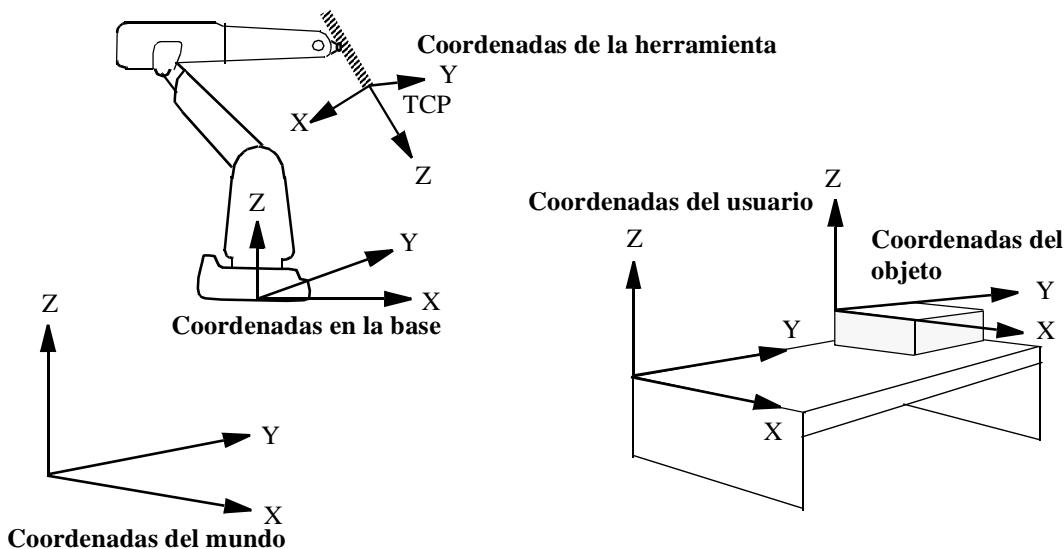


Figura 1 Los diferentes sistemas de coordenadas del robot (cuando el robot está sujetando la herramienta).

Ejemplo

```
PERS wobjdata obj2 :=[ FALSE, TRUE, "", [ [300, 600, 200], [1, 0, 0 ,0] ],
[ [0, 200, 30], [1, 0, 0 ,0] ]];
```

El objeto de trabajo de la Figura 1 se describe utilizando los siguientes valores:

- El robot no está sujetando el objeto de trabajo.
- Se usa el sistema de coordenadas fijo del usuario.
- El sistema de coordenadas del usuario no ha sido girado y las coordenadas de su origen están en x= 300, y = 600, z = 200 mm en el sistema de coordenadas del mundo.
- El sistema de coordenadas del objeto no ha sido girado y las coordenadas de su origen están en x= 0, y= 200, z= 30 mm en el sistema de coordenadas del usuario.

```
obj2.oframe.trans.z := 38.3;
```

- La posición del objeto de trabajo *obj2* está ajustada en 38,3 mm en la dirección z.

Limitaciones

Los datos del objeto de trabajo deberán definirse como una variable persistente (*PERS*) y no deberán definirse dentro de una rutina. Los valores utilizados serán entonces guardados cuando el programa sea almacenado en un disquete y serán reutilizados al cargar de nuevo el contenido del disquete.

El argumento del tipo datos de objeto de trabajo en cualquier instrucción de movimiento deberá ser un dato persistente entero (no un elemento matriz ni un componente de registro).

Datos predefinidos

Los datos del objeto de trabajo *wobj0* deberán ser definidos de forma que el sistema de coordenadas del objeto coincida con el sistema de coordenadas mundo. El robot no sujeta el objeto de trabajo.

Siempre se podrá acceder a *Wobj0* desde el programa, pero nunca podrá ser cambiado (se encuentra almacenado en el módulo del sistema *BASE*).

```
PERS wobjdata wobj0 := [ FALSE, TRUE, "", [ [0, 0, 0], [1, 0, 0, 0] ],  
[ [0, 0, 0], [1, 0, 0, 0] ]];
```

Estructura

```
<dataobject of wobjdata >  
  <robhold of bool>  
  <ufsprog of bool>  
  <ufmec of string>  
  <uframe of pose>  
    <trans of pos>  
      <x of num>  
      <y of num>  
      <z of num>  
    <rot of orient>  
      <q1 of num>  
      <q2 of num>  
      <q3 of num>  
      <q4 of num>  
  <oframe of pose>  
    <trans of pos>  
      <x of num>  
      <y of num>  
      <z of num>  
    <rot of orient>  
      <q1 of num>  
      <q2 of num>  
      <q3 of num>  
      <q4 of num>
```

Información Relacionada

Descripción en:

Instrucciones de posicionamiento

Resumen RAPID - *Movimiento*

Sistemas de coordenadas

Principios de Movimiento y de E/S -
Sistemas de coordenadas

Ejes externos coordinados

Principios de Movimiento y de E/S -
Sistemas de coordenadas

Calibración de los ejes externos coordinados

Guía del Usuario - *Parámetros del Sistema*

wzstationary **Datos de zona mundo estacionaria**

wzstationary (*world zone stationary*) sirve para identificar una zona mundo estacionaria y sólo podrá usarse en una rutina de evento conectada al evento POWER ON.

Una zona mundo es supervisada durante los movimientos del robot tanto durante la ejecución del programa como durante el movimiento manual. Si el TCP del robot entra en esta zona mundo, el movimiento es detenido o se producirá la activación o la reinitialización de una señal de salida digital.

Descripción

Una zona mundo *wzstationary* es definida y activada por la instrucción *WZLimSup* o *WZDOSet*.

WZLimSup o *WZDOSet* proporciona a la variable o persistente de *wzstationary* un valor numérico que identifica la zona mundo.

Una zona estacionaria está siempre activa y sólo se borrará en un arranque en caliente (desactivar y luego activar la alimentación del sistema o cambiar los parámetros del sistema). No se podrá desactivar, activar o borrar una zona mundo estacionaria mediante las instrucciones RAPID.

Las zonas mundo estacionarias deben estar activas a partir de la activación de la alimentación y deben definirse en una rutina de evento POWER ON o en una tarea semiestática.

Ejemplo

```
VAR wzstationary conveyor;  
...  
PROC POWER_ON()  
    VAR shapedata volume;  
    ...  
    WZBoxDef \Inside, volume, p_corner1, p_corner2;  
    WZLimSup \Stat, conveyor, volume;  
ENDPROC
```

Un *transportador* es definido como un volumen en forma de caja rectangular (el volumen debajo de la correa). Si el robot alcanza este volumen, el movimiento queda detenido.

Limitaciones

Un dato *wzstationary* sólo podrá ser definido como una variable global (VAR) (no local dentro de un módulo o rutina) o como un dato persistente (PERS).

Los argumentos del tipo *wzstationary* sólo podrán ser datos enteros (no elementos matriciales o componentes de registro).

Los valores init para datos del tipo *wzstationary* no son utilizados por el sistema. Cuando se usa una variable persistente en un sistema de multitarea, se deberá activar el valor init en 0, por ejemplo, PERS wzstationary share_workarea: = [0];

Características

wzstationary es un tipo de dato similar a *wztemporary* y por consiguiente se beneficia de las misma características.

Información relacionada

Descripción:

Zona Mundo temporal

Tipos de datos - *wztemporary*

Activación de la supervisión límite de la zona mundo

Instrucciones - *WZLimSup*

Activación de una zona mundo para activar una salida digital

Instrucciones - *WZDOSet*

wztemporary**Datos de zona mundo temporal**

wztemporary (*world zone temporary*) sirve para identificar una zona mundo temporal y puede usarse en cualquier parte dentro del programa RAPID para la tarea MAIN.

Una zona mundo es supervisada durante los movimientos del robot tanto durante la ejecución del programa como durante el movimiento manual. Si el TCP del robot entra en esta zona mundo, el movimiento es detenido o se producirá la activación o la reinitialización de una señal de salida digital.

Descripción

Una zona mundo *wztemporary* es definida y activada por la instrucción *WZLimSup* o *WZDOSet*.

WZLimSup o *WZDOSet* proporciona a la variable o persistente de *wztemporary* un valor numérico que identifica la zona mundo.

Una vez definida y activada, una zona mundo temporal podrá ser desactivada mediante la instrucción *WZDisable*, activada de nuevo mediante la instrucción *WZEnable* y borrada con la instrucción *WZFree*.

Todas las zonas mundo temporales del sistema serán automáticamente borradas (borradas en el sistema y todos los datos del tipo *wztemporary* de la tarea MAIN están puestos a 0):

- cuando se cargue un programa nuevo en la tarea MAIN
- cuando se inicie la ejecución del programa desde el principio en la tarea MAIN

Ejemplo

```
VAR wztemporary roll;  
...  
PROC ...  
    VAR shapedata volume;  
    ...  
    WZCylDef \Inside, volume, p_center, 400, 1000;  
    WZLimSup \Temp, roll, volume;  
ENDPROC
```

Un *rodillo* (que la aplicación acaba de introducir en el área de trabajo) es definido como un volumen en forma de cilindro. Si el robot alcanza este volumen, el movimiento queda detenido.

Limitaciones

Un dato *wztemporary* sólo podrá ser definido como una variable global (VAR) (no local dentro del módulo o rutina) o como un dato persistente (PERS).

Los argumentos del tipo *wztemporary* deberán ser únicamente datos enteros (no elementos matriciales o componentes de registro).

Las zonas mundo temporales (instrucciones *WZLimSup* o *WZDOSet*) no deberán ser definidas en tareas otras que MAIN, dado que este tipo de definición se verá afectada por la ejecución del programa en la tarea MAIN.

Los valores init para datos del tipo *wzstationary* no son utilizados por el sistema. Cuando se usa una variable persistente en un sistema de multitarea, se deberá activar el valor init en 0, por ejemplo, PERS wztemporary share_workarea: = [0];

Estructura

<dataobject de *wztemporary*>
<wz de num>

Información relacionada

	<u>Descripción:</u>
Zona mundo estacionaria	Tipos de datos - <i>wzstationary</i>
Activación de la supervisión límite de la zona mundo	Instrucciones - <i>WZLimSup</i>
Activación de una zona mundo para activar una salida digital	Instrucciones - <i>WZDOSet</i>
Desactivación de una zona mundo	Instrucciones - <i>WZDisable</i>
Activación de una zona mundo	Instrucciones - <i>WZEnable</i>
Borrado de una zona mundo	Instrucciones - <i>WZFree</i>

zonedata**Datos de zona**

Zonedata se usa para especificar como debe terminarse una posición, es decir, a qué distancia de la posición programada deben encontrarse los ejes antes de moverse hacia la posición siguiente.

Descripción

Una posición puede terminarse en un punto de paro o un punto de paso.

Un punto de paro significa que tanto el robot como los ejes externos deben alcanzar la posición especificada (parada) antes de que la ejecución del programa prosiga con la instrucción siguiente.

Un punto de paso significa que la posición programada nunca es alcanzada. En lugar de ello, la dirección del movimiento es cambiada antes de que se pueda alcanzar dicha posición. Para cada posición, se podrá definir dos zonas distintas (áreas):

- La zona de la trayectoria del TCP.
- La zona extendida de reorientación de la herramienta y de los ejes externos.

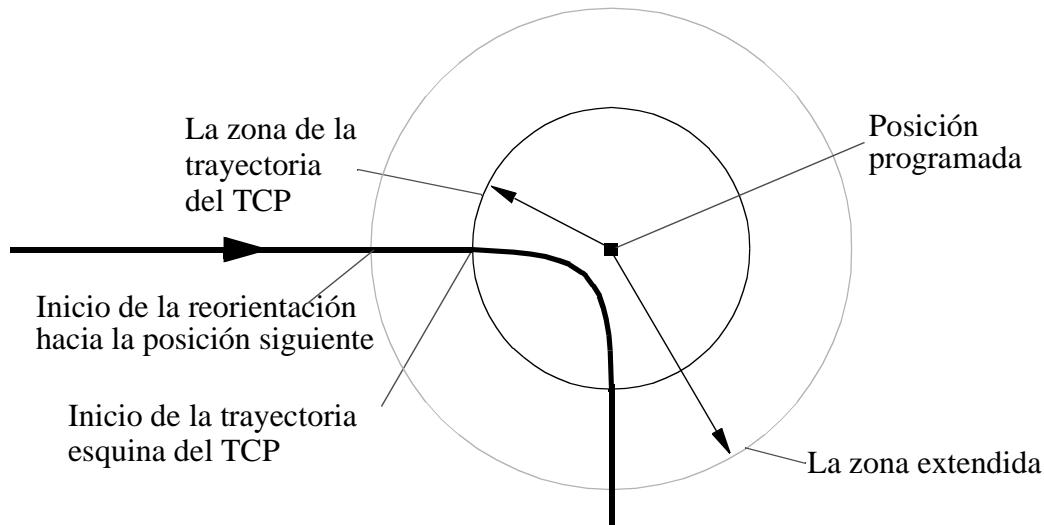


Figura 1 Las zonas de un punto de paso.

Las zonas funcionan de la misma manera durante el movimiento de los ejes, sin embargo, el tamaño de la zona puede diferir de algo respecto a la programada.

El tamaño de la zona nunca podrá ser mayor que la mitad de la distancia a la posición más cercana (hacia adelante o hacia atrás). Si una zona mayor es especificada, el robot automáticamente la reducirá.

La zona de la trayectoria del TCP

Una trayectoria esquina (parabólica) es generada tan pronto como se alcanza el borde de la zona (véase la Figura 1).

La zona de reorientación de la herramienta

La reorientación se inicia en cuanto el TCP alcanza la zona extendida. La herramienta es reorientada de forma que la orientación sea la misma al abandonar la zona que si hubiera habido un punto de paro programado. La reorientación será más suave si se aumenta el tamaño de la zona y habrá menos riesgo de tener que reducir la velocidad para llevar a cabo la reorientación.

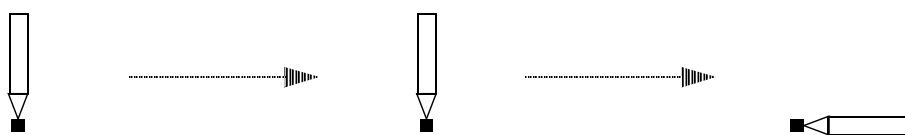


Figura 2 Tres posiciones han sido programadas; la última con una orientación de herramienta diferente.

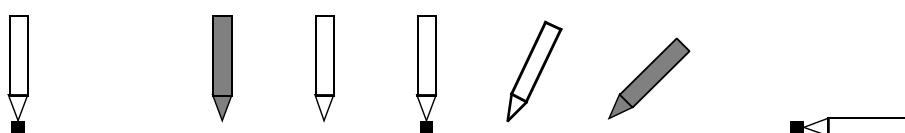


Figura 3 Si todas las posiciones fueran puntos de paro, la ejecución del programa tendría este aspecto.

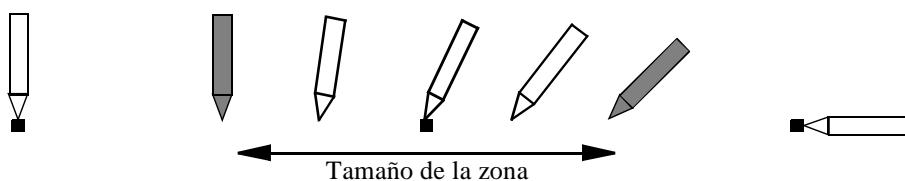


Figura 4 Si la posición del medio fuera un punto de paso, la ejecución del programa tendría este aspecto

La zona de los ejes externos

Los ejes externos empiezan a moverse hacia la siguiente posición tan pronto como el TCP alcanza la zona extendida. De esta forma, un eje lento puede iniciar una aceleración en una etapa anterior y luego seguir ejecutando el programa de forma más regular.

Zona reducida

Con mayores reorientaciones de la herramienta o con movimientos más amplios de los ejes externos, la zona extendida e incluso la zona del TCP podrán ser reducidas por el robot. La zona será definida como el tamaño relativo más pequeño de la zona basándose en los componentes de zona (véase la página siguiente) y

en el movimiento programado.

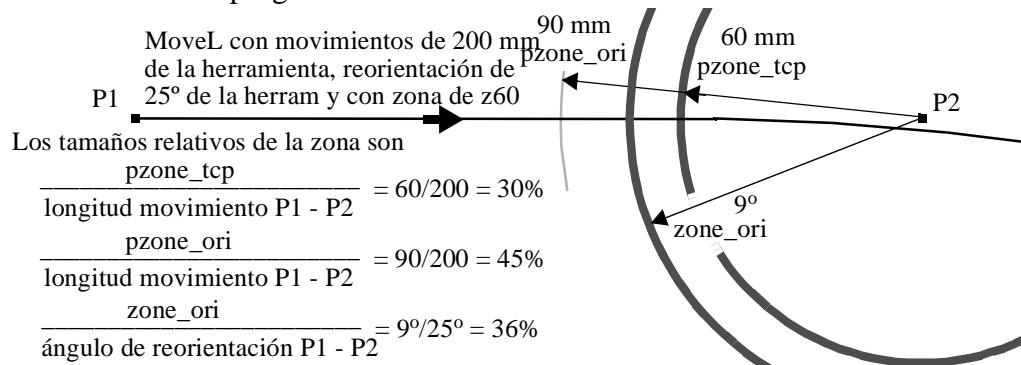


Figura 5 Ejemplo de zona reducida al 36% del movimiento

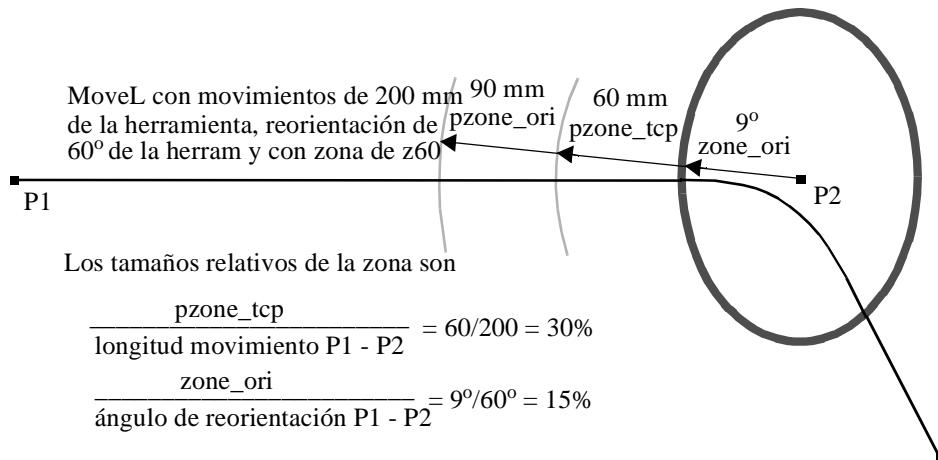


Figura 6 Ejemplo de zona reducida al 15% del movimiento

Componentes

finep

(punto fino)

Tipo de dato: *bool*

Define si el movimiento debe terminar como un punto de paro (punto fino) o como un punto de paso.

- *TRUE* -> El movimiento termina como un punto de paro.
Los componentes restantes en los datos de zona no son utilizados.
- *FALSE* -> El movimiento termina como un punto de paso.

pzone_tcp

(zona de la trayectoria TCP)

Tipo de dato: *num*

El tamaño (el radio) de la zona del TCP en mm.

La zona extendida será definida como el tamaño relativo más pequeño de la zona basado en los siguientes componentes y en el movimiento programado.

pzone_ori (zona de la trayectoria de la orientación)

Tipo de dato: *num*

El tamaño de la zona (el radio) de la reorientación de la herramienta. El tamaño

se definirá como siendo la distancia del TCP a partir del punto programado, en mm.

El tamaño deberá ser mayor que el valor correspondiente de la *pzone_tcp*. Si se especifica un valor más bajo, el tamaño de la zona aumentará automáticamente para aumentar de la misma forma los valores de *pzone_tcp*.

pzone_eax (*zona de la trayectoria de los ejes externos*) Tipo de dato: *num*

El tamaño de la zona (el radio) de los ejes externos. El tamaño se definirá como siendo la distancia del TCP a partir del punto programado, en mm.

El tamaño deberá ser mayor que el valor correspondiente de la *pzone_tcp*. Si se especifica un valor más bajo, el tamaño de la zona aumentará automáticamente para aumentar de la misma forma los valores de *pzone_tcp*.

zone_ori (zona de la orientación) Tipo de dato: num

El tamaño de la zona de la reorientación de la herramienta en grados. Si el robot está sujetando el objeto de trabajo, esto representará un ángulo de rotación del objeto de trabajo.

zone_leax (*zona de los ejes externos lineales*) Tipo de dato: *num*

El tamaño de la zona de los ejes externos lineales en mm.

zone_reax (*zona de los ejes externos rotativos*) Tipo de dato: *num*

El tamaño de zona de los ejes externos rotativos en grados.

Ejemplos

```
VAR zonedata trayec := [ FALSE, 25, 40, 40, 10, 35, 5];
```

El dato de zona *trayec* será definido mediante las siguientes características:

- El tamaño de zona de la trayectoria del TCP es de 25 mm.
 - El tamaño de zona de la reorientación de la herramienta es de 40 mm (movimiento del TCP).
 - El tamaño de zona de los ejes externos es de 40 mm (movimiento del TCP).

Sin embargo, si el TCP está parado, o si hay una reorientación más amplia, o si el movimiento de los ejes externos es mayor respecto a la zona, se aplicarán los siguientes valores:

- El tamaño de la zona de la reorientación de la herramienta es de 10 grados.
 - El tamaño de la zona de los ejes externos lineales es de 35 mm.
 - El tamaño de la zona de los ejes externos rotativos es de 5 grados.

```
trayec.pzone_tcp := 40;
```

El tamaño de la zona de la trayectoria del TCP está ajustada a los 40 mm.

Datos predefinidos

Hay una serie de datos de zona que ya están definidos en el módulo del sistema *BASE*.

Puntos de paro

Nombre

fine	0 mm
-------------	------

Puntos de paso

<u>Nombre</u>	<u>movimiento del TCP</u>			<u>reorientación de la herramienta</u>		
	<u>trayec.TCP</u>	<u>Orientación</u>	<u>Ejes ext.</u>	<u>Orientación</u>	<u>Ejes lin.</u>	<u>Ejes rotat.</u>
z1	1 mm	1 mm	1 mm	0.1 °	1 mm	0.1 °
z5	5 mm	8 mm	8 mm	0.8 °	8 mm	0.8 °
z10	10 mm	15 mm	15 mm	1.5 °	15 mm	1.5 °
z15	15 mm	23 mm	23 mm	2.3 °	23 mm	2.3 °
z20	20 mm	30 mm	30 mm	3.0 °	30 mm	3.0 °
z30	30 mm	45 mm	45 mm	4.5 °	45 mm	4.5 °
z40	40 mm	60 mm	60 mm	6.0 °	60 mm	6.0 °
z50	50 mm	75 mm	75 mm	7.5 °	75 mm	7.5 °
z60	60 mm	90 mm	90 mm	9.0 °	90 mm	9.0 °
z80	80 mm	120 mm	120 mm	12 °	120 mm	12 °
z100	100 mm	150 mm	150 mm	15 °	150 mm	15 °
z150	150 mm	225 mm	225 mm	23 °	225 mm	23 °
z200	200 mm	300 mm	300 mm	30 °	300 mm	30 °

Estructura

```
< data object de zonedata >
  < finep de bool >
  < pzone_tcp de num >
  < pzone_ori de num >
  < pzone_eax de num >
  < zone_ori de num >
  < zone_leax de num >
  < zone_reax de num >
```

Información relacionada

Instrucciones de posicionamiento
Movimientos/Trayectorias en general
Configuración de los ejes externos

Descripción:

Resumen RAPID - *Movimiento*
Principios de Movimiento y de E/S -
Posicionamiento durante la Ejecución del Programa
Guía del Usuario - *Parámetros del Sistema*

Datos del Sistema

Los datos del sistema son los datos internos del robot a los que accede el programa para su lectura. Pueden ser utilizados para leer el estado utilizado, por ejemplo, la velocidad máxima utilizada.

La siguiente tabla contiene una relación de todos los datos del sistema.

Nombre	Descripción	Tipo de dato	Cambiado por	Véase también
C_MOTSET	Características de movimiento utilizadas, es decir: -velocidad máx. y ajuste de la velocidad - aceleración máxima - movimiento relativo a puntos singulares - control de la configuración de los ejes - carga útil en la pinza - resolución de la trayectoria	motsetdata	Instrucciones - VelSet - AccSet - SingArea - ConfL,Conf - GripLoad - PathResol	Tipos de datos- <i>motsetdata</i> Instrucciones - <i>VelSet</i> Instrucciones - <i>AccSet</i> Instrucciones - <i>SingArea</i> Instrucciones - <i>ConfL, ConfJ</i> Instrucciones - <i>GripLoad</i> Instrucciones - <i>PathResol</i>
C_PROGDISP	Desplazamiento del programa utilizado del robot y de los ejes externos.	progdisp	Instrucciones - PDispSet - PDispOn - PDispOff - EOffsSet - EOffsOn - EOffsOff	Tipos de datos- <i>progdisp</i> Instrucciones - <i>PDispSet</i> Instrucciones - <i>PDispOn</i> Instrucciones - <i>PDispOff</i> Instrucciones - <i>EOffsSet</i> Instrucciones- <i>EOffsOn</i> Instrucciones - <i>EOffsOff</i>
ERRNO	El último error ocurrido	errnum	El robot	Tipos de datos- <i>errnum</i> Resumen RAPID - <i>Recuperación de errores</i>
INTNO	La última interrupción ocurrida	intnum	El robot	Tipos de datos- <i>intnum</i> Resumen RAPID - <i>Interrupciones</i>

Datos del Sistema

INDICE

“:=”	Asignación de un valor
AccSet	Reducción de la aceleración
ActUnit	Activación de una unidad mecánica
Add	Adición de un valor numérico
Break	Interrupción de la ejecución del programa
CallByVar	Llamada de un procedimiento mediante una variable
Clear	Borrado de un valor
ClkReset	Puesta a cero de un reloj para el cronometraje
ClkStart	Arranque de un reloj para el cronometraje
ClkStop	Paro de un reloj de cronometraje
Close	Cerrar un archivo o un canal serie
comment	Comentarios
ConfJ	Control de la configuración durante el movimiento eje a eje
ConfL	Control de la configuración durante el movimiento lineal
CONNECT	Conexión de una interrupción a una rutina de tratamiento de interrupciones
CorrClear	Eliminación de todos los generadores de correcciones
CorrCon	Conexión de un generador de correcciones
CorrDiscon	Desconexión de un generador de correcciones
CorrWrite	Escritura en un generador de correcciones
DeactUnit	Desactivación de una unidad mecánica
Decr	Disminución de 1
EOffsOff	Desactivación de un offset de los ejes externos
EOffsOn	Activación de un offset de los ejes externos
EOffsSet	Activación de un offset de ejes externos utilizando un valor
ErrWrite	Escribir un Mensaje de Error
EXIT	Fin de ejecución del programa
ExitCycle	Interrupción del ciclo actual y arranque del siguiente
FOR	Repetición de un número dado de veces
GOTO	Ir a una instrucción nueva identificada por una etiqueta (label)
GripLoad	Definición de la carga útil del robot
IDelete	Anulación de una interrupción
IDisable	Inhabilitación de las interrupciones
IEnable	Habilitación de las interrupciones
Compact IF	Si se cumple una condición, entonces... (una instrucción)
IF	Si se cumple una condición, entonces...; si no...
Incr	Incremento de 1

Instrucciones

IndAMove	Movimiento de una posición absoluta independiente
IndCMove	Movimiento continuo independiente
IndDMove	Movimiento de una posición delta independiente
IndReset	Reinicialización independiente
IndRMove	Movimiento de una posición relativa independiente
InvertDO	Inversión del valor de una señal de salida digital
IODisable	Inhabilitar la unidad de E/S
IOEnable	Habilitar la unidad de E/S
ISignalDI	Orden de interrupción a partir de una señal de entrada digital
ISignalDO	Orden de interrupción a partir de una señal de salida digital
ISleep	Desactivación de una interrupción
ITimer	Ordena una interrupción temporizada
IVarValue	Ordena una interrupción de un valor de variable
IWatch	Habilitación de una interrupción
label	Nombre de línea
Load	Cargar un módulo de programa durante la ejecución
MoveAbsJ	Movimiento del robot a una posición de ejes absoluta
MoveC	Movimiento circular del robot
MoveJ	Movimiento eje a eje del robot
MoveL	Movimiento lineal del robot
Open	Apertura de un archivo o de un canal serie
PathResol	Ajuste de la resolución de trayectoria
PDispOff	Desactivación de un desplazamiento de programa
PDispOn	Activación de un desplazamiento de programa
PDispSet	Activación de un desplazamiento del programa utilizando un valor
ProcCall	Llamada de un procedimiento nuevo
PulseDO	Generación de un pulso en una señal de salida digital
RAISE	Llamada al gestor de error
Reset	Puesta a cero de una señal de salida digital
RestoPath	Restauración de la trayectoria después de una interrupción
RETRY	Rearranque después de un error
RETURN	Fin de ejecución de una rutina
Rewind	Reiniciar la posición del archivo
SearchC	Búsqueda circular utilizando el robot
SearchL	Búsqueda lineal utilizando el robot
Set	Activación de una señal de salida digital

SetAO	Cambio del valor de una señal de salida analógica
SetDO	Cambio del valor de una señal de salida digital
SetGO	Cambio del valor de un grupo de señales de salidas digitales
SingArea	Definición de la interpolación en torno a puntos singulares
SoftAct	Activación del servo suave
SoftDeact	Desactivación del servo suave
StartMove	Rearranque del movimiento del robot
Stop	Paro de la ejecución del programa
StopMove	Paro del movimiento del robot
StorePath	Almacenamiento de una trayectoria cuando se produce una interrupción
TEST	Dependiendo del valor de una expresión...
TPErase	Borrado del texto impreso en la unidad de programación
TPReadFK	Lectura de las teclas de función
TPReadNum	Lectura de un número en la unidad de programación
TPShow	Cambio de ventana de la unidad de programación
TPWrite	Escritura en la unidad de programación
TriggC	Movimiento circular del robot con eventos
TriggEquip	Definición de un evento de E/S de tiempo-posición fijas
TriggInt	Definición de una interrupción relativa a una posición
TriggIO	Definición de un evento de E/S de posición fija
TriggJ	Movimientos de los ejes del robot con eventos
TriggL	Movimientos lineales del robot con eventos
TRYNEXT	Salto de una instrucción que ha causado un error
TuneReset	Reinicialización del ajuste del servo
TuneServo	Ajuste de los servos
UnLoad	Descarga de un módulo de programa durante la ejecución
VelSet	Cambio de la velocidad programada
WaitDI	Espera hasta la activación de una señal de entrada digital
WaitDO	Espera hasta la activación de una señal de salida digital
WaitTime	Espera durante un tiempo especificado
WaitUntil	Esperar hasta el cumplimiento de una condición
WHILE	Repetición de una instrucción mientras...
Write	Escritura en un archivo basado en caracteres o en un canal serie
WriteBin	Escritura en un canal serie binario
WriteStrBin	Escritura de una cadena en un canal serie binario
WZBoxDef	Definición de una zona mundo en forma de caja rectangular

Instrucciones

WZCylDef	Definición de una zona mundo en forma de cilindro
WZDisable	Desactivación de la supervisión de la zona mundo temporal
WZDOSet	Activación de la zona mundo para activar una salida digital
WZEnable	Activación de la supervisión de una zona mundo temporal
WZFree	Borrado de la supervisión de la zona mundo temporal
WZLimSup	Activación de la supervisión del límite de la zona mundo
WZSphDef	Definición de una zona mundo en forma de esfera

“:=”

Asignación de un valor

La instrucción “:=” sirve para asignar un valor a un dato. Este valor puede ser cualquier valor constante contenido en una expresión aritmética, por ejemplo, *reg1+5*reg3*.

Ejemplos

reg1 := 5;

reg1 tiene asignado el valor 5.

reg1 := reg2 - reg3;

reg1 tiene asignado el valor resultante del cálculo *reg2-reg3*.

contador := contador + 1;

contador ha sido incrementado en1.

Argumentos

Dato := Valor

Dato

Tipos de dato: Todos

El dato al que se desea asignar un

Valor

Tipo de dato: Igual que
Dato

El valor deseado.

Ejemplos

herram1.tframe.trans.x := herram1.tframe.trans.x + 20;

El TCP del *herram1* será movido de 20 mm en la dirección X.

palet{5,8} := Abs(valor);

Un elemento de la matriz *pallet* tiene asignado un valor igual al valor absoluto de la variable *valor*.

Limitaciones

El dato (cuyo valor debe ser cambiado) no debe ser ni:

- una constante
- un tipo de dato sin valor.

Los datos y valores deben tener tipos de datos similares (los mismos o equivalentes).

Sintaxis

(EBNF)

```
<assignment target> ':=' <expresión> ';  
<assignment target> ::=  
    <variable>  
    |<persistente>  
    |<DOB>
```

Información Relacionada

Expresiones

Descripción en:

Características Básicas- *Expresiones*

Tipos de datos sin valor

Características Básicas- *Tipos de Datos*

Asignación de un valor inicial a un dato

Características Básicas - *Datos*

Asignación manual de un valor a un dato

Guía del Usuario - *Programación y Pruebas*

AccSet**Reducción de la aceleración**

AccSet sirve cuando se manipulan cargas frágiles. Permite aceleraciones y deceleraciones más lentas, para la realización de movimientos más suaves del robot.

Ejemplos

AccSet 50, 100;

La aceleración está limitada al 50% del valor normal.

AccSet 100, 50;

La rampa de aceleración está limitada al 50% del valor normal.

Argumentos**AccSet Acc Rampa****Acc**

Tipo de dato: *num*

La aceleración y deceleración como un porcentaje de los valores normales.

El 100% corresponde a la aceleración máxima. Valor máximo: 100%.

Un valor de entrada < 20% da el 20% de aceleración máxima.

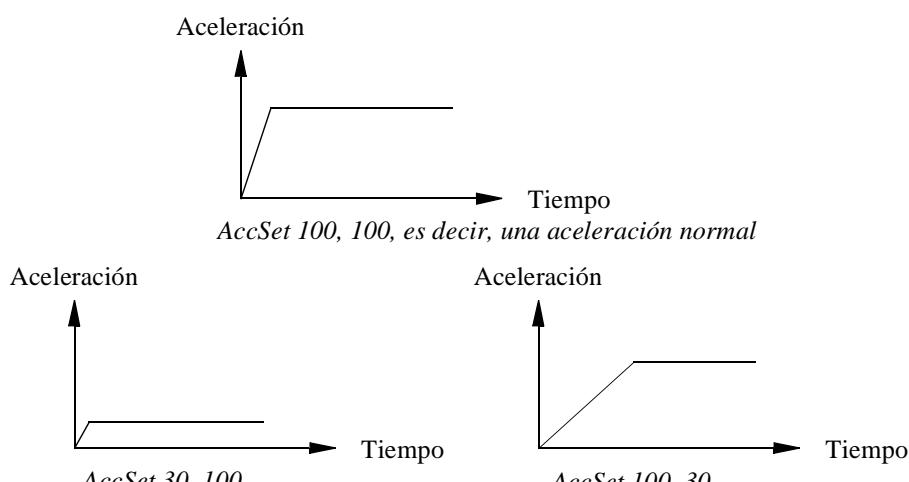
Rampa

Tipo de dato: *num*

El coeficiente con el que la aceleración y deceleración aumenta como porcentaje de los valores normales (véase la Figura 1). Los impulsos del movimiento podrán reducirse disminuyendo este valor.

El 100% corresponde al coeficiente máximo. Valor máximo: 100%.

Un valor de entrada < 10% da el 10% de la aceleración máxima.



Ejecución del Programa

La aceleración se aplica tanto al robot como a los ejes externos hasta que se haya ejecutado una nueva instrucción *AccSet*.

Los valores por defecto (100%) se activan automáticamente

- a la puesta en marcha en frío
- cuando se carga un programa nuevo
- cuando se ejecuta la primera instrucción del programa.

Sintaxis

AccSet
[Acc ':='] < expression (**IN**) of num > ','
[Rampa ':='] < expression (**IN**) of num > ','

Información relacionada

Instrucciones de posicionamiento

Descripción:

Resumen RAPID - *Movimiento*

ActUnit

Activación de una unidad mecánica

ActUnit sirve para activar una unidad mecánica.

Puede utilizarse para determinar la unidad que debe estar activada, por ejemplo, cuando se utilizan unidades de accionamiento comunes.

Ejemplo

ActUnit orbit_a;

Activación de la unidad mecánica *orbit_a*.

Argumentos

ActUnit MecUnit

MecUnit	(<i>Unidad Mecánica</i>)	Tipo de dato: <i>mecunit</i>
---------	----------------------------	------------------------------

El nombre de la unidad mecánica que debe ser activada.

Ejecución del programa

Cuando el robot y los ejes externos están inmovilizados, la unidad mecánica especificada será activada. Ello significa que será controlada y monitorizada por el robot.

En el caso en que varias unidades mecánicas comparten una misma unidad de accionamiento, la activación de una de estas unidades mecánicas conectará también esta unidad a la unidad de accionamiento común.

Limitaciones

La instrucción ActUnit no podrá utilizarse en:

- la secuencia de programa StorePath ... RestoPath
- rutina de evento REARRANQUE

La instrucción de movimiento anterior a esta instrucción deberá terminar con un punto de paro para que en esta instrucción sea posible realizar un rearanque después de un corte de potencia.

Sintaxis

ActUnit
[MecUnit ':='] < variable (**VAR**) of *mecunit* > ';

Información relacionada

Desactivación de las unidades mecánicas
Unidades mecánicas
Más ejemplos

Descripción:

Instrucciones - *DeactUnit*
Tipos de Datos - *mecunit*
Instrucciones - *DeactUnit*

Add**Adición de un valor numérico**

Add sirve para sumar o restar un valor a o de una variable o persistente numérica.

Ejemplos

Add reg1, 3;

3 ha sido añadido a *reg1*, es decir, *reg1:=reg1+3*.

Add reg1, -reg2;

El valor de *reg2* ha sido restado de *reg1*, es decir, *reg1:=reg1-reg2*.

Argumentos**Add Nombre ValorAñadir**

Nombre

Tipo de dato: *num*

El nombre de la variable o persistente que se desea cambiar.

ValorAñadir

Tipo de dato: *num*

El valor que se desea añadir.

Sintaxis

Add

[**Nombre** ':='] < variable o persistente (**INOUT**) de *num* > ','
 [**ValorAñadir** ':='] < expresión (**IN**) de *num* > ';'

Información relacionadaDescripción:

Incrementar una variable de 1

Instrucciones - *Incr*

Disminuir una variable de 1

Instrucciones - *Decr*

Cambiar un dato utilizando una expresión arbitraria, por ejemplo, una multiplicación

Instrucciones - *:=*

Add

Instrucciones

Break Interrupción de la ejecución del programa

Break sirve para realizar una interrupción inmediata en la ejecución del programa para propósitos de detección y eliminación de los códigos de error del programa RAPID.

Ejemplo

```
..  
Break;  
...
```

La ejecución del programa se detiene y entonces es posible realizar un análisis de las variables, valores, etc.. para propósitos de detección y eliminación de errores.

Ejecución del programa

La instrucción detiene la ejecución del programa inmediatamente, sin esperar que el robot y los ejes externos hayan alcanzado su punto de destino programado del movimiento que se está realizando en este momento. La ejecución del programa podrá entonces ser rearrancada a partir de la instrucción siguiente.

En el caso de haber una instrucción *Break* en alguna rutina de evento, la rutina será ejecutada a partir del principio del evento siguiente.

Sintaxis

```
Break';'
```

Información relacionada

Descripción:

- Paro de la actividad del programa
- Paro después de un error fatal
- Terminar la ejecución del programa
- Sólo detener los movimientos del robot

- Instrucciones - Stop*
- Instrucciones - EXIT*
- Instrucciones - EXIT*
- Instrucciones - StopMove*

Break

Instrucciones

CallByVar**Llamada de un procedimiento
mediante una variable**

CallByVar (Call By Variable) sirve para llamar procedimientos con nombres específicos, por ejemplo, *proc_name1*, *proc_name2*, *proc_name3* ... *proc_namex* mediante una variable.

Ejemplo

```
reg1 := 2;
CallByVar "proc", reg1;
```

El procedimiento *proc2* es llamado.

Argumentos**CallByVar Nombre Número****Nombre**Tipo de dato: *string*

Es la primera parte del nombre del procedimiento, por ejemplo, *proc_name*.

NúmeroTipo de dato: *num*

Es el valor numérico del número del procedimiento. Este valor será convertido en una cadena y producirá la segunda parte del nombre del procedimiento, por ejemplo, *1*. El valor deberá ser un número entero positivo.

Ejemplo**Selección estática de una llamada de procedimiento**

```
TEST reg1
CASE 1:
    lf_door door_loc;
CASE 2:
    rf_door door_loc;
CASE 3:
    lr_door door_loc;
CASE 4:
    rr_door door_loc;
DEFAULT:
    EXIT;
ENDTEST
```

Dependiendo de si el valor del registro *reg1* es 1, 2, 3 o 4, diferentes procedi-

mientos serán llamados para realizar el tipo de trabajo apropiado para la puerta seleccionada. La situación de la puerta se encuentra en el argumento *door_loc*.

Selección dinámica de la llamada de procedimiento con una sintaxis en RAPID

```
reg1 := 2;  
%”proc”+NumToStr(reg1,0)% door_loc;
```

El procedimiento *proc2* es llamado con el argumento *door_loc*.

Limitación: Todos los procedimientos deben tener un nombre específico, por ejemplo, *proc1*, *proc2*, *proc3*.

Selección dinámica de la llamada de procedimiento con CallByVar

```
reg1 := 2;  
CallByVar “proc”,reg1;
```

El procedimiento *proc2* es llamado.

Limitación: Todos los procedimientos deben tener un nombre específico, por ejemplo, *proc1*, *proc2*, *proc3*, y no se podrán utilizar argumentos.

Limitaciones

Sólo puede utilizarse para llamar procedimientos sin parámetros.

La ejecución de CallByVar es un poco más larga en tiempo que la ejecución de una llamada normal de procedimiento.

Gestión de errores

En caso de producirse una referencia a un procedimiento desconocido, la variable del sistema ERRNO se activará en ERR_REFUNKPRC.

En caso de producirse un error de llamada a un procedimiento, la variable del sistema ERRNO se activará en ERR_CALLPROC.

Estos errores pueden ser procesados en el gestor de errores.

Sintaxis

```
CallByVar  
[Name ‘:=’] <expresión (IN) de string>,’  
[Number ‘:=’] <expresión (IN) de num>;’
```

Información relacionada

Descripción:

Llamada de procedimientos

Características Básicas - *Rutinas*
Guía del Usuario - *El lenguaje de programación RAPID*

Clear**Borrado de un valor**

Clear sirve para inicializar una variable o persistente numérica, es decir, activarlas en el valor 0.

Ejemplo

Clear reg1;

El valor de *Reg1* queda eliminado, es decir que, *reg1:=0*.

Argumentos**Clear Nombre****Nombre**Tipo de dato: *num*

El nombre de la variable o persistente que se desea eliminar.

Sintaxis**Clear**[Nombre ':='] < variable o persistente (**INOUT**) de *num* > ' ;'

Información relacionadaDescripción:

Incrementar la variable de 1

Instrucciones - *Incr*

Disminuir la variable de 1

Instrucciones - *Decr*

Clear

Instrucciones

ClkReset Puesta a cero de un reloj para el cronometraje

ClkReset sirve para poner a cero un reloj que funciona como un cronómetro.

Esta instrucción puede utilizarse antes de usar un reloj, para asegurarse de que está a 0.

Ejemplo

ClkReset reloj1;

El reloj *reloj1* será puesto a cero.

Argumentos

ClkReset Reloj

Reloj

Tipo de dato: *reloj*

El nombre del reloj que se desea reinicializar.

Ejecución del programa

Cuando se reinicializa un reloj, éste se pondrá a 0.

Si el reloj está funcionando, se parará y luego se reinicializará.

Sintaxis

ClkReset
[Reloj ':='] < variable (**VAR**) de *reloj* > ';

Información Relacionada

Descripción:

Otras instrucciones de reloj

Resumen RAPID - *Sistema y Hora*

ClkReset

Instrucciones

ClkStart

Arranque de un reloj para el cronometraje

ClkStart sirve para arrancar un reloj que funciona como un cronómetro.

Ejemplo

```
ClkStart reloj1;
```

El reloj *reloj1* será arrancado.

Argumentos

ClkStart Reloj

Reloj

Tipo de dato: *reloj*

El nombre del reloj que se desea arrancar.

Ejecución del programa

Cuando se arranca un reloj, seguirá funcionando y contando los segundos hasta que sea detenido.

Cuando la ejecución del programa se para, el reloj sigue funcionando. Sin embargo, el acontecimiento que se deseaba cronometrar puede ya no ser válido. Por ejemplo, si el programa estaba cronometrando el tiempo de espera de una entrada, puede ocurrir que la entrada haya sido recibida mientras el programa estaba detenido. En este caso, el programa no será capaz de “ver” el acontecimiento que ocurrió mientras el programa estaba parado.

El reloj seguirá funcionando cuando se corte el suministro de potencia siempre y cuando el sistema de alimentación de baterías de seguridad mantenga el programa que contiene la variable del reloj.

Cuando el reloj está funcionando, podrá ser leído, parado o reinicializado.

Ejemplo

```
VAR reloj reloj2;
```

```
ClkReset reloj2;  
ClkStart reloj2;  
WaitUntil DInput(di1) = 1;  
ClkStop reloj2;
```

```
time:=ClkRead(reloj2);
```

El tiempo de espera que *di1* tardará para pasar a 1 está siendo cronometrado.

Sintaxis

```
ClkStart  
[ Reloj ':=' ] < variable (VAR) de reloj > ','
```

Información Relacionada

Descripción:

Otras instrucciones de reloj

Resumen RAPID - *Sistema y Hora*

ClkStop Paro de un reloj de cronometraje

ClkStop sirve para parar un reloj que funciona como cronómetro.

Ejemplo

ClkStop reloj1;

El reloj *reloj1* será parado.

Argumentos

ClkStop Reloj

Reloj

Tipo de dato: *reloj*

El nombre del reloj que se desea parar.

Ejecución del programa

Cuando se para un reloj, éste dejará de funcionar.

Si se para un reloj, podrá ser leído, arrancado de nuevo o reinicializado.

Sintaxis

ClkStop
[Reloj ':='] < variable (**VAR**) de *reloj* > ';

Información Relacionada

Descripción:

Otras instrucciones de reloj

Resumen RAPID - *Sistema y Hora*

Más ejemplos

Instrucciones - *ClkStart*

Close**Cerrar un archivo o un canal serie**

Close sirve para cerrar un archivo o un canal serie.

Ejemplo

Close canal2;

El canal serie denominado *canal2* será cerrado.

Argumentos

Close DispositivoE/S

DispositivoE/S

Tipo de dato: *iodev*

El nombre (referencia) del archivo o del canal serie que se desea cerrar.

Ejecución del programa

El archivo o canal serie especificado será cerrado y por lo tanto para realizar una lectura o escritura, deberá volver a abrirse. Si el archivo o canal serie ya está cerrado, el sistema ignorará la instrucción.

Sintaxis

Close
[**DispositivoE/S** ':='] <variable (**VAR**) de *iodev*>' ;'

Información Relacionada

Descripción:

Abrir un archivo o un canal serie

Resumen RAPID - *Comunicación*

Close

Instrucciones

comment**Comentarios**

Comment sirve únicamente para que el programa sea más inteligible. No tiene ningún efecto en la ejecución del programa.

Ejemplo

```
! Ir a la posición situada encima del pallet  
MoveL p100, v500, z20, herramienta1;
```

Se ha introducido un comentario en el programa para hacerlo más inteligible.

Argumentos**! Comentario****Comentario**

Cadena de texto

Cualquier texto.

Ejecución del Programa

La ejecución del programa no se verá en nada afectada por la instrucción de comentario.

Sintaxis

(EBNF)

'!' {<carácter>} <nuevalinea>

Información relacionadaDescripción:

Caracteres permitidos en un comentario

Características Básicas-
Elementos de Base

Comentarios dentro de las declaraciones
de datos y de rutinas

Características Básicas-
Elementos de Base

comment

Instrucciones

Compact IF**Si se cumple una condición, entonces... (una instrucción)**

Compact IF sirve cuando una sola instrucción debe ser ejecutada únicamente si se cumple una condición específica.

En el caso en que diferentes instrucciones deban ser ejecutadas, según se cumpla una condición específica o no, se utilizará la instrucción *IF*.

Ejemplos

IF reg1 > 5 GOTO sig;

Si *reg1* es mayor que 5, entonces la ejecución del programa prosigue a la etiqueta *sig*.

IF contador > 10 Set do1;

La señal *do1* se activará si el *contador* > 10.

Argumentos

IF Condición ...

Condición

Tipo de dato: *bool*

La condición que debe cumplirse para que la instrucción pueda ejecutarse.

Sintaxis

(EBNF)

IF <expresión condicional> (<instrucción> | <SMT>) ;'

Información Relacionada

Descripción:

Condiciones (expresiones lógicas)

Características Básicas - *Expresiones*

IF con varias instrucciones

Instrucciones - *IF*

ConfJ

Control de la configuración durante el movimiento eje a eje

ConfJ (Configuration Joint) sirve para especificar si la configuración del robot debe ser controlada o no durante el movimiento eje a eje. Si la configuración no va a ser controlada, el robot podrá, en algunas ocasiones, utilizar una configuración distinta de la que había sido programada.

Con ConfJ\Off, el robot no podrá cambiar de configuración de los ejes principales; buscará una solución con la misma configuración de ejes principales que la que tiene. Se moverá a la configuración de muñeca más cercana de los ejes 4 y 6.

Ejemplos

ConfJ \Off;
MoveJ *, v1000, fine, herram1;

El robot se mueve a la posición y en la orientación programadas. En el caso en que esta posición pueda ser alcanzada de varias maneras diferentes, con distintas configuraciones de los ejes, el sistema escogerá la posición más cercana posible.

ConfJ \On;
MoveJ *, v1000, fine, herram1;

El robot se mueve a la posición, con la orientación y la configuración de ejes programadas. Si ello no es posible, la ejecución del programa se detiene.

Argumentos

ConfJ [**\On**] | [**\Off**]

\On

Tipo de dato: *switch*

El robot siempre se mueve a la configuración de ejes programada. Si ello no es posible utilizando la posición y la orientación programadas, la ejecución del programa se para.

El robot IRB5400 se moverá a la configuración de ejes programada o a una configuración de ejes parecida a la programada. La ejecución del programa no se detendrá si es imposible alcanzar la configuración de ejes programada.

\Off

Tipo de dato: *switch*

El robot siempre se mueve a la configuración de ejes más cercana.

Ejecución del programa

Si se selecciona el argumento *\On* (o ningún argumento) el robot siempre se mueve a la configuración de ejes programada. Si esto no es posible utilizando la posición y la orientación programadas, la ejecución del programa se detiene antes de que se inicie el movimiento.

Si se selecciona el argumento *\Off*, el robot siempre se moverá a la configuración de ejes más cercana. Esta puede ser distinta de la programada si la configuración ha sido especificada de forma incorrecta manualmente o si se ha realizado un desplazamiento del programa.

El control de la configuración se activará por defecto, es decir que se activa automáticamente:

- a la puesta en marcha
 - cuando se carga un programa nuevo
 - cuando se arranca la ejecución del programa desde el principio.
-

Sintaxis

ConfJ
['\' On] | ['\' Off] ';'

Información relacionada

Descripción:

Manipulación de las distintas configuraciones Principios de Movimiento -
Configuración del Robot

Configuración del robot en el movimiento Instrucciones - *ConfL*
lineal

ConfL

Control de la configuración durante el movimiento lineal

ConfL (Configuration Linear) sirve para especificar si la configuración del robot debe ser controlada o no durante un movimiento circular o lineal. En el caso en que la configuración no sea controlada, la configuración en el momento de la ejecución podrá diferir de la configuración realizada en el momento de la programación. Puede ocurrir también que haya algún cambio de orientación en los movimientos del robot cuando se cambie el modo a un movimiento eje a eje.

NOTA: La monitorización de la configuración no es utilizada en el robot IRB5400.

Ejemplos

```
ConfL \On;  
MoveL *, v1000, fine, herram1;
```

La ejecución del programa se detiene cuando el robot no puede alcanzar la configuración programada desde la posición actual.

```
SingArea \Wrist;  
Confl \On;  
MoveL *, v1000, fine, tool1;
```

El robot se mueve a la posición programada, con la orientación y configuración de los ejes deseada. En el caso en que ello no sea posible, la ejecución del programa se detiene.

```
ConfL \Off;  
MoveL *, v1000, fine, herram1;
```

No aparecerá ningún mensaje de error cuando la configuración programada no corresponda con la configuración realizada en la ejecución del programa.

Argumentos

ConfL [\\On] | [\\Off]

|On

Tipo de dato: *switch*

La configuración del robot será monitorizada.

|Off

Tipo de dato: *switch*

La configuración del robot no será monitorizada.

Ejecución del programa

Durante un movimiento lineal o circular, el robot siempre se mueve a la posición programada y a la orientación que tenga la configuración de ejes más cercana. En el caso en que se seleccione el argumento *\On* (o ningún argumento) entonces la ejecución del programa se detiene en cuanto:

- la configuración de la posición programada no es alcanzada a partir de la posición actual.
- la orientación necesaria de cualquier eje de la muñeca para alcanzar la posición programada a partir de la posición actual, excede un límite (140-180 grados).

No obstante se podrá volver a arrancar el programa, incluso si los ejes de la muñeca continúan hacia una configuración incorrecta. En un punto de paro, el robot realizará una comprobación de que todas las configuraciones de los ejes hayan sido respetadas, no sólo los ejes de la muñeca.

En el caso en que también se utilice SingArea\Wrist, el robot siempre se moverá a la configuración programada de los ejes de la muñeca y las demás configuraciones de los ejes restantes serán comprobadas por el robot en un punto de paro.

En el caso en que se seleccione el argumento *\Off*, no habrá ninguna monitorización de la configuración.

La monitorización está activada por defecto. Se activa automáticamente:

- a la puesta en marcha
- cuando se carga un programa nuevo
- cuando se ejecuta la primera instrucción en el programa

Sintaxis

ConfL
['\' On] | ['\' Off] ;'

Información relacionada

Descripción:

Manipulación de distintas configuraciones

Principios de Movimiento y de E/S -
Configuración del Robot

Configuración del robot durante un
movimiento eje a eje

Instrucciones - *ConfJ*

CONNECT**Conexión de una interrupción a una rutina de tratamiento de interrupciones**

CONNECT sirve para encontrar la identificación de una interrupción y conectarla a una rutina de tratamiento de interrupciones.

La interrupción es definida ordenando un evento de interrupción y especificando su identificación. Así, cuando ocurre el evento, la rutina de tratamiento de interrupciones se ejecuta automáticamente.

Ejemplo

```
VAR intnum aliment_bajo;  
CONNECT aliment_bajo WITH aliment_vacio;  
ISignalDI di1, 1 , aliment_bajo;
```

La identificación de interrupción *aliment_bajo* es creada y es conectada a la rutina de tratamiento de interrupciones *aliment_vacio*. La interrupción ha sido definida como *entrada di1 se está activando*. En otras palabras, cuando esta señal se activa, la rutina de tratamiento de interrupción *aliment_vacio* se ejecuta.

Argumentos**CONNECT Interrupción WITH Rutina tratamiento interrup****Interrupción**Tipo de dato: *intnum*

La variable que debe ser asignada a la identificación de la interrupción.
Esto no deberá ser declarado dentro de una rutina (datos de rutina).

Rutina tratamiento interrup

Identificador

El nombre de la rutina de tratamiento de interrupciones.

Ejecución del programa

La variable tiene asignada una identificación de interrupción que podrá ser utilizada posteriormente al ordenar o inhabilitar las interrupciones. Esta identificación está también conectada a la rutina de tratamiento de interrupciones especificada.

Observar que antes de poder procesar un evento, se deberá ordenar una interrupción, es decir que el evento deberá ser especificado.

Limitaciones

Una interrupción (identificación de interrupción) no podrá estar nunca conectada a más de una rutina de tratamiento de interrupciones. No obstante, diferentes interrupciones podrán ser conectadas a la misma rutina de tratamiento de interrupciones.

Cuando se ha conectado una interrupción a una rutina de tratamiento de interrupciones, no se podrá transferir o volver a conectar dicha interrupción a otra rutina; deberá primero ser borrada mediante la instrucción *IDelete*.

Gestión de errores

En el caso en que la variable de interrupción esté ya conectada a una rutina de tratamiento de interrupciones, la variable del sistema ERRNO se activará en ERR_ALRDYCNT.

En el caso en que la variable de interrupción no sea una referencia de variable, la variable del sistema ERRNO se activará en ERR_CNTNOTVAR.

En el caso en que ya no hayan números de interrupción disponibles, la variable del sistema ERRNO se activará ERR_INOMAX.

Estos errores pueden ser manipulados en el gestor de errores.

Sintaxis

(EBNF)

CONNECT <elemento conexión> **WITH** <trap>;

<elemento conexión> ::= <variable>
| <parámetro>
| <VAR>
<trap> ::= <identificador>

Información relacionada

Descripción:

Resumen de interrupciones

Resumen RAPID - *Interrupciones*

Más información sobre la manipulación de las interrupciones

Características Básicas- *Interrupciones*

CorrClear Eliminación de todos los generadores de correcciones

CorrClear sirve para eliminar todos los generadores de correcciones conectados. Esta instrucción puede utilizarse para eliminar todos los offsets proporcionados anteriormente por todos los generadores de correcciones.

Ejemplo

CorrClear;

La instrucción elimina todos los generadores de correcciones conectados.

Nota: Para asegurarse fácilmente de que todos los generadores de corrección (con correcciones) han sido eliminados al arranque del programa se deberá ejecutar la instrucción *CorrClear* en una rutina de evento de ARRANQUE. Véase Parámetros del Sistema - Tema: *Controlador*.

Sintaxis

CorrClear ‘;’

Información relacionada

	Descripción
Conexión a un generador de correcciones	Instrucciones - <i>CorrCon</i>
Desconexión de un generador de correcciones	Instrucciones - <i>CorrDiscon</i>
Escritura en un generador de correcciones	Instrucciones - <i>CorrWrite</i>
Lectura de todos los offsets actuales	Funciones - <i>CorrRead</i>
Descriptor de correcciones	Tipos de datos - <i>corrdescr</i>

CorrCon**Conexión de un generador
de correcciones**

CorrCon sirve para realizar la conexión a un generador de correcciones.

Ejemplo

```
VAR corrdescr id;  
...  
CorrCon id;
```

La referencia del generador de correcciones corresponde a la reserva *id* de la variable.

Argumentos**CorrCon Descr**

Descr	Tipo de dato: <i>corrdescr</i>
--------------	--------------------------------

Descriptor del generador de correcciones.

Ejemplo**Sistema de coordenadas de la trayectoria**

Todas las correcciones de trayectoria (offsets en la trayectoria) son añadidas al sistema de coordenadas de la trayectoria. El sistema de coordenadas de la trayectoria se define

de la siguiente forma:

P = Sistema de coordenadas de la trayectoria
 T = Sistema de coordenadas de la herramienta

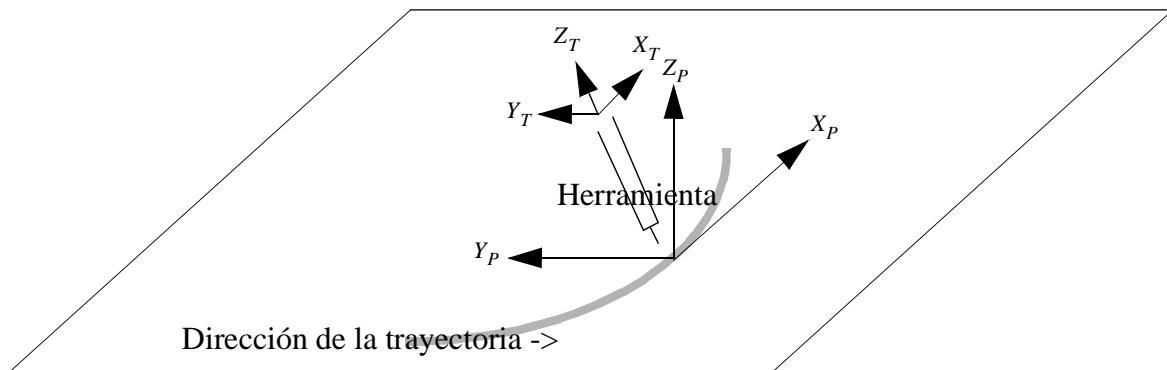


Figura 1 Sistema de coordenadas de la trayectoria.

- El eje de coordenadas X de la trayectoria viene dado por la tangente de la trayectoria.
- El eje de coordenadas Y es normal a la trayectoria y apunta hacia el centro interior de curvatura de la trayectoria.
- El eje de coordenadas Z de la trayectoria se obtiene como el producto vectorial del eje X de coordenadas de la trayectoria y el eje Y de coordenadas de la trayectoria.

Ejemplo de aplicación

Un ejemplo de aplicación en que se utilizan correcciones de trayectoria sería el caso de un robot que sujeta una herramienta montada con dos sensores para detectar la distancia vertical y horizontal de un objeto de trabajo.

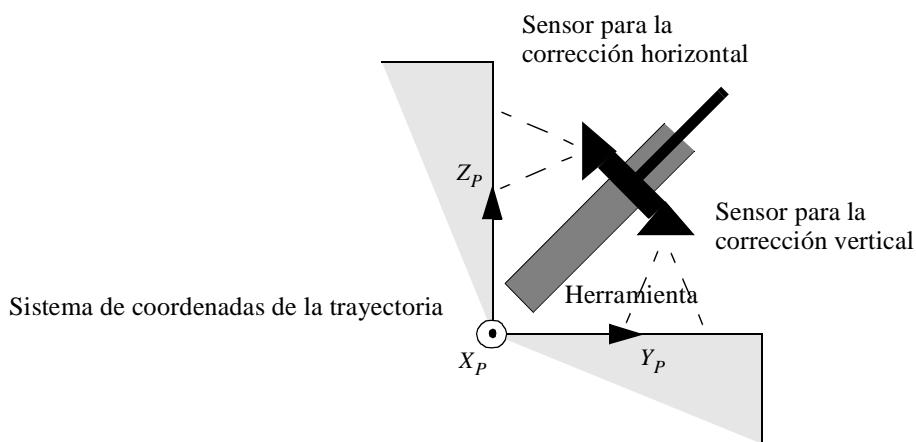


Figura 2 Dispositivo de corrección de la trayectoria.

Ejemplo de programa

```

CONST num TARGET_DIST := 5;
CONST num SCALE_FACTOR := 0.5;
VAR intnum intno1;
VAR corrdesc hori_id;
VAR corrdesc vert_id;
VAR pos total_offset;
VAR signalai hori_sig;
VAR signalai vert_sig;
VAR pos write_offset;

PROC PathRoutine()

    ! Conectar los generadores de corrección para la corrección horizontal y vertical.
    CorrCon hori_id;
    CorrCon vert_id;

    ! Configurar una interrupción temporizada de 5 Hz. La rutina de tratamiento de
    ! interrupciones leerá los valores del sensor y computará las correcciones de
    ! trayectoria.
    CONNECT intno1 WITH ReadSensors;
    ITimer, 0.2, intno1;

    ! Activación de la instrucción MoveL para las correcciones de trayectoria.
    MoveL p10,v100,z10,tool\Corr;
    ! Lectura de todas las correcciones añadidas por todos los generadores de corrección
    ! conectados.
    total_offset := CorrRead();
    ! Escritura de la corrección vertical en la Unidad de Programación.
    TPWrite "La corrección total vertical es: "\Num:=total_offset.z;
    ! Desconectar el generador de correcciones para la corrección vertical.
    ! Las correcciones horizontales no se verán afectadas.
    CorrDiscon vert_id;
    ! Activación de la instrucción MoveL para las correcciones de trayectoria.
    MoveL p20,v100,z10,tool\Corr;

    ! Eliminar todos los generadores de corrección conectados.
    ! En este caso, el único generador conectado es el que realiza la corrección
    ! horizontal.
    CorrClear;

    ! Eliminar la interrupción temporizada.
    IDElete intno1;

ENDPROC

TRAP ReadSensors

    ! Computar los valores de corrección horizontal y ejecutar la corrección.
    write_offset.x := 0;

```

```

write_offset.y := (hori_sig - TARGET_DIST)*SCALE_FACTOR;
write_offset.z := 0;
CorrWrite hori_id, write_offset;

! Computar los valores de corrección vertical y ejecutar la corrección.
write_offset.x := 0;
write_offset.y := 0;
write_offset.z := (vert_sig - TARGET_DIST)*SCALE_FACTOR;
CorrWrite vert_id, write_offset;

ENDTRAP

```

Explicación del programa

Se conectarán dos generadores de correcciones mediante la instrucción *CorrCon*. Cada generador de corrección tiene un único descriptor (*hori_id* y *vert_id*) del tipo *corrdesc*. Cada uno de los dos sensores utilizará un generador de correcciones.

Una interrupción temporizada es activada para llamar la rutina de tratamiento de interrupciones *ReadSensors* con una frecuencia de 5 Hz. Los offsets necesarios para la corrección de la trayectoria son computados en la rutina de tratamiento de interrupciones y escritos en el generador de correcciones correspondiente (cuya referencia viene indicada por los descriptores *hori_id* y *vert_id*) mediante la instrucción *CorrWrite*. Todas las correcciones tendrán un impacto inmediato en la trayectoria.

La instrucción *MoveL* deberá ser programada con el argumento switch *Corr* cuando se usen las correcciones de trayectoria. De lo contrario, no se ejecutarán las correcciones.

Cuando la primera instrucción *MoveL* esté lista, la función *CorrRead* será utilizada para leer la suma de todas las correcciones (la corrección total de trayectoria) proporcionadas por todos los generadores de corrección conectados. El resultado de la corrección total vertical de trayectoria aparecerá indicada en la unidad de programación mediante la instrucción *TPWrite*.

La instrucción *CorrDiscon* desconectará entonces el generador de correcciones para la corrección vertical (cuya referencia viene indicada por el descriptor *vert_id*). Todas las correcciones añadidas por este generador de correcciones serán eliminadas de la corrección total de trayectoria. No obstante, las correcciones añadidas por el generador de correcciones para la corrección horizontal serán guardadas.

Por último, la función *CorrClear* eliminará todos los generadores de correcciones conectados y sus correcciones previamente añadidas. En este caso, sólo se eliminará el generador de correcciones para la corrección horizontal. La interrupción temporizada será también eliminada mediante la instrucción *IDelete*.

Los generadores de corrección

<table border="1"> <tr> <td>x</td><td>y</td><td>z</td></tr> </table>	x	y	z	Ejes de coordenadas de trayectoria.
x	y	z		
<table border="1"> <tr> <td>0</td><td>0</td><td>3</td></tr> </table>	0	0	3	Generador de corrección vertical, con la suma de sus propias correcciones de trayectoria.
0	0	3		
<table border="1"> <tr> <td>0</td><td>1</td><td>0</td></tr> </table>	0	1	0	Generador de corrección horizontal, con la suma de sus propias correcciones de trayectoria
0	1	0		
<table border="1"> <tr> <td>-</td><td>-</td><td>-</td></tr> </table>	-	-	-	Generador de corrección no conectado.
-	-	-		
<table border="1"> <tr> <td>-</td><td>-</td><td>-</td></tr> </table>	-	-	-	Generador de corrección no conectado.
-	-	-		
<table border="1"> <tr> <td>-</td><td>-</td><td>-</td></tr> </table>	-	-	-	Generador de corrección no conectado.
-	-	-		
<table border="1"> <tr> <td>0</td><td>1</td><td>3</td></tr> </table>	0	1	3	La suma de todas las correcciones realizadas por todos los generadores de correcciones conectados.
0	1	3		

Figura 3 Generadores de corrección.

Limitaciones

Un número máximo de 5 generadores de corrección podrán ser conectados simultáneamente.

Sintaxis

CorrCon
[Descr ':='] < variable (**VAR**) de *corrdescr* > ';'

Información relacionada

Descripción en:

Desconexión de un generador de correcciones Instrucciones - *CorrDiscon*

Escritura en un generador de correcciones Instrucciones - *CorrWrite*

Lectura de todos los offsets actuales Funciones - *CorrRead*

Eliminación de todos los generadores de correcciones Instrucciones - *CorrClear*

Descriptor de un generador de correcciones Tipos de datos - *corrdescr*

CorrDiscon**Desconexión de un generador de correcciones**

CorrDiscon sirve para desconectar el sistema de un generador de correcciones previamente conectado. Esta instrucción servirá también para eliminar correcciones proporcionadas previamente.

Ejemplo

```
VAR corrdescr id;  
...  
CorrCon id;  
...  
CorrDiscon id;
```

CorrDiscon desconecta del generador de correcciones previamente conectado cuya referencia viene indicada por el descriptor *id*.

Argumentos**CorrDiscon Descr**

Descr	Tipo de dato: <i>corrdescr</i>
--------------	--------------------------------

Descriptor del generador de correcciones.

Ejemplo

Véase las instrucciones - *CorrCon*

Sintaxis

```
CorrDiscon  
[ Descr ':=' ] < variable (VAR) de corrdescr > ';'
```

Información relacionada

Conexión a un generador de correcciones
Escritura en un generador de correcciones
Lectura de todos los offsets actuales
Eliminación de todos los generadores de correcciones
Descriptor de correcciones

Descripción:

Instrucciones - *CorrCon*
Instrucciones - *CorrWrite*
Funciones - *CorrRead*

Instrucciones - *CorrClear*
Tipos de datos- *corrdescr*

CorrWrite

Escritura en un generador de correcciones

CorrWrite sirve para escribir offsets en el sistema de coordenadas de la trayectoria en un generador de correcciones.

Ejemplo

```
VAR corrdescr id;  
VAR pos offset;  
...  
CorrWrite id, offset;
```

Los offsets utilizados, almacenados en la variable *offset*, son indicados en el generador de correcciones cuya referencia está determinada por el descriptor *id*.

Argumentos

CorrWrite Descr Datos

Descr	Tipo de dato: <i>corrdescr</i>
Descriptor del generador de correcciones.	
Datos	Tipo de dato: <i>pos</i>
El offset que se debe indicar.	

Ejemplo

Véase Instrucciones - *CorrCon*

Limitaciones

El mejor resultado se obtiene en trayectorias rectas. Cuanto mayor sea la velocidad y los ángulos entre trayectorias lineales consecutivas, la desviación respecto a la trayectoria esperada será mayor. El mismo concepto es válido también para los círculos con un radio decreciente.

Sintaxis

CorrWrite
[Descr ':='] < variable (**VAR**) de *corrdescr* > ','
[Data ':='] < expresión (**IN**) de *pos* > ';'

Información relacionada

Descripción en:

Conexión a un generador de correcciones	Instrucciones - <i>CorrCon</i>
Desconexión de un generador de correcciones	Instrucciones - <i>CorrDiscon</i>
Lectura de todos los offsets actuales	Funciones - <i>CorrRead</i>
Eliminación de todos los generadores de correcciones	Instrucciones - <i>CorrClear</i>
Descriptor de un generador de correcciones	Tipos de datos - <i>corrdescr</i>

DeactUnit Desactivación de una unidad mecánica

DeactUnit sirve para desactivar una unidad mecánica.

Puede utilizarse para determinar la unidad que debe estar activada, por ejemplo, cuando se usan unidades de accionamiento comunes.

Ejemplos

DeactUnit orbit_a;

Desactivación de la unidad mecánica *orbit_a*.

```
MoveL p10, v100, fine, herramienta1;  
DeactUnit transportador;  
MoveL p20, v100, z10, herramienta1;  
MoveL p30, v100, fine, herramienta1;  
ActUnit transportador;  
MoveL p40, v100, z10, herramienta1;
```

La unidad *transportador* estará estacionaria cuando el robot se mueva a la posición *p20* y *p30*. Después de esto, tanto el robot como el *transportador* se moverán a la posición *p40*.

```
MoveL p10, v100, fine, herramienta1;  
DeactUnit orbit1;  
ActUnit orbit2;  
MoveL p20, v100, z10, herramienta1;
```

La unidad *orbit1* será desactivada y la unidad *orbit2* activada.

Argumentos

DeactUnit MecUnit

MecUnit	<i>(Unidad Mecánica)</i>	Tipo de dato: <i>mecunit</i>
----------------	--------------------------	------------------------------

El nombre de la unidad mecánica que deberá ser desactivada.

Ejecución del programa

Cuando el robot y los ejes externos se hayan inmovilizado, la unidad mecánica especificada será desactivada. Esto significa que no será controlada ni monitorizada hasta que se vuelva a activar.

En el caso en que varias unidades mecánicas comparten una misma unidad de accionamiento, la desactivación de una de las unidades mecánicas desconectará también esta unidad de la unidad de accionamiento común.

Limitaciones

La instrucción DeactUnit no puede ser utilizada

- en secuencias de programa StorePath ... RestoPath
- en la rutina de evento REARRANQUE
- cuando uno de los ejes de la unidad mecánica está en modo independiente

La instrucción de movimiento anterior a esta instrucción deberá terminar con un punto de paro para que en esta instrucción sea posible realizar un rearranque después de un corte de potencia.

Sintaxis

DeactUnit
[MecUnit ':='] < variable (**VAR**) de *mecunit*> ';'

Información relacionada

Activación de las unidades mecánicas
Unidades mecánicas

Descripción:

Instrucciones - *ActUnit*
Tipos de Datos - *mecunit*

Decr**Disminución de 1**

Decr sirve para restar 1 de una variable o persistente numérica.

Ejemplo

Decr reg1;

I será restado de *reg1*, es decir, *reg1:=reg1-1*.

Argumentos**Decr Name****Name**Tipo de dato: *num*

El nombre de la variable o persistente que se desea disminuir.

Ejemplo

```
TPReadNum num_part, "Cuántas piezas deben ser procesadas? ";
WHILE num_part>0 DO
    part_produc;
    dec_part;
ENDWHILE
```

Se pide al operador que introduzca el número de piezas que van a ser procesadas.
La variable *num_part* sirve para contar el número que quedan por procesar.

Sintaxis**Decr**

[Name ':='] < variable o persistente (**INOUT**) de *num* > ';

Información relacionada

Incrementar una variable de 1
Restar cualquier valor de una variable
Cambio de datos utilizando una expresión arbitraria, por ejemplo, multiplicación

Descripción:

Instrucciones - *Incr*
Instrucciones - *Add*
Instrucciones - *:=*

EOffsOff Desactivación de un offset de los ejes externos

EOffsOff (External Offset Off) sirve para desactivar un offset de los ejes externos.

El offset de los ejes externos será activado mediante la instrucción *EOffsSet* o *EOffsOn* y se aplica a todos los movimientos hasta que se active cualquier otro offset de ejes externos o hasta que se desactive el offset de los ejes externos.

Ejemplos

EOffsOff;

Desactivación del offset de los ejes externos.

```
MoveL p10, v500, z10, herramienta;  
EOffsOn \ExeP:=p10, p11;  
MoveL p20, v500, z10, herramienta;  
MoveL p30, v500, z10, herramienta;  
EOffsOff;  
MoveL p40, v500, z10, herramienta;
```

Un offset está definido como la diferencia entre la posición de cada eje en la posición *p10* y *p11*. Este desplazamiento afecta a los movimientos a *p20* y *p30*, pero no a *p40*.

Ejecución del programa

Los offsets activados de los ejes externos serán reactivados.

Sintaxis

EOffsOff ‘;’

Información relacionada

Descripción:

Definición de un offset utilizando dos posiciones

Instrucciones - *EOffsOn*

Definición de los valores utilizados por los offsets

Instrucciones - *EOffsSet*

Desactivación del movimiento de desplazamiento del robot

Instrucciones - *PDispOff*

EOffsOn Activación de un offset de los ejes externos

EOffsOn (External Offset On) sirve para definir y activar un offset de ejes externos utilizando dos posiciones.

Ejemplos

MoveL p10, v500, z10, herramienta;
EOffsOn \ExeP:=p10, p20;

Activación de un offset de los ejes externos. Esto se calcula para cada eje, basándose en la diferencia entre las posiciones *p10* y *p20*.

MoveL p10, v500, fine, herramienta;
EOffsOn *;

Activación de un offset de los ejes externos. Dado que se ha utilizado un punto de paro en la instrucción precedente, no será necesario utilizar el argumento \ExeP. El desplazamiento será calculado sobre la base de la diferencia entre la posición utilizada de cada eje y el punto programado (*) almacenado en la instrucción.

Argumentos

EOffsOn [\ExeP] PuntProg

[\ExeP] (Punto Ejecutado)

Tipo de dato: *robtarget*

La nueva posición de los ejes durante la ejecución del programa. Si se omite este argumento, se utilizará la posición normal de los ejes durante la ejecución del programa.

PuntProg (Punto Programado)

Tipo de dato: *robtarget*

La posición original de los ejes durante la programación.

Ejecución del programa

El offset será calculado como la diferencia entre *ExeP* y *PuntProg* para cada eje externo separado. En el caso en que el *ExeP* no haya sido especificado, se utilizará en su lugar la posición normal de los ejes durante la ejecución del programa. Dado que es la posición correspondiente de los ejes la que se ha utilizado, los ejes no deberán moverse cuando *EOffsOn* ha sido ejecutada.

Este offset se utilizará entonces para desplazar la posición de los ejes externos en instrucciones de posicionamiento siguientes y permanecerá activado hasta que se active otro offset (la instrucción *EOffsSet* o *EOffsOn*) o hasta que se desactive el offset de los ejes externos (la instrucción *EOffsOff*).

Sólo se podrá activar un offset para cada eje externo individual a la vez. Por otra parte, se podrán programar varios *EOffsOn* uno detrás de otro. En el caso en que lo sean, los diferentes offset serán añadidos.

El offset de los ejes externos será reinicializado automáticamente:

- a la puesta en marcha en frío
- cuando se carga un programa nuevo
- cuando se ejecuta la primera instrucción en el programa.

Ejemplo

```
SearchL sen1, pbusc, p10, v100, herram1;  
PDispOn \ExeP:=pbusc, *, herram1;  
EOffsOn \ExeP:=pbusc, *;
```

Se realiza una búsqueda en la que la posición buscada del robot y de los ejes externos se encuentra almacenada en la posición pbusc. Cualquier movimiento realizado después de esto empezará a partir de esta posición utilizando un desplazamiento de programa tanto del robot como de los ejes externos. Esto se calcula basándose en la diferencia entre la posición buscada y el punto programado (*) almacenado en la instrucción.

Sintaxis

```
EOffsOn  
[ '\' ExeP ':=' < expresión (IN) de robtarget > ',' ]  
[ PuntProg ':=' ] < expresión (IN) de robtarget > ','
```

Información relacionada

	<u>Descripción</u>
Desactivación del offset de los ejes externos	Instrucciones - <i>EOffsOff</i>
Definición del offset utilizando valores	Instrucciones - <i>EOffsSet</i>
Desplazamiento de los movimientos del robot	Instrucciones - <i>PDispOn</i>
Sistemas de Coordenadas	Principios de Movimiento y de E/S - <i>Sistemas de Coordenadas</i>

EOffsSet**Activación de un offset de ejes externos utilizando un valor**

EOffsSet (External Offset Set) sirve para definir y activar un offset de ejes externos que utiliza valores.

Ejemplo

```
VAR extjoint eax_a_p100 := [100, 0, 0, 0, 0, 0];
```

```
EOffsSet eax_a_p100;
```

La activación de un offset *eax_a_p100* de ejes externos, significa (siempre y cuando el eje externo “a” sea lineal) que:

- El sistema de coordenadas ExtOffs ha sido desplazado de 100 mm para el eje lógico “a” (véase la Figura 1).
- Mientras que el offset está activado, todas las posiciones serán desplazadas de 100 mm en la dirección del eje x-.

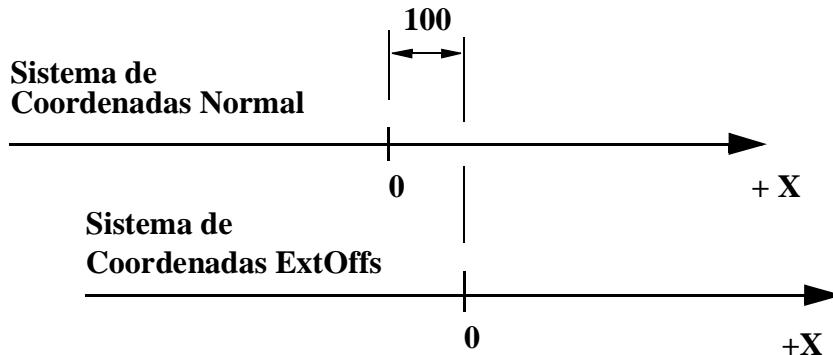


Figura 1 Desplazamiento de un eje externo.

Argumentos**EOffsSet EAxOffs**

EAxOffs (Offset Eje Externo)

Tipo de dato: *extjoint*

El offset de los ejes externos será definido como un dato del tipo *extjoint*, expresado en:

- mm para los ejes lineales
- grados para los ejes rotativos

Ejecución del programa

El offset de los ejes externos se activará cuando la instrucción *EOffsSet* se active y permanecerá en este estado hasta que se active cualquier otro offset (la instrucción *EOffsSet* o *EOffsOn*) o hasta que se desactive el offset de los ejes externos (la instrucción *EOffsOff*).

Sólo se podrá activar un offset de ejes externos a la vez. No se deberá acumular los offset utilizando la instrucción *EOffsSet*.

El offset de los ejes externos serán reseteados automáticamente:

- a la puesta en marcha
- cuando se carga un programa nuevo
- cuando se ejecuta la primera instrucción del programa

Sintaxis

```
EOffsSet
[ EAxOffs ':=' ] <expresión (IN) de extjoint> ';
```

Información relacionada

Descripción en:

Desactivación de un offset de ejes externos	Instrucciones - <i>EOffsOff</i>
Definición de un offset utilizando dos posiciones	Instrucciones - <i>EOffsSet</i>
Desplazamiento de los movimientos del robot	Instrucciones - <i>PDispOn</i>
Definición de los datos del tipo <i>extjoint</i>	Tipos de datos - <i>extjoint</i>
Sistemas de Coordenadas	Principios de Movimiento y de E/S - <i>Sistemas de Coordenadas</i>

ErrWrite Escribir un Mensaje de Error

ErrWrite (Error Write) sirve para visualizar un mensaje de error en la unidad de programación y escribirlo en la lista de mensajes del robot.

Ejemplo

```
ErrWrite "PLC error" , "Fatal error in PLC" \RL2:="Call service";
Stop;
```

Un mensaje está almacenado en la lista del robot. El mensaje aparece también en el visualizador de la unidad de programación.

```
ErrWrite \ W, " Search error" , "No hit for the first search";
RAISE try_search_again;
```

Un mensaje está almacenado únicamente en la lista del robot. A continuación, la ejecución del programa prosigue.

Argumentos

ErrWrite [\W] Header Reason [\RL2] [\RL3] [\RL4]

[\W]	<i>(Aviso)</i>	Tipo de dato: <i>switch</i>
---------------	----------------	-----------------------------

Produce un mensaje de aviso que será almacenado únicamente en la lista de mensajes de error del robot (y por tanto, no aparecerá directamente en el visualizador de la unidad de programación).

Header		Tipo de dato: <i>string</i>
---------------	--	-----------------------------

Es el encabezado del mensaje de error (24 caracteres como máximo).

Reason		Tipo de dato: <i>string</i>
---------------	--	-----------------------------

Es el motivo que ha provocado el error (línea 1 de 40 caracteres como máximo).

[\RL2]	<i>(Motivo Línea 2)</i>	Tipo de dato: <i>string</i>
----------------	-------------------------	-----------------------------

Es el motivo que ha provocado el error (línea 2 de 40 caracteres como máximo).

[\RL3]	<i>(Motivo Línea 3)</i>	Tipo de dato: <i>string</i>
----------------	-------------------------	-----------------------------

Es el motivo que ha provocado el error (línea 3 de 40 caracteres como máximo).

[\RL4]	<i>(Motivo Línea 4)</i>	Tipo de dato: <i>string</i>
----------------	-------------------------	-----------------------------

Es el motivo que ha provocado el error (línea 4 de 40 caracteres como máximo).

Ejecución del programa

Un mensaje de error (5 líneas como máximo) aparece visualizado en la unidad de programación y es escrito en la lista de mensajes del robot.

ErrWrite siempre genera el error de programa nº 80001 o en el caso de ocurrir un aviso (argumento \W) genera el nº 80002.

Limitaciones

La longitud total de la cadena (Encabezado+Motivo+\RL2\RL3\RL4) está limitada a 145 caracteres.

Sintaxis

```
ErrWrite
[ '\' W ',' ]
[ Header '==' ] < expresión (IN) de string> ','
[ Reason '==' ] < expresión (IN) de string>
[ '\' RL2 '==' < expresión (IN) de string> ]
[ '\' RL3 '==' < expresión (IN) de string> ]
[ '\' RL4 '==' < expresión (IN) de string> ] ';'
```

Información relacionada

Visualización de un mensaje únicamente en la unidad de programación

Listas de mensajes

Descripción:

Instrucciones - *TPWrite*

Guía del Usuario - *Servicio*

EXIT**Fin de ejecución del programa**

EXIT sirve para finalizar la ejecución del programa. El rearranque del programa se bloquea, es decir, que el programa sólo podrá ser rearrancado únicamente a partir de la primera instrucción de la rutina principal (si el punto de arranque no ha sido movido manualmente).

La instrucción *EXIT* deberá utilizarse cuando ocurren errores muy graves o cuando se desea parar la ejecución del programa definitivamente. La instrucción *Stop* se usa para parar de forma temporal la ejecución del programa.

Ejemplo

```
ErrWrite "Fatal error", "Illegal state";
EXIT;
```

La ejecución del programa se para y no puede ser rearrancada a partir de esta posición en el programa.

Sintaxis

```
EXIT ';
```

Información relacionada

Descripción:

Parada temporal de la ejecución del programa *Instrucciones - Stop*

EXIT

Instrucciones

ExitCycle**Interrupción del ciclo actual
y arranque del siguiente**

ExitCycle sirve para interrumpir el ciclo actual y volver a colocar el indicador de principio de programa a la primera instrucción de la rutina principal. En caso de estar activado el modo de ejecución CONT, la ejecución empezará a ejecutar el ciclo siguiente.

Ejemplo

```
VAR num cyclecount:=0;  
VAR intnum error_intno;  
  
PROC main()  
    IF cyclecount = 0 THEN  
        CONNECT error_intno WITH error_trap;  
        ISignalDI di_error,1,error_intno;  
    ENDIF  
    cyclecount:=cyclecount+1;  
    ! start to do something intelligent  
    ....  
  
ENDPROC  
  
TRAP error_trap  
    TPWrite "ERROR, I will start on the next item";  
    ExitCycle;  
ENDTRAP
```

Esta secuencia iniciará el ciclo siguiente si la señal *di_error* está activada.

Funcionamiento del programa

Todas las variables, persistentes, interrupciones definidas y características de movimiento permanecen incambiadas.

Sintaxis

```
ExitCycle';'
```

Información relacionada

Paro después de un error fatal
Terminar la ejecución del programa
Paro de la actividad del programa
Acabar la ejecución de una rutina

Descripción:

Instrucciones - *EXIT*
Instrucciones - *EXIT*
Instrucciones - *Stop*
Instrucciones - *RETURN*

Instrucciones

ExitCycle

ExitCycle

Instrucciones

FOR Repetición de un número dado de veces

FOR se usa cuando una o varias instrucciones deben repetirse un número dado de veces.

En el caso en que las instrucciones deban repetirse mientras se cumpla una condición específica, se deberá usar la instrucción **WHILE**.

Ejemplo

```
FOR i FROM 1 TO 10 DO  
    rutina1;  
ENDFOR
```

Rpite el procedimiento *rutina1*, 10 veces.

Argumentos

**FOR Contador bucle FROM Valor inicial TO Valor final
[STEP Incremento] DO ... ENDFOR**

Contador bucle

Identificador

El nombre del dato que contendrá el valor del contador de bucle actual. El dato es declarado automáticamente y su nombre no podrá coincidir con ningún otro nombre de ningún dato ya existente.

Valor inicial

Tipo de dato: *Num*

El valor inicial deseado del contador de bucle.
(suelen ser números enteros).

Valor final

Tipo de dato: *Num*

El valor final deseado del contador de bucle.
(suelen ser números enteros).

Incremento

Tipo de dato: *Num*

El valor con el que el contador de bucle deberá ser incrementado (o disminuido).
(suelen ser números enteros).

Si este valor no está especificado, el valor de incremento será automáticamente 1 (o -1 si el valor inicial es mayor que el valor final).

Ejemplo

```
FOR i FROM 10 TO 2 STEP -1 DO  
    a{i} := a{i-1};  
ENDFOR
```

Los valores de una matriz se ajustan de forma creciente, de la siguiente forma a {10}:=a{9}, a{9}:=a{8} etc.

Ejecución del programa

1. Las expresiones correspondientes a los valores iniciales, final y de incremento serán calculadas.
 2. El contador de bucle tendrá asignado el valor inicial.
 3. El valor del contador de bucle será comprobado para ver si dicho valor se encuentra entre el valor inicial y el valor final o si es igual al valor inicial o al valor final. Si el valor del contador se encuentra fuera de esta gama, el bucle FOR se para y la ejecución del programa continua con la instrucción que sigue ENDFOR.
 4. Las instrucciones del bucle FOR se ejecutarán.
 5. El contador de bucle será incrementado (o disminuido) de acuerdo con el valor de incremento.
 6. El bucle FOR se repetirá, empezando a partir del punto 3.
-

Limitaciones

Sólo se podrá acceder al contador de bucle (del tipo de dato *num*) desde dentro del bucle FOR y por consiguiente esconderá los demás datos y rutinas que tengan el mismo nombre. Sólo podrá ser leído (no actualizado) por las instrucciones contenidas en el bucle FOR.

No se podrán utilizar valores decimales para los valores iniciales, finales o de paro, en combinación con una condición de terminación exacta para el bucle FOR (no definido en el caso en que el último bucle esté funcionando o no).

Sintaxis

(EBNF)

```
FOR <variable bucle> FROM <expresión> TO <expresión>  
    [ STEP <expresión> ] DO  
        <lista instrucciones>  
    ENDFOR  
<variable bucle> ::= <identificador>
```

Información relacionada

Descripción en:

Expresiones

Características Básicas - *Expresiones*

Identificadores

Características Básicas -
Elementos de Base

FOR

Instrucciones

GOTO Ir a una instrucción nueva identificada por una etiqueta (label)

GOTO sirve para transferir la ejecución del programa a otra línea (una etiqueta) dentro de la misma rutina.

Ejemplos

GOTO sig;

sig:

La ejecución del programa continúa con la instrucción que viene después de *next*.

reg1 := 1;
sig:

reg1 := reg1 + 1;
IF reg1<=5 GOTO sig;

El bucle de programa *sig* será ejecutado cinco veces.

IF reg1>100 GOTO valoralto;
valorbajo:

GOTO prepar;
valoralto:

prepar:

Si *reg1* es mayor que 100, el bucle de programa *valoralto* será ejecutado; de lo contrario, se ejecutará el bucle *valorbajo*.

Argumentos**GOTO Etiqueta****Etiqueta**

Identificador

La etiqueta a partir de donde la ejecución del programa deberá continuar.

Limitaciones

Sólo se podrá transferir la ejecución del programa a una etiqueta dentro de la misma rutina.

Sólo se podrá transferir la ejecución del programa a una etiqueta dentro de una instrucción IF o TEST si la instrucción GOTO está también situada dentro de la misma rama de esta instrucción.

Sólo se podrá transferir la ejecución del programa a una etiqueta dentro de una instrucción FOR o WHILE si la instrucción GOTO está también situada dentro de esta instrucción.

Sintaxis

(EBNF)

GOTO <identificador>;'

Información Relacionada

Etiqueta

Descripción:

Instrucciones - *etiqueta*

Otras instrucciones que cambian el flujo
del programa

Resumen RAPID -
Control del Flujo de Programa

GripLoad Definición de la carga útil del robot

GripLoad sirve definir la carga útil que el robot sujeta con su pinza.

Descripción



Es importante definir siempre la carga actual de la herramienta y cuando se use, la carga útil del robot. Definiciones incorrectas de los datos de carga pueden provocar una sobrecarga de la estructura mecánica del robot.

Si se especifican datos incorrectos de carga, se pueden generar las siguientes consecuencias:

- Si el valor de los datos de carga especificados es mayor que el valor real de la carga;
 - > El robot no será utilizado a su capacidad máxima
 - > La precisión de trayectoria es incorrecta incluyendo el riesgo de vibraciones.
- Si el valor de los datos de carga especificados es menor que el valor real de la carga;
 - > La precisión de trayectoria es incorrecta incluyendo el riesgo de vibraciones.
 - > Existe un riesgo de sobrecarga de la estructura mecánica

Ejemplos

`GripLoad pieza1;`

La pinza del robot sujeta una carga llamada *pieza1*.

`GripLoad load0;`

La pinza del robot suelta todas las cargas.

Argumentos

GripLoad Carga

Carga

Tipo de dato: *loaddata*

Los datos de carga que describen la carga útil utilizada.

Ejecución del programa

La carga especificada afecta la capacidad del robot.

La carga por defecto, 0 kg, se activa automáticamente

- a la puesta en marcha en frío
- cuando se carga un programa nuevo
- cuando se ejecuta el programa desde el principio.

Sintaxis

```
GripLoad  
[ Carga ':=' ] <persistente (PERS) de loaddata > ';
```

Información relacionada

Definición de los datos de carga
Definición de tool load

Descripción:

Tipos de Datos - *loaddata*
Tipos de Datos - *tooldata*

IDelete**Anulación de una interrupción**

IDelete (Interrupt Delete) sirve para anular (borrar) una interrupción.

En el caso en que se desee inhabilitar temporalmente la instrucción, se deberá utilizar la instrucción *ISleep* o *IDisable*.

Ejemplo

IDelete aliment_bajo;

La interrupción *aliment_bajo* será anulada.

Argumentos**IDelete Interrupción****Interrupción**

Tipo de dato: *intnum*

La identidad de la interrupción.

Ejecución del programa

La definición de la interrupción ha sido borrada completamente. Para volver a definirla, deberá primero ser conectada de nuevo a una rutina de tratamiento de interrupciones.

La instrucción deberá estar precedida de un punto de paro. De lo contrario, la interrupción será desactivada antes de alcanzar el punto final.

No se deberá borrar las interrupciones; esta operación se realiza automáticamente cuando:

- se carga un programa nuevo
- se rearanca el programa desde el principio
- se mueve el puntero del programa al principio de una rutina

Sintaxis**IDelete**

[Interrupción ‘:=’] < variable (**VAR**) de *intnum* > ‘;’

Información relacionada

Resumen de interrupciones

Inhabilitación temporal de una interrupción

Inhabilitación temporal de
todas las interrupciones

Descripción:

Resumen RAPID - *Interrupciones*

Instrucciones - *ISleep*

Instrucciones - *IDisable*

IDisable Inhabilitación de las interrupciones

IDisable (Interrupt Disable) sirve para inhabilitar todas las interrupciones temporalmente. Esta función puede utilizarse por ejemplo, en una parte particularmente sensible del programa donde no se permite que ocurran interrupciones para no perturbar su ejecución normal.

Ejemplo

```
IDisable;  
FOR i FROM 1 TO 100 DO  
    caracter[i]:=ReadBin(sensor);  
ENDFOR  
IEnable;
```

No se permiten interrupciones mientras se está leyendo el canal serie.

Ejecución del programa

Las interrupciones que ocurren mientras una instrucción *IDisable* está activada, quedarán registradas en una cola. Cuando las interrupciones vuelvan a ser admitidas por el programa, empezarán a generarse inmediatamente, y se ejecutarán una por una siguiendo el orden de la cola.

Sintaxis

```
IDisable‘;’
```

Información Relacionada

Descripción:

Resumen de interrupciones
Permiso de las interrupciones

Resumen RAPID - *Interrupción*
Instrucciones - *IEnable*

IDisable

Instrucciones

IEnable Habilitación de las interrupciones

IEnable (Interrupt Enable) sirve para habilitar las interrupciones durante la ejecución del programa.

Ejemplo

```
IDisable;  
FOR i FROM 1 TO 100 DO  
    caracter[i]:=ReadBin(sensor);  
ENDFOR  
IEnable;
```

No se permiten las interrupciones mientras se está leyendo el canal serie. Una vez finalizada la lectura, las interrupciones serán de nuevo permitidas.

Ejecución del programa

Las interrupciones que ocurren mientras una instrucción *IDisable* está activada, quedan registradas en una cola. Cuando las interrupciones vuelven a ser permitidas (*IEnable*), empiezan a generarse, ejecutándose una por una siguiendo el orden de la cola. Luego, la ejecución del programa continúa en el programa normal y las interrupciones que se producen después de esto serán tratadas según van produciéndose.

Las interrupciones siempre están permitidas cuando se arranca un programa desde el principio. Las interrupciones inhabilitadas mediante la instrucción *ISleep* no se verán afectadas por la instrucción *IEnable*.

Sintaxis

```
IEnable‘;’
```

Información Relacionada

Resumen de interrupciones
Inhabilitación de interrupciones

Descripción:

Resumen RAPID - *Interrupciones*
Instrucciones - *IDisable*

IF**Si se cumple una condición, entonces...; si no...**

IF sirve cuando diferentes instrucciones deben cumplirse según se cumpla o no una condición.

Ejemplos

```
IF reg1 > 5 THEN  
    Set do1;  
    Set do2;  
ENDIF
```

Las señales *do1* y *do2* son activadas únicamente si *reg1 es mayor que 5*.

```
IF reg1 > 5 THEN  
    Set do1;  
    Set do2;  
ELSE  
    Reset do1;  
    Reset do2;  
ENDIF
```

Las señales *do1* y *do2* son activadas o desactivadas dependiendo si *reg1 es mayor que 5* o no.

Argumentos

```
IF Condición THEN ...  
{ELSEIF Condición THEN ...}  
[ELSE ...]  
ENDIF
```

Condición	Tipo de dato: <i>bool</i>
------------------	---------------------------

La condición que debe cumplirse para que las instrucciones entre THEN y ELSE/ELSEIF puedan ejecutarse.

Ejemplo

```
IF contador > 100 THEN  
    contador := 100;  
ELSEIF contador < 0 THEN  
    contador := 0;  
ELSE  
    contador := contador + 1;
```

ENDIF

El *contador* ha sido incrementado en 1. Sin embargo, si el valor del *contador* está fuera del límite 0-100, el *contador* tendrá asignado el valor límite correspondiente.

Ejecución del programa

Las condiciones son comprobadas siguiendo un orden secuencial, hasta que una de ellas se cumpla. La ejecución del programa continúa con las instrucciones asociadas a esa condición. Si no se cumple ninguna de las condiciones, la ejecución del programa continúa con las instrucciones que siguen ELSE. Si se cumple más de una condición, sólo se ejecutarán las instrucciones que están asociadas con la primera de las condiciones.

Sintaxis

(EBNF)

```
IF <expresión condicional> THEN
    <lista instrucciones>
{ELSEIF <expresión condicional> THEN <lista instrucciones> | <EIF>}
[ELSE
    <lista instrucciones>]
ENDIF
```

Información Relacionada

Descripción:

Condiciones (expresiones lógicas)

Características Básicas - *Expresiones*

Incr**Incremento de 1**

Incr sirve para añadir 1 a una variable o persistente numérica.

Ejemplo

Incr reg1;

I será añadido a *reg1*, es decir, *reg1:=reg1+1*.

Argumentos**Incr Nombre**

Nombre

Tipo de dato: *num*

El nombre de la variable o persistente que se desea cambiar.

Ejemplo

```
WHILE DInput(stop_produc)=0 DO
    produc_part;
    Incr_parts;
    TPWrite "Nº de piezas producidas= "\Num:=parts;
ENDWHILE
```

En cada ciclo el número de piezas producidas será actualizado en la unidad de programación. El proceso de producción continua su ejecución mientras no se active la señal *stop_production*.

Sintaxis

Incr

[Nombre ':='] < variable o persistente (**INOUT**) de *num* > ':'

Información relacionada

Disminuir una variable de 1
Adición de cualquier valor a una variable
Cambio de datos utilizando una expresión arbitraria, por ejemplo, la multiplicación

Descripción:

Instrucciones - *Decr*
Instrucciones - *Add*
Instrucciones - *:=*

IndAMove

Movimiento de una posición absoluta independiente

IndAMove sirve para cambiar un eje del modo en que está, pasarlo al modo independiente y moverlo a una posición específica.

Un eje independiente es un eje que se mueve de forma independiente respecto a otros ejes del sistema robot. Dado que la ejecución del programa continua inmediatamente, se podrá ejecutar otras instrucciones (incluyendo instrucciones de posicionamiento) durante el tiempo que el eje independiente se está moviendo.

En el caso en que el eje deba ser movido dentro de una revolución, se deberá utilizar la instrucción *IndRMove* en su lugar. Si el movimiento debe producirse a una corta distancia de la posición actual, se deberá usar la instrucción *IndDMove*.

Ejemplo

```
IndAMove Station_A,2\ToAbsPos:=p4,20;
```

El eje 2 de *Station_A* será movido a la posición p4 a la velocidad de 20 grados/s.

Argumentos

IndAMove MecUnit Axis [\ToAbsPos] | [\ToAbsNum] Speed [\Ramp]

MecUnit	(<i>Unidad Mecánica</i>)	Tipo de dato: <i>mecunit</i>
----------------	----------------------------	------------------------------

El nombre de la unidad mecánica.

Axis	Tipo de dato: <i>num</i>
-------------	--------------------------

El número del eje utilizado por la unidad mecánica (1-6).

[\ToAbsPos]	(<i>A Posición Absoluta</i>)	Tipo de dato: <i>robtarget</i>
----------------------	--------------------------------	--------------------------------

La posición del eje especificada como un tipo de dato *robtarget*. Sólo se utiliza el componente para este eje específico. El valor es utilizado como un valor de posición absoluta en grados (mm para los ejes lineales).

La posición del eje será afectada si el eje ha sido desplazado utilizando la instrucción *EOffsSet* o *EOffsOn*.

Para los ejes del robot, se deberá utilizar el argumento *\ToAbsNum*.

[\ToAbsNum] (*A valor Numérico Absoluto*) Tipo de dato: *num*

La posición de eje definida en grados (mm para el eje lineal).

Con este argumento, la posición NO estará afectada por ningún desplazamiento, por ejemplo, *EOffsSet* o *PDispOn*.

Cumple la misma función que *\ToAbsPos* pero la diferencia es que la posición es definida como un valor numérico, para facilitar el cambio manual de la posición.

Speed Tipo de dato: *num*

La velocidad del eje en grados/s (mm/s para el eje lineal).

[\Ramp] Tipo de dato: *num*

Disminución de la aceleración y la deceleración en función de la capacidad máxima (1 - 100%, 100% = capacidad máxima).

Ejecución del programa

Cuando se ejecuta *IndAMove*, el eje especificado empieza a moverse con la velocidad programada a la posición de ejes especificada. Si se ha programado *\Ramp*, se producirá una reducción de la aceleración/deceleración.

Para volver a colocar los ejes en el modo normal, se deberá utilizar la instrucción *IndReset*. En conexión con esto, se podrá cambiar la posición lógica del eje, de forma que un cierto número de revoluciones serán borradas de la posición, por ejemplo, para evitar la rotación para el siguiente movimiento.

La velocidad podrá ser modificada ejecutando otra instrucción *IndAMove* (u otra instrucción *Ind_Move*). Si se ha seleccionado una velocidad en la dirección opuesta, el eje se detiene y luego acelera para alcanzar la nueva dirección y la nueva velocidad.

Para una ejecución paso a paso de la instrucción, el eje estará activado únicamente en modo independiente. El eje inicia su movimiento cuando se ejecuta la siguiente instrucción, y continua mientras la ejecución del programa tiene lugar. Para más información al respecto, véase el Capítulo 6, Principios de movimiento y de E/S.

Cuando se mueve el puntero del programa al principio del programa, o a una rutina nueva, todos los ejes se activan automáticamente en el modo normal, sin cambiar el sistema de medida (equivale a la ejecución de la instrucción *IndReset\Old*).

Observése que el uso de una instrucción *IndAMove* después de una operación *IndCMove* puede provocar que el eje realice el mismo número de revoluciones que ha realizado en la instrucción *IndCMove* en sentido inverso. Para impedir esto, se deberá utilizar una instrucción *IndReset* antes de la instrucción *IndAMove*, o utilizar una instrucción *IndRMove*.

Limitaciones

En el modo independiente, no se podrá mover los ejes. En el caso en que se haya intentado hacer funcionar el eje manualmente, el eje no se moverá y se generará un mensaje de error. Se deberá ejecutar una instrucción *IndReset* o mover el puntero del programa al programa principal, para salir del modo independiente.

Si se produce una pérdida de tensión cuando un eje está en modo independiente, el programa no podrá ser rearrancado. El sistema generará un mensaje de error y el programa deberá ser arrancado desde el principio.

Ejemplo

```
ActUnit Station_A;  
weld_stationA;  
IndAMove Station_A,1\ToAbsNum:=90,20\Ramp:=50;  
ActUnit Station_B;  
weld_stationB_1;  
WaitUntil IndInpos(Station_A,1 ) = TRUE;  
WaitTime 0.2;  
DeactUnit Station_A;  
weld_stationB_2;
```

Station_A se activa cuando la soldadura ha comenzado en la estación A.

Station_A (eje 1) será entonces movido a la posición de 90 grados mientras el robot está realizando la soldadura en la estación B. La velocidad del eje es de 20 grados/s. La velocidad cambia con la aceleración/deceleración reducida a 50% de su capacidad máxima.

Cuando la estación A alcanza esta posición, será desactivada, y podrá tener lugar de nuevo el proceso de carga en la estación, al mismo tiempo que el robot continúa el proceso de soldadura en la estación B.

Gestión de errores

Si los ejes no están activados, la variable del sistema ERRNO pasará a ser ERR_AXIS_ACT. Este error podrá entonces ser manipulado en el gestor de errores.

Sintaxis

IndAMove

```
[ MecUnit' := ] < variable (VAR) de mecunit > ','  
[ Axis' := ] < expresión (IN) de num >  
[ '\'ToAbsPos' := ] < expresión (IN) de robtarget >  
| [ '\' ToAbsNum' := ] < expresión (IN) de num > ','  
[ Speed' := ] < expresión (IN) de num >  
[ '\' Ramp' := ] < expresión (IN) de num > ] ';'
```

Información relacionada

	<u>Descripción en:</u>
Ejes independientes en general	Principios de Movimiento y de E/S - <i>Ejecución del programa</i>
Volver a pasar al modo normal	Instrucciones - <i>IndReset</i>
Reinicializar el sistema de medida	Instrucciones - <i>IndReset</i>
Mover un eje independiente a una posición específica dentro de la revolución actual	Instrucciones - <i>IndRMove</i>
Mover un eje independiente a una distancia específica	Instrucciones - <i>IndDMove</i>
Comprobar el estado de velocidad de los ejes independientes	Funciones - <i>IndSpeed</i>
Comprobar el estado de posición de los ejes independientes	Funciones - <i>IndIpos</i>

IndCMove**Movimiento continuo independiente**

IndCMove sirve para cambiar el modo de un eje al modo independiente e iniciar su movimiento de forma continua a una velocidad específica.

Un eje independiente es un eje que se mueve de forma independiente respecto a otros ejes del sistema robot. Dado que la ejecución del programa continua inmediatamente, se podrá ejecutar otras instrucciones (incluyendo instrucciones de posicionamiento) durante el tiempo que el eje independiente se está moviendo.

Ejemplo

IndCMove Station_A,2,-30.5;

El eje 2 de *Station_A* empieza a moverse en una dirección negativa a una velocidad de 30,5 grados/s.

Argumentos**IndCMove MecUnit Axis Speed [\Ramp]**

MecUnit	(<i>Unidad Mecánica</i>)	Tipo de dato: <i>mecunit</i>
----------------	----------------------------	------------------------------

El nombre de la unidad mecánica.

Axis	Tipo de dato: <i>num</i>
-------------	--------------------------

El número del eje utilizado por la unidad mecánica (1-6).

Speed	Tipo de dato: <i>num</i>
--------------	--------------------------

La velocidad del eje en grados/s (mm/s para el eje lineal).

La dirección del movimiento está especificada con el signo del argumento de velocidad.

[\Ramp]	Tipo de dato: <i>num</i>
------------------	--------------------------

Disminución de la aceleración y la deceleración en función de la capacidad máxima (1 - 100%, 100% = capacidad máxima).

Ejecución del programa

Cuando se ejecuta *IndCMove*, el eje especificado empieza a moverse a la velocidad programada. La dirección del movimiento viene especificada por el signo del argumento de velocidad. Si se ha programado el argumento *\Ramp* se producirá una reducción de la aceleración/deceleración.

Para volver a colocar los ejes en el modo normal, se deberá utilizar la instrucción *IndReset*. Con *IndReset* la posición lógica del eje podrá ser cambiada, de forma que el número de revoluciones realizadas anteriormente serán borradas, con el objeto, por ejemplo, de evitar la rotación en sentido inverso en el siguiente movimiento.

La velocidad podrá ser modificada ejecutando otra instrucción *IndCMove*. Si se ha ordenado una velocidad en la dirección opuesta, el eje se detiene y luego acelera para alcanzar la nueva dirección y la nueva velocidad. Para parar el eje, se usará el argumento de velocidad 0. Entonces seguirá estando en modo independiente.

Durante la ejecución paso a paso de la instrucción, el eje estará activado únicamente en modo independiente. El eje inicia su movimiento cuando se ha ejecutado la siguiente instrucción, y continua mientras la ejecución del programa tiene lugar. Para más información al respecto, véase el Capítulo 6, Principios de Movimiento y de E/S.

Cuando se mueve el puntero del programa al principio del programa, o a una rutina nueva, todos los ejes se activan automáticamente en modo normal, sin cambiar el sistema de medida (equivale a la ejecución de la instrucción *IndReset\Old*).

Limitaciones

La resolución de la posición del eje empeorará cuanto más se aleje de su posición cero lógica (normalmente el medio del área de trabajo). Para volver a conseguir una alta resolución, el área de trabajo lógica será puesta a cero con la instrucción *IndReset*. Para más información al respecto, consultar el Capítulo 6, Principios de Movimiento y de E/S.

En el modo independiente, no se podrá mover los ejes. En el caso en que se haya intentado hacer funcionar el eje manualmente, éste no se moverá y el sistema generará un mensaje de error. Se deberá ejecutar una instrucción *IndReset* o mover el puntero del programa al menú principal, para salir del modo independiente.

Si se produce una pérdida de tensión cuando un eje está en modo independiente, el programa no podrá ser rearrancado. El sistema generará un mensaje de error y el programa deberá ser arrancado desde el principio.

Ejemplo

```
IndCMove Station_A,2,20;
WaitUntil IndSpeed(Station_A,2 \InSpeed) = TRUE;
WaitTime 0.2;
MoveL p10, v1000, fine, tool1;
IndCMove Station_A,2,-10\Ramp:=50;
MoveL p20, v1000, z50, tool1;
IndRMove Station_A,2 \ToRelPos:=p1 \Short,10;
MoveL p30, v1000, fine, tool1;
WaitUntil IndInpos(Station_A,2 ) = TRUE;
WaitTime 0.2;
IndReset Station_A,2 \RefPos:=p40\Short;
MoveL p40, v1000, fine, tool1;
```

El eje 2 de *Station_A* empieza a moverse en una dirección positiva a una velocidad de 20 grados/s. Cuando este eje haya alcanzado la velocidad seleccionada los ejes del robot empezarán a moverse.

Cuando el robot alcanza la posición *p10*, el eje externo cambia de dirección y gira a una velocidad de 10 grados/s. El cambio de velocidad se lleva a cabo con una aceleración/deceleración reducida al 50% de la capacidad máxima. Al mismo tiempo, el robot se moverá hacia *p20*.

Entonces, el eje 2 de *Station_A* se para lo más rápidamente posible en la posición *p1* dentro de la revolución actual.

Cuando el eje 2 ha alcanzado esta posición, y que el robot se ha parado en la posición *p30*, el eje 2 regresará de nuevo al modo normal. El offset del sistema de medida para este eje es cambiado de un número entero de revoluciones de eje, de forma que la posición actual está lo más cerca posible de *p40*.

Cuando el robot se mueve entonces a la posición *p40*, el eje 2 de *Station_A* se moverá, utilizando la vía más corta, a la posición *p40* (máx. ± 180 grados).

Gestión de errores

Si los ejes no están activados, la variable del sistema ERRNO pasará a ser ERR_AXIS_ACT. Este error podrá entonces ser manipulado en el gestor de errores.

Sintaxis

```
IndCMove
[ MecUnit':=' ] < variable (VAR) de mecunit > ',' 
[ Axis':=' ] < expresión (IN) de num > ',' 
[ Speed ':=' ] < expresión (IN) de num >
[ '\' Ramp':=' < expresión (IN) de num > ] ';
```

Información relacionada

Descripción:

Ejes independientes en general	Principios de Movimiento y de E/S - <i>Ejecución del programa</i>
Volver a pasar al modo normal	Instrucciones - <i>IndReset</i>
Reinicializar el sistema de medida	Instrucciones - <i>IndReset</i>
Mover un eje independiente a una posición específica	Instrucciones - <i>IndAMove, IndRMove</i>
Mover un eje independiente de una distancia específica	Instrucciones - <i>IndDMove</i>
Comprobar el estado de velocidad de los ejes independientes	Funciones - <i>IndSpeed</i>
Comprobar el estado de posición de los ejes independientes	Funciones - <i>IndInpos</i>

IndDMove

Movimiento de una posición delta independiente

IndDMove sirve para cambiar el modo de un eje al modo independiente y mover el eje de una distancia específica.

Un eje independiente es un eje que se mueve de forma independiente respecto a otros ejes del sistema robot. Dado que la ejecución del programa continua inmediatamente, se podrá ejecutar otras instrucciones (incluyendo instrucciones de posicionamiento) durante el tiempo que el eje independiente se está moviendo.

En el caso en que se deba mover el eje a una posición específica, se deberá utilizar la instrucción *IndAMove* o *IndRMove* en su lugar.

Ejemplo

IndDMove Station_A,2,-30,20;

El eje 2 de *Station_A* se mueve de 30 grados en una dirección negativa a una velocidad de 20 grados/s.

Argumentos

IndDMove MecUnit Axis Delta Speed [\Ramp]

MecUnit *(Unidad Mecánica)* Tipo de dato: *mecunit*

El nombre de la unidad mecánica.

Axis Tipo de dato: *num*

El número del eje utilizado por la unidad mecánica (1-6).

Delta Tipo de dato: *num*

La distancia de la que el eje actual debe ser movido, expresada en grados (mm para los ejes lineales). El signo especifica la dirección del movimiento.

Speed Tipo de dato: *num*

La velocidad del eje en grados/s (mm/s para el eje lineal).

[\Ramp] Tipo de dato: *num*

Disminución de la aceleración y la deceleración en función de la capacidad máxima (1 - 100%, 100% = capacidad máxima).

Ejecución del programa

Cuando se ejecuta *IndAMove*, el eje especificado empieza a moverse a la velocidad programada de la distancia determinada. La dirección del movimiento viene indicada por el signo del argumento *Delta*. Si se ha programado el argumento *\Ramp* se producirá una reducción de la aceleración/deceleración.

Si el eje se está moviendo, la nueva posición es calculada a partir de la posición instantánea de los ejes, cuando la instrucción *IndDMove* es ejecutada. Si una instrucción *IndDMove* con distancia 0 es ejecutada, el eje se parará y luego regresará a la posición que tenía en el momento en que se ejecutó la instrucción.

Para volver a colocar los ejes en el modo normal, se deberá utilizar la instrucción *IndReset*. Con *IndReset* la posición lógica del eje podrá ser cambiada, de forma que el número de revoluciones realizadas anteriormente serán borradas, con el objeto, por ejemplo, de evitar la rotación en sentido inverso en el siguiente movimiento.

La velocidad podrá ser modificada ejecutando otra instrucción *IndDMove* (u otra instrucción *Ind_Move*). Si se ha seleccionado una velocidad en la dirección opuesta, el eje se detiene y luego acelera para alcanzar la nueva dirección y la nueva velocidad.

Durante la ejecución paso a paso de la instrucción, el eje estará activado únicamente en modo independiente. El eje inicia su movimiento cuando se ha ejecutado la siguiente instrucción, y continua mientras la ejecución del programa tiene lugar. Para más información al respecto, véase el Capítulo 6, Principios de Movimiento y de E/S.

Cuando se mueve el puntero del programa al principio del programa, o a una rutina nueva, todos los ejes se activan automáticamente en modo normal, sin cambiar el sistema de medida (equivale a la ejecución de la instrucción *IndReset\Old*).

Limitaciones

En el modo independiente, no se podrá mover los ejes. En el caso en que se haya intentado hacer funcionar el eje manualmente, éste no se moverá y el sistema generará un mensaje de error. Se deberá ejecutar una instrucción *IndReset* o mover el puntero del programa al menú principal, para salir del modo independiente.

Si se produce una pérdida de tensión cuando un eje está en modo independiente, el programa no podrá ser rearrancado. El sistema generará un mensaje de error y el programa deberá ser arrancado desde el principio.

Ejemplo

```
IndAMove Robot,6\ToAbsNum:=90,20;
WaitUntil IndInpos(Station_A,1 ) = TRUE;
WaitTime 0.2;
IndDMove Station_A,2,-30,20;
WaitUntil IndInpos(Station_A,1 ) = TRUE;
WaitTime 0.2;
IndDMove Station_A,2,400,20;
```

El eje 6 del robot se moverá a las siguientes posiciones:
 90 grados
 60 grados
 460 grados (1 revolución + 100 grados).

Gestión de errores

Si los ejes no están activados, la variable del sistema ERRNO pasará a ser ERR_AXIS_ACT. Este error podrá entonces ser manipulado en el gestor de errores.

Sintaxis

```
IndDMove
[ MecUnit':=' ] < variable (VAR) de mecunit > ',' 
[ Axis':=' ] < expresión (IN) de num > ',' 
[ Delta':=' ] < expresión (IN) de num > ',' 
[ Speed ':=' ] < expresión (IN) de num > 
[ '\' Ramp':=' < expresión (IN) de num > ] ';
```

Información relativa

	<u>Descripción:</u>
Ejes independientes en general	Principios de Movimiento y de E/S - <i>Ejecución del programa</i>
Volver a pasar al modo normal	Instrucciones - <i>IndReset</i>
Reinicializar el sistema de medida	Instrucciones - <i>IndReset</i>
Mover un eje independiente a una posición específica	Instrucciones - <i>IndAMove, IndRMove</i>
Comprobar el estado de velocidad de los ejes independientes	Funciones - <i>IndSpeed</i>
Comprobar el estado de posición de los ejes independientes	Funciones - <i>IndInpos</i>

IndReset**Reinicialización independiente**

IndReset sirve para volver a cambiar un eje independiente al modo normal. Al mismo tiempo, el sistema de medida de los ejes de rotación podrán ser movidos de un cierto número de revoluciones.

Ejemplo

```
IndCMove Station_A,2,5;
MoveL *,v1000,fine,tool1;
IndCMove Station_A,2,0;
WaitUntil IndSpeed(Station_A,2\ZeroSpeed);
WaitTime 0.2
IndReset Station_A,2;
```

El eje 2 de *Station_A* se moverá primero en el modo independiente y luego regresará al modo normal. El eje mantendrá su posición.

Téngase en cuenta que el eje independiente actual, y los ejes normales, no deberán moverse cuando se ha ejecutado la instrucción *IndReset*. Esto es porque la posición previa es un punto de paro, y que una instrucción *IndCMove* es ejecutada a la velocidad cero. Además, se utiliza una pausa de 0,2 segundos para garantizar que el ha alcanzado el estado correcto.

Argumentos

IndReset MecUnit Axis [\RefPos] | [\RefNum] [\Short] | [\Fwd] | [\Bwd] | [\Old]

MecUnit	<i>(Unidad Mecánica)</i>	Tipo de dato: <i>mecunit</i>
----------------	--------------------------	------------------------------

El nombre de la unidad mecánica.

Axis	Tipo de dato: <i>num</i>
-------------	--------------------------

El número del eje utilizado por la unidad mecánica (1-6).

[\RefPos]	<i>(Posición de referencia)</i>	Tipo de dato: <i>robtarget</i>
--------------------	---------------------------------	--------------------------------

La posición del eje especificada como un tipo de dato *robtarget*. Sólo se utiliza el componente para este eje específico. La posición deberá estar dentro del área de trabajo normal.

Para los ejes del robot, se deberá utilizar en su lugar el argumento *\RefNum*.

El argumento sólo deberá ser definido junto con el argumento *\Short*, *\Fwd* o *\Bwd*. No está permitido junto con el argumento *\Old*.

[\RefNum] *(valor de referencia numérico)* Tipo de dato: *num*

La posición del eje definida en grados (mm para el eje lineal). La posición debe encontrarse dentro del área de trabajo normal.

El argumento sólo deberá definirse junto con el argumento *\Short*, *\Fwd* o *\Bwd*. No está permitido junto con el argumento *\Old*.

Tiene la misma función que *\RefPos* pero la posición es definida como un valor numérico para facilitar el cambio de posición de forma manual.

[\Short] Tipo de dato: *switch*

El sistema de medida cambiará de un número entero de revoluciones en el lado del eje, de forma que el eje esté lo más cerca posible de la posición *\RefPos* o *\RefNum* especificada. Si se ejecuta una instrucción de posicionamiento con la misma posición después de *IndReset*, el eje se moverá utilizando la ruta más corta, menor que ± 180 grados, a fin de alcanzar la posición.

[\Fwd] *(Hacia adelante)* Tipo de dato: *switch*

El sistema de medida cambiará de un número entero de revoluciones en el lado del eje, de forma que la posición de referencia se encontrará en el lado positivo de la posición *\RefPos* o *\RefNum* especificada. Si una instrucción de posicionamiento con la misma posición es ejecutada después de *IndReset*, el eje girará en dirección positiva menos que 360 grados a fin de alcanzar la posición.

[\Bwd] *(Hacia atrás)* Tipo de dato: *switch*

El sistema de medida cambiará de un número entero de revoluciones en el lado del eje de forma que la posición de referencia se encontrará en el lado negativo de la posición *\RefPos* o *\RefNum* especificada. Si una instrucción de posicionamiento con la misma posición es ejecutada después de *IndReset*, el eje girará en dirección negativa menos que 360 grados a fin de alcanzar la posición.

[\Old] Tipo de dato: *switch*

Mantiene la antigua posición. Téngase en cuenta que la resolución es disminuida en posiciones muy lejanas de cero.

Si no se especifica ningún argumento *\Short*, *\Fwd*, *\Bwd* o *\Old*, se utilizará *\Old* como valor por defecto.

Ejecución del programa

Cuando se ejecuta *IndReset*, el eje independiente regresará al modo normal. Al mismo tiempo, el sistema de medida de este eje podrá moverse de un número completo de revoluciones de eje.

La instrucción también podrá utilizarse en el modo normal a fin de cambiar el sistema de medida.

Observar que se utiliza la posición sólo para ajustar el sistema de medida - el eje no se moverá a la posición.

Limitaciones

La instrucción sólo puede ejecutarse cuando todos los ejes activos que funcionan en modo normal están inmóviles. El eje en modo independiente que será cambiado al modo normal también deberá ser estacionario. Para los ejes en modo normal, esto se consigue ejecutando una instrucción de movimiento con el argumento *fine*. El eje independiente se para con una instrucción *IndCMove* que tiene *Speed:=0* (seguida de un periodo de espera de 0,2 segundos), *IndRMove*, *IndAMove* o *IndDMove*.

La resolución de posiciones es disminuida conforme se va apartando de la posición lógica 0. Un eje que progresivamente gira cada vez más allá de la posición 0 deberá ponerse en posición cero utilizando la instrucción *IndReset* con un argumento distinto de *\Old*.

El sistema de medida no puede cambiarse para los ejes lineales.

Para garantizar un arranque adecuado después la reinicialización *IndReset* de un eje con un sistema de medida relativo (interruptores de sincronización), se deberá añadir un retraso adicional de 0,12 segundos después de la instrucción *IndReset*.

Únicamente el eje 6 del robot podrá usarse como eje independiente. La instrucción *IndReset* también podrá utilizarse para el eje 4 en los modelos IRB 2400 e IRB 4400. Si se usa *InsReset* en el eje 4 del robot, entonces el eje 6 no deberá estar en el modo independiente.

Ejemplo

```
IndAMove Station_A,1\ToAbsNum:=750,50;  
WaitUntil IndInpos(Station_A,1);  
WaitTime 0.2;  
IndReset Station_A,1 \RefNum:=0 \Short;  
. . .  
IndAMove Station_A,1\ToAbsNum:=750,50;  
WaitUntil IndInpos(Station_A,1);  
WaitTime 0.2;  
IndReset Station_A,1 \RefNum:=300 \Short;
```

El eje 1 de *Station_A* se moverá en primer lugar independientemente de la posición de 750 grados (2 revoluciones y 30 grados). Al mismo tiempo, mientras cambia al modo normal, la posición lógica es determinada a 30 grados.

El eje 1 de *Station_A* se moverá luego a la posición de 750 grados (2 revoluciones y 30 grados). Al mismo tiempo, mientras cambia al modo normal, la posición lógica es determinada a 390 grados (1 revolución y 30 grados).

Gestión de errores

Si el eje se está moviendo, la variable del sistema ERRNO pasará a ser ERR_AXIS_MOVING.

Si el eje no está activado, la variable del sistema ERRNO pasará a ser ERR_AXIS_ACT. Este error podrá entonces ser manipulado en el gestor de errores.

Sintaxis

```
IndReset
[ MecUnit':=' ] < variable (VAR) de mecunit > ','  

[ Axis':=' ] < expresión (IN) de num >  

[ '\' RefPos':=' < expresión (IN) de robtarget > ]  

[ [ '\' RefNum':=' < expresión (IN) de num > ]  

[ '\' Short ] | [ '\' Fwd ] | [ '\' Bwd ] | [ '\' Old ]';'
```

Información relacionada

Ejes independientes en general

Descripción en:

Principios de Movimiento y de E/S -
Ejecución del programa

Cambio de un eje al modo independiente

Instrucciones - *IndAMove, IndCMove,*
IndDMove, IndRMove

Comprobar el estado de velocidad de los
ejes independientes

Funciones - *IndSpeed*

Comprobar el estado de posición de los
ejes independientes

Funciones - *IndInpos*

IndRMove**Movimiento de una posición relativa independiente**

IndRMove sirve para cambiar el modo de un eje de rotación al modo independiente y mover el eje a una posición específica dentro de una revolución.

Un eje independiente es un eje que se mueve de forma independiente respecto a otros ejes del sistema robot. Dado que la ejecución del programa continua inmediatamente, se podrá ejecutar otras instrucciones (incluyendo instrucciones de posicionamiento) durante el tiempo que el eje independiente se está moviendo.

En el caso en que se deba mover el eje a una posición absoluta (varias revoluciones), o si el eje es lineal, se deberá utilizar la instrucción *IndAMove* en su lugar. Si el movimiento debe tener lugar a cierta distancia a partir de la posición actual, se deberá utilizar la instrucción *IndDMove*.

Ejemplo

```
IndRMove Station_A,2\ToRelPos:=p5 \Short,20;
```

El eje 2 de *Station_A* se moverá siguiendo la vía más corta a la posición *p5* dentro de una revolución (rotación máxima de ± 180 grados) a una velocidad de 20 grados/s.

Argumentos

IndRMove MecUnit Axis [\ToRelPos] | [\ToRelNum] [\Short] | [\Fwd] | [\Bwd] Speed [\Ramp]

MecUnit	<i>(Unidad Mecánica)</i>	Tipo de dato: <i>mecunit</i>
----------------	--------------------------	------------------------------

El nombre de la unidad mecánica.

Axis	Tipo de dato: <i>num</i>
-------------	--------------------------

El número del eje utilizado por la unidad mecánica (1-6).

[\ToRelPos]	<i>(A Posición Relativa)</i>	Tipo de dato: <i>robtarget</i>
----------------------	------------------------------	--------------------------------

La posición especificada como un tipo de dato *robtarget*. Sólo se utiliza el componente para este eje específico. El valor es utilizado como un valor de posición en grados dentro de una revolución de eje. Esto significa que el eje se mueve menos que de una revolución.

La posición del eje será afectada si el eje es desplazado mientras se utiliza la instrucción *EOffsSet* o *EOffsOn*.

Para los ejes del robot, se deberá utilizar en su lugar el argumento *\ToRelNum*.

[\ToRelNum]	(A valor numérico relativo)	Tipo de dato: num
La posición del eje definida en grados.		
Cuando se utiliza este argumento, la posición NO será afectada por ningún desplazamiento, por ejemplo, <i>EOffset</i> o <i>PDispOn</i> .		
Tiene la misma función que <i>ToRelPos</i> pero la posición es definida como un valor numérico para facilitar el cambio de la posición de forma manual.		
[\Short]		Tipo de dato: switch
El eje se mueve a la posición nueva siguiendo la ruta más corta. Esto significa que la rotación máxima será de 180 grados en cualquier dirección. Por lo tanto, la dirección del movimiento dependerá de la situación actual del eje.		
[\Fwd]	(Hacia adelante)	Tipo de dato: switch
El eje se mueve en dirección positiva a la posición nueva. Esto significa que la rotación máxima será de 360 grados y siempre en una dirección positiva (valor de posición incrementado).		
[\Bwd]	(Hacia Atrás)	Tipo de dato: switch
El eje se mueve en dirección negativa a la posición nueva. Esto significa que la rotación máxima será de 360 grados y siempre en una dirección negativa (valor de posición disminuido).		
Si se omite el argumento <i>Short</i> , <i>Fwd</i> o <i>Bwd</i> , <i>Short</i> será utilizado como valor por defecto.		
Speed		Tipo de dato: num
La velocidad del eje en grados/s.		
[\Ramp]		Tipo de dato: num
Disminución de la aceleración y la deceleración en función de la capacidad máxima (1 - 100%, 100% = capacidad máxima).		

Ejecución del programa

Cuando se ejecuta *IndRMove*, el eje especificado empieza a moverse con la velocidad programada, a la posición de eje indicada, pero con una sola revolución como máximo. Si se ha programado el argumento *|Ramp* se producirá una reducción de la aceleración/deceleración.

Para volver a colocar los ejes en el modo normal, se deberá utilizar la instrucción *IndReset*. Con *IndReset* la posición lógica del eje podrá ser cambiada, de forma que el número de revoluciones realizadas anteriormente serán borradas, con el objeto, por ejemplo, de evitar la rotación en sentido inverso en el siguiente movimiento.

La velocidad podrá ser modificada ejecutando otra instrucción *IndRMove* (u otra instrucción *Ind_Move*). Si se ha seleccionado una velocidad en la dirección opuesta, el eje se detiene y luego acelera para alcanzar la nueva dirección y la nueva velocidad.

Durante la ejecución paso a paso de la instrucción, el eje estará activado únicamente en modo independiente. El eje inicia su movimiento cuando se ha ejecutado la siguiente instrucción, y continua mientras la ejecución del programa tiene lugar. Para más información al respecto, véase el Capítulo 6, Principios de Movimiento y de E/S.

Cuando se mueve el puntero del programa al principio del programa, o a una rutina nueva, todos los ejes se activan automáticamente en modo normal, sin cambiar el sistema de medida (equivale a la ejecución de la instrucción *IndReset\Old*).

Limitaciones

En el modo independiente, no se podrá mover los ejes. En el caso en que se haya intentado hacer funcionar el eje manualmente, éste no se moverá y el sistema generará un mensaje de error. Se deberá ejecutar una instrucción *IndReset* o mover el puntero del programa al menú principal, para salir del modo independiente.

Si se produce una pérdida de tensión cuando un eje está en modo independiente, el programa no podrá ser rearrancado. El sistema generará un mensaje de error y el programa deberá ser arrancado desde el principio.

Ejemplos

```
IndRMove Station_A,1\ToRelPos:=p5 \Fwd,20\Ramp:=50;
```

El eje 1 de *Station_A* empieza a moverse en una dirección positiva a la posición *p5* dentro de una revolución (rotación máxima de 360 grados) a una velocidad de 20 grados/s. La velocidad será cambiada con una aceleración/deceleración reducida al 50% de la capacidad máxima.

```
IndAMove Station_A,1\ToAbsNum:=90,20;
WaitUntil IndInpos(Station_A,1 ) = TRUE;
IndRMove Station_A,1\ToRelNum:=80 \Fwd,20;
WaitTime 0.2;
WaitUntil IndInpos(Station_A,1 ) = TRUE;
WaitTime 0.2;
IndRMove Station_A,1\ToRelNum:=50 \Bwd,20;
WaitUntil IndInpos(Station_A,1 ) = TRUE;
WaitTime 0.2;
IndRMove Station_A,1\ToRelNum:=150 \Short,20;
WaitUntil IndInpos(Station_A,1 ) = TRUE;
WaitTime 0.2;
IndAMove Station_A,1\ToAbsNum:=10,20;
```

El eje 1 de *Station_A* se mueve a las posiciones siguientes:

- 90 grados
- 440 grados (1 revolución + 80 grados)
- 410 grados (1 revolución + 50 grados)
- 510 grados (1 revolución + 150 grados)
- 10 grados

Gestión de errores

Si los ejes no están activados, la variable del sistema ERRNO pasará a ser ERR_AXIS_ACT. Este error podrá entonces ser manipulado en el gestor de errores.

Sintaxis

```
IndRMove
[ MecUnit' := ] < variable (VAR) de mecunit > ,
[ Axis' := ] < expresión (IN) de num >
[ '\'ToRelPos' := ] < expresión (IN) de robttargets >
| [ '\'ToRelNum' := ] < expresión (IN) de num >
| [ '\'Short ] | [ '\' Fwd ] | [ '\' Bwd ] ,
[ Speed' := ] < expresión (IN) de num >
[ '\'Ramp' := ] < expresión (IN) de num > ] ;
```

Información relacionada

Descripción:

Ejes independientes en general	Principios de Movimiento y de E/S - <i>Ejecución del programa</i>
Volver a pasar al modo normal	Instrucciones - <i>IndReset</i>
Reinicializar el sistema de medida	Instrucciones - <i>IndReset</i>
Mover un eje independiente a una posición absoluta	Instrucciones - <i>IndAMove</i>
Mover un eje independiente de una distancia específica	Instrucciones - <i>IndDMove</i>
Más ejemplos	Instrucciones - <i>IndCMove</i>
Comprobar el estado de velocidad de los ejes independientes	Funciones - <i>IndSpeed</i>
Comprobar el estado de posición de los ejes independientes	Funciones - <i>IndInpos</i>

InvertDO Inversión del valor de una señal de salida digital

InvertDO (Invert Digital Output) invierte el valor de una señal de salida digital (0 -> 1 y 1 -> 0).

Ejemplo

InvertDO do15;

El valor utilizado del valor de la señal *do15* será invertida.

Argumentos

InvertDO Señal

Señal

Tipo de dato: *signaldo*

El nombre de la señal que se desea invertir.

Ejecución del programa

El valor utilizado por la señal es invertido (véase la Figura 1).

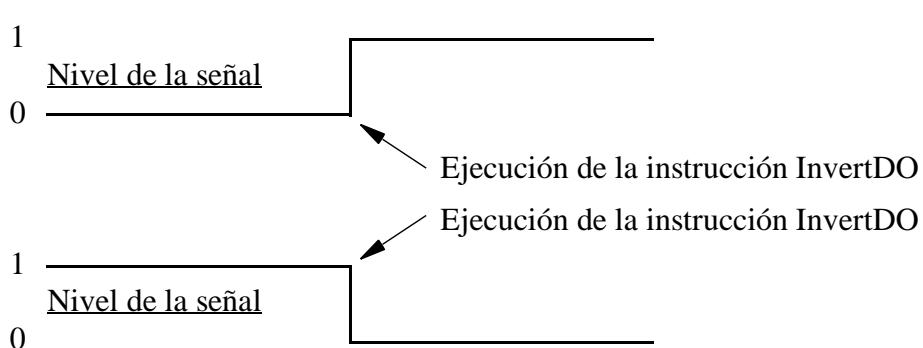


Figura 1 Inversión de una señal de salida digital.

Sintaxis

InvertDO
 [Señal ':='] < variable (**VAR**) de *signaldo* > ';

Información Relacionada

Descripción:

Instrucciones de entrada/salida

Resumen RAPID -
Señales de entrada y salida

Funciones de las entradas/salidas
en general

Principios de Movimiento y E/S -
Principios de E/S

Configuración de las E/S

Guía del Usuario - *Parámetros del
sistema*

IODisable**Inhabilitar la unidad de E/S**

IODisable sirve para inhabilitar una unidad de E/S durante la ejecución del programa (sólo en el sistema S4C).

Las unidades de E/S son automáticamente habilitadas después de la puesta en marcha siempre y cuando estén definidas en los parámetros del sistema. Cuando se requiere por alguna razón, las unidades de E/S pueden ser inhabilitadas o habilitadas durante la ejecución del programa.

Ejemplos

IODisable “cell1”, 5;

Inhabilitar la unidad de E/S que lleva el nombre *cell1*. Tiempo de espera máx. 5 s.

Argumentos

IODisable NombreUnidad TiempoMax

NombreUnidad

Tipo de dato: *string*

El nombre de la unidad de E/S que se desea inhabilitar (con el mismo nombre que el configurado).

TiempoMax

Tipo de dato: *num*

El período máximo de tiempo de espera permitido, expresado en segundos. Si este tiempo transcurre antes de que la unidad de E/S haya terminado los pasos de inhabilitación, el sistema llamará al gestor de errores, si hay uno, con el código de error **ERR_IODISABLE**. En el caso en que no haya ningún gestor de errores, la ejecución se detendrá.

La inhabilitación de una unidad de E/S toma aproximadamente entre 2-5 s.

Ejecución del programa

La unidad de E/S especificada inicia los pasos de inhabilitación. La instrucción está lista cuando los pasos de inhabilitación han terminado. Si el *TiempoMax* transcurre antes de que la unidad de E/S haya terminado sus pasos de inhabilitación, se generará un error recuperable.

Después de inhabilitar una unidad de E/S, cualquier activación de una salida en esta unidad producirá un error.

Ejemplo

```

PROC go_home()
  VAR num recover_flag :=0;
  ...
  ! Start to disable I/O unit cell1
  recover_flag := 1;
  IODisable "cell1", 0;
  ! Move to home position
  MoveJ home, v1000,fine,tool1;
  ! Wait until disable of I/O unit cell1 is ready
  recover_flag := 2;
  IODisable "cell1", 5;
  ...
  ERROR
    IF ERRNO = ERR_IODISABLE THEN
      IF recover_flag = 1 THEN
        TRYNEXT;
      ELSEIF recover_flag = 2 THEN
        RETRY;
      ENDIF
    ELSEIF ERRNO = ERR_EXCRTYMAX THEN
      ErrWrite "IODisable error", "Not possible to disable I/O unit cell1";
      Stop;
    ENDIF
  ENDPROC

```

Para ahorrar tiempo de ciclo, la unidad de E/S *cell1* es inhabilitada durante el movimiento del robot hacia su posición inicial (*home*). Cuando el robot está en su posición inicial, se realizará una prueba para determinar si la unidad de E/S *cell1* está totalmente inhabilitada. Después del número máximo de reintentos (5 con un tiempo de espera de 5 s), la ejecución del robot se detendrá y se producirá un mensaje de error.

El mismo principio podrá utilizarse con la instrucción *IOEnable* (*HabilitarE/S*) (ello ahorrará más tiempo de ciclo comparado con *IODisable* (*Inhabilitar E/S*)).

Sintaxis

```

IODisable
[ NombreUnidad ':=' ] < expresión (IN) de string > ;
[ TiempoMax ':=' ] < expresión (IN) de num > ;

```

Información relacionada

	<u>Descripción en:</u>
Habilitar una unidad de E/S	Instrucciones - <i>IOEnable</i>
Instrucciones de Entrada/Salida	Resumen RAPID - <i>Señales de Entrada y Salida</i>
Funciones de Entradas/Salidas en general	Principios de Movimiento y de E/S - <i>Principios de E/S</i>
Configuración de E/S	Guía del Usuario - <i>Parámetros del Sistema</i>

IOEnable**Habilitar la unidad de E/S**

IOEnable sirve para habilitar una unidad de E/S durante la ejecución del programa (sólo en el sistema S4C).

Las unidades de E/S son automáticamente habilitadas después de la puesta en marcha siempre y cuando estén definidas en los parámetros del sistema. Cuando se requiere por alguna razón, las unidades de E/S pueden ser inhabilitadas o habilitadas durante la ejecución del programa.

Ejemplos

IOEnable “cell1”, 5;

Habilitar la unidad de E/S que lleva el nombre *cell1*. Tiempo de espera máx. 5 s.

Argumentos

IOEnable NombreUnidad TiempoMax

NombreUnidad

Tipo de dato: *string*

El nombre de la unidad de E/S que se desea habilitar (con el mismo nombre que el configurado).

TiempoMax

Tipo de dato: *num*

El período máximo de tiempo de espera permitido, expresado en segundos. Si este tiempo transcurre antes de que la unidad de E/S haya terminado los pasos de habilitación, el sistema llamará al gestor de errores, si hay uno, con el código de error **ERR_IOENABLE**. En el caso en que no haya ningún gestor de errores, la ejecución se detendrá.

La habilitación de una unidad de E/S toma aproximadamente entre 2-5 s.

Ejecución del programa

La unidad de E/S especificada inicia los pasos de habilitación. La instrucción está lista cuando los pasos de habilitación han terminado. Si el *TiempoMax* transcurre antes de que la unidad de E/S haya terminado sus pasos de habilitación, se generará un error recuperable.

Después de una secuencia de *IODisable - IOEnable*, todas las salidas de la unidad de E/S actual se activarán en los valores antiguos (antes de *IODisable*).

Ejemplo

IOEnable puede utilizarse también para comprobar si, por alguna razón, hay alguna unidad de E/S desconectada.

```
VAR num max_retry:=0;  
...  
IOEnable "cell1", 0;  
SetDO cell1_sig3, 1;  
...  
ERROR  
  IF ERRNO = ERR_IOENABLE THEN  
    IF max_retry < 5 THEN  
      WaitTime 1;  
      max_retry := max_retry + 1;  
      RETRY;  
    ELSE  
      RAISE;  
    ENDIF  
  ENDIF
```

Antes de utilizar las señales de la unidad de E/S *cell1*, se llevará a cabo una prueba que consiste en intentar la habilitación de la unidad de E/S con un límite de tiempo después de 0 seg. Si la prueba falla, se realiza un salto al gestor de errores. En el gestor de errores, la ejecución del programa espera 1 seg. y luego se realiza otro intento. Después de 5 reintentos, el error ERR_IOENABLE es propagado a la parte del programa que ha llamado esta rutina.

Sintaxis

```
IOEnable  
[ NombreUnidad ':=' ] <expresión (IN) de string> ','  
[ TiempoMax ':=' ] <expresión (IN) de num > ';
```

Información relacionada

Más ejemplos

Descripción en:

Instrucciones - *IODisable*

Inhabilitación de una unidad de E/S

Instrucciones - *IODisable*

Instrucciones de Entrada/Salida

Resumen RAPID -

Señales de Entrada y Salida

Funciones de Entradas/Salidas en general

Principios de Movimiento y de E/S -

Principios de E/S

Configuración de E/S

Guía del Usuario -

Parámetros del Sistema

ISignalDI Orden de interrupción a partir de una señal de entrada digital

ISignalDI (Interrupt Signal Digital In) sirve para ordenar y habilitar una interrupción a partir de una señal de entrada digital.

Las señales del sistema pueden también generar interrupciones.

Ejemplos

```
VAR intnum señint1;
CONNECT señint1 WITH intrutina1;
ISignalDI di1,1,señint1;
```

Ordena una interrupción que ocurrirá cada vez que la señal de entrada digital *di1* se active en *1*. Luego, se realizará una llamada a la rutina de tratamiento de interrupciones *intrutina1*.

```
ISignalDI di1,0,señint1;
```

Ordena una interrupción que ocurrirá cada vez que la señal de entrada digital *di1* se active en *0*.

```
ISignalDI \Single, di1,1,señint1;
```

Ordena una interrupción que ocurrirá únicamente la primera vez que la señal de entrada digital *di1* se active en *1*.

Argumentos

ISignalDI [\Single] Señal ValorUmbra Interrupción

[\Single]

Tipo de dato: *switch*

Especifica si la interrupción debe ocurrir una sola vez o cíclicamente.

Si el argumento *Single* está activado, la interrupción sólo se producirá una vez.
Si se omite el argumento, ocurrirá una interrupción cada vez que se cumpla la condición.

Señal

Tipo de dato: *signaldi*

Es el nombre de la señal que debe generar las interrupciones.

ValorUmbra

Tipo de dato: *dionum*

Es el valor al que la señal debe pasar para que ocurra una interrupción.

El valor está especificado en 0 o 1 o en un valor simbólico (por ejemplo: *abierto/cerrado*). La señal será activada por el flanco ascendente para el cambio de 0 a 1.

Interrupción

Tipo de dato: *intnum*

La identidad de la interrupción. Esta deberá haber sido conectada previamente a una rutina de tratamiento de interrupciones, mediante la instrucción *CONNECT*.

Ejecución del programa

Cuando la señal haya adoptado el valor especificado, se realizará una llamada a la rutina correspondiente de tratamiento de interrupciones. Cuando esto haya sido ejecutado, proseguirá la ejecución del programa a partir de donde ocurrió la interrupción.

En el caso en que la señal cambie al valor especificado antes de que la interrupción haya sido ordenada, no ocurrirá ninguna interrupción (véase la Figura 1).

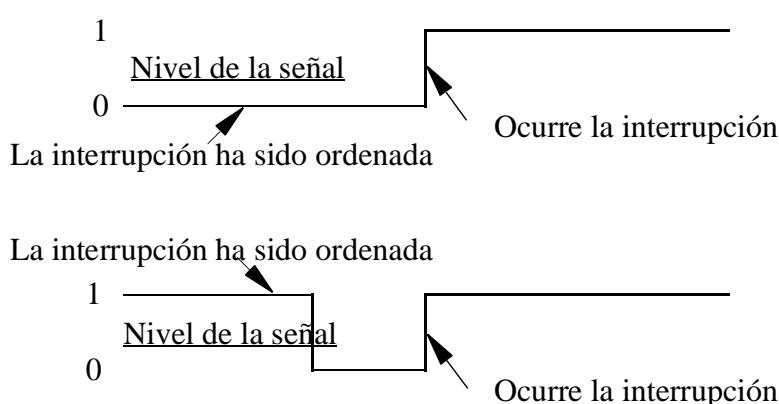


Figura 1 Interrupciones a partir de una señal de entrada digital en el nivel de señal 1.

Limitaciones

La misma variable para la identidad de la interrupción no podrá utilizarse más que una vez, si en primer lugar no se borra. Las interrupciones deberán no obstante utilizarse según se indica en una de las siguientes alternativas.

```

PROC main ()
  VAR intnum sig1int;
  CONNECT sig1int WITH iroutine1;
  ISignalDI di1, 1, sig1int;
  WHILE TRUE DO
    :
    :
  ENDWHILE
ENDPROC

```

Cualquier activación de una interrupción se realizará al principio del programa. Estas

instrucciones se mantendrán por tanto fuera del flujo principal del programa.

```
PROC main ()
    VAR intnum sig1int;
    CONNECT sig1int WITH iroutine1;
    ISignalDI di1, 1, sig1int;
    :
    :
    IDElete sig1int;
ENDPROC
```

La interrupción será borrada al final del programa y luego será reactivada. En este caso, deberá notarse que la interrupción está inactivada durante un corto periodo.

Sintaxis

```
ISignalDI
[ '\' Single',']
[ Señal ':=' ] < variable (VAR) de signaldi > ',' 
[ ValorUmbral':=' ] < expresión (IN) de dionum > ',' 
[ Interrupción ':=' ] < variable (VAR) de intnum > ';'
```

Información relacionada

Resumen de interrupciones

Descripción en:

Resumen RAPID - *Interrupciones*

Interrupción desde una señal de salida

Instrucciones - *ISignalDO*

Más información sobre la gestión
de interrupciones

Características Básicas - *Interrupcio-
nes*

Más ejemplos

Tipos de datos - *intnum*

ISignalDO Orden de interrupción a partir de una señal de salida digital

ISignalDO (Interrupt Signal Digital Out) sirve para ordenar y habilitar una interrupción a partir de una señal de salida digital.

Las señales del sistema pueden también generar interrupciones.

Ejemplos

```
VAR intnum señint1;
CONNECT señint1 WITH intrutina1;
ISignalDO do1,1,señint1;
```

Ordena una interrupción que ocurrirá cada vez que la señal de salida digital *do1* se active en *1*. Luego, se realizará una llamada a la rutina de tratamiento de interrupciones *intrutina1*.

```
ISignalDO do1,0,señint1;
```

Ordena una interrupción que ocurrirá cada vez que la señal de salida digital *do1* se active en *0*.

```
ISignalDO \Single, do1,1,señint1;
```

Ordena una interrupción que ocurrirá únicamente la primera vez que la señal de salida digital *do1* se active en *1*.

Argumentos

ISignalDo [\Single] Señal ValorUmbra Interrupción

[\Single]

Tipo de dato: *switch*

Especifica si la interrupción debe ocurrir una sola vez o cíclicamente.

Si el argumento *Single* está activado, la interrupción sólo se producirá una vez. Si se omite el argumento, ocurrirá una interrupción cada vez que se cumpla la condición.

Señal

Tipo de dato: *signaldo*

Es el nombre de la señal que debe generar las interrupciones.

ValorUmbra

Tipo de dato: *dionum*

Es el valor al que la señal debe pasar para que ocurra una interrupción.

El valor está especificado en 0 o 1 o en un valor simbólico (por ejemplo: *abierto/cerrado*). La señal será activada por el flanco ascendente para el cambio de 0 a 1.

Interrupción

Tipo de dato: *intnum*

La identidad de la interrupción. Esta deberá haber sido conectada previamente a una rutina de tratamiento de interrupciones, mediante la instrucción *CONNECT*.

Ejecución del programa

Cuando la señal haya adoptado el valor especificado, se realizará una llamada a la rutina correspondiente de tratamiento de interrupciones. Cuando ésta haya sido ejecutado, proseguirá la ejecución del programa a partir de donde ocurrió la interrupción.

En el caso en que la señal cambie al valor especificado antes de que la interrupción haya sido ordenada, no ocurrirá ninguna interrupción (véase la Figura 1).

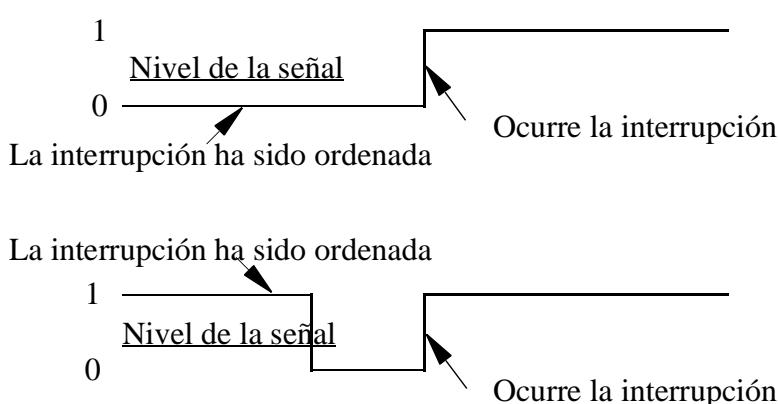


Figura 1 Interrupciones a partir de una señal de salida digital en el nivel de señal 1.

Limitaciones

La misma variable para la identidad de la interrupción no podrá utilizarse más que una vez, si en primer lugar no se borra. Las interrupciones deberán no obstante utilizarse según se indica en una de las siguientes alternativas.

```
PROC main ()
  VAR intnum sig1int;
  CONNECT sig1int WITH iroutine1;
  ISignalDO do1, 1, sig1int;
  WHILE TRUE DO
    :
    :
  ENDWHILE
ENDPROC
```

Cualquier activación de una interrupción se realizará al principio del programa. Estas

instrucciones se mantendrán por tanto fuera del flujo principal del programa.

```
PROC main ()  
    VAR intnum sig1int;  
    CONNECT sig1int WITH iroutine1;  
    ISignalDO do1, 1, sig1int;  
    :  
    :  
    IDElete sig1int;  
ENDPROC
```

La interrupción será borrada al final del programa y luego será reactivada. En este caso, deberá notarse que la interrupción está inactivada durante un corto periodo.

Sintaxis

```
ISignalDO  
[ '\' Single',']  
[ Señal ':=' ] < variable (VAR) de signaldo > ','  
[ ValorUmbral':=' ] < expresión (IN) de dionum > ','  
[ Interrupción ':=' ] < variable (VAR) de intnum > ';
```

Información relacionada

Resumen de interrupciones

Descripción en:

Resumen RAPID - *Interrupciones*

Interrupción desde una señal de entrada

Instrucciones - *ISignalDI*

Más información sobre la gestión
de interrupciones

Características Básicas - *Interrupcio-
nes*

Más ejemplos

Tipos de datos - *intnum*

ISleep**Desactivación de una interrupción**

ISleep (Interrupt Sleep) sirve para desactivar temporalmente una interrupción determinada.

Ejemplo

ISleep señal 1;

La interrupción *señal 1* será desactivada.

Argumentos**ISleep Interrupción****Interrupción**Tipo de dato: *intnum*

La variable (identidad de la interrupción) de la interrupción.

Ejecución del programa

El evento conectado a esta interrupción no la generará hasta que la misma haya sido rehabilitada mediante la instrucción *IWatch*. Las interrupciones generadas mientras la instrucción *ISleep* está activada, serán ignoradas.

Ejemplo

```
VAR intnum intemps;
CONNECT intemps WITH compr_canalserie;
ITimer 60, intemps;

ISleep intemps;
WriteBin canal1, buffer, 30;
IWatch intemps;

TRAP compr_canalserie
    WriteBin canal1, buffer, 1;
    IF ReadBin(canal1\Time:=5) < 0 THEN
        TPWrite "La comunicación serie falla";
        EXIT;
    ENDIF
ENDTRAP
```

La comunicación realizada a través del canal serie canal1 está monitorizada

mediante interrupciones que son generadas cada 60 segundos. La rutina de tratamiento de interrupciones comprueba si la comunicación funciona o no. Mientras la comunicación está en curso, las interrupciones no están permitidas.

Gestión de errores

Las interrupciones que no han sido ordenadas ni habilitadas no son permitidas. En el caso en que el número de interrupción sea desconocido, la variable del sistema ERRNO se activará en ERR_UNKINO (véase “Tipos de Datos - errnum”). El error podrá ser manipulado por el gestor de errores.

Sintaxis

ISleep
[Interrupción ‘:=’] < variable (**VAR**) of *intnum* > ‘;’

Información Relacionada

	<u>Descripción en:</u>
Resumen de interrupciones	Resumen RAPID - <i>Interrupciones</i>
Habilitación de interrupciones	Instrucciones - <i>IWatch</i>
Inhabilitación de todas las interrupciones	Instrucciones - <i>IDisable</i>
Anulación de una interrupción	Instrucciones - <i>IDelete</i>

ITimer Ordena una interrupción temporizada

ITimer (Interrupt Timer) sirve para ordenar y habilitar una interrupción temporizada.

Esta instrucción puede utilizarse, por ejemplo, para comprobar el estado del equipo periférico a cada instante.

Ejemplos

```
VAR intnum intemps;  
CONNECT intemps WITH rutinaint1;  
ITimer 60, intemps;
```

Ordena una interrupción que ocurrirá cíclicamente cada 60 segundos. Se realizará entonces una llamada a la rutina de tratamiento de interrupciones *rutinaint1*.

```
ITimer \Single, 60, intemps;
```

Ordena una interrupción que ocurrirá una sola vez, después de 60 segundos.

Argumentos

ITimer [\Single] Tiempo Interrupción

[\Single]

Tipo de dato: *switch*

Especifica si la interrupción debe ocurrir una sola vez o cíclicamente.

En el caso en que se active el argumento *Single*, la interrupción sólo se producirá una vez. Si se omite el argumento, la interrupción ocurrirá cada vez en el momento especificado.

Tiempo

Tipo de dato: *num*

Es el intervalo de tiempo que debe transcurrir antes de que ocurra la interrupción.

El valor deberá ser especificado en segundos. Si se activa *Single*, este tiempo no podrá ser inferior a 0,05 segundos. El tiempo correspondiente utilizado para las interrupciones cíclicas es de 0,25 segundos.

Interrupción

Tipo de dato: *intnum*

Es la variable (identidad de la interrupción) de la interrupción. Esta deberá haber sido conectada previamente a una rutina de tratamiento de interrupciones mediante la instrucción *CONNECT*.

Ejecución del programa

La rutina de tratamiento de interrupciones correspondiente será llamada automáticamente en un momento preciso según la orden de interrupción. Cuando esto haya sido ejecutado, proseguirá la ejecución del programa a partir de donde se produjo la interrupción.

Si la interrupción se genera cíclicamente, se iniciará un nuevo cómputo del tiempo a partir de cuando se produce la interrupción.

Ejemplo

```
VAR intnum intemps;  
CONNECT intemps WITH compr_serie;  
ITimer 60, intemps;  
  
TRAP compr_serie  
    WriteBin canal1, buffer, 1;  
    IF ReadBin(canal1\Time:=5) < 0 THEN  
        TPWrite "La comunicación serie falla";  
        EXIT;  
    ENDIF  
ENDTRAP
```

La comunicación a través del canal serie canal1 está monitorizada mediante las interrupciones que son generadas cada 60 segundos. La rutina de tratamiento de interrupciones comprueba que la comunicación funciona. De lo contrario, la ejecución del programa será interrumpida y aparecerá un mensaje de error.

Limitaciones

La misma variable para la identidad de la interrupción no podrá ser utilizada más que una vez, sin ser borrada primero. Véase el capítulo referente a Instrucciones - *ISignalDI*.

Sintaxis

```
ITimer  
[ '\'Single ',' ]  
[ Tiempo ':=' ] < expresión (IN) de num > ','  
[ Interrupción ':=' ] < variable (VAR) de intnum > ';'
```

Información relacionada

Resumen de interrupciones
Más información sobre la gestión
de las interrupciones

Descripción:

Resumen RAPID - *Interrupciones*
Características Básicas - *Interrupcio-
nes*

IVarValue**Ordena una interrupción
de un valor de variable**

*IvarValue (Interrupt Variable Value)*sirve para ordenar y habilitar una interrupción cuando el valor de una variable a la que se ha accedido mediante el interface de sensor serie, ha sido cambiado.

Esta instrucción puede utilizarse, por ejemplo, para obtener el volumen de costura o valores de apertura gracias a un sensor de costura.

Ejemplos

```

LOCAL PERS num adtVlt{25}:=[1,1.2,1.4,1.6,1.8,2,2.16667,2.33333,2.5,...];
LOCAL PERS num adptWfd{25}:=[2,2.2,2.4,2.6,2.8,3,3.16667,3.33333,3.5,...];
LOCAL PERS num adptSpd{25}:=10,12,14,16,18,20,21.6667,23.3333,25[...];
LOCAL CONST num GAP_VARIABLE_NO:=11;
PERS num gap_value;
VAR intnum IntAdap;

PROC main()
! Activar la interrupción. La rutina de tratamiento de interrupciones AdapTrp será
! llamada cuando la variable gap con el número 'GAP_VARIABLE_NO' en
! el interface de sensor haya sido cambiada. El valor nuevo estará disponible
! en el valor de la variable PERS gp_value.
CONNECT IntAdap WITH AdapTrp;
IVarValue GAP_VARIABLE_NO, gap_value, IntAdap;

! Iniciar la soldadura
ArcL\On,*,v100,adaptSm,adaptWd,adaptWv,z10,tool\j\Track:=track;
ArcL\On,*,v100,adaptSm,adaptWd,adaptWv,z10,tool\j\Track:=track;
ENDPROC

TRAP AdapTrap
VAR num ArrInd;
```

```
! Escalar el valor gap recibido
ArrInd:=ArrIdx(gap_value);

! Actualizar la variable PERS de datos de soldadura activa ‘adaptWd’ con
! datos nuevos procedentes de las matrices de los parámetros predefinidos
! El valor gap escalado es utilizado como índice en las matrices de tensión,
! alimentación de hilo y velocidad.
adaptWd.weld_voltage:=adptVlt{ArrInd};
adaptWd.weld_wirefeed:=adptWfd{ArrInd};
adaptWd.weld_speed:=adptSpd{ArrInd};

!Petición de renovación de los parámetros AW utilizando los datos nuevos
!inadaptWd
ArcRefresh;
ENDTRAP
```

Argumentos

IVarValue N°Var Valor, Interrupción

N°VarTipo de dato: *num*

Especifica el número de la variable que debe ser supervisada.

ValorTipo de dato: *num*

Una variable PERS que contendrá el nuevo valor de N°Var.

InterrupciónTipo de dato: *intnum*

Es la variable (identidad de la interrupción) de la interrupción. Esta deberá haber sido conectada previamente a una rutina de tratamiento de interrupciones mediante la instrucción *CONNECT*.

Ejecución del programa

La rutina de tratamiento de interrupciones correspondiente será llamada automáticamente en un momento preciso según la orden de interrupción. Cuando esto haya sido ejecutado, proseguirá la ejecución del programa a partir de donde se produjo la interrupción.

Limitaciones

La misma variable para la identidad de interrupción no podrá ser utilizada más de cinco veces, sin ser borrada en primer lugar.

Sintaxis

IVarValue

[N°Var ':='] < expresión (**IN**) de *num* > ','
[Valor ':='] < persistente (**PERS**) de *num* > ','
[Interrupción ':='] < variable (**VAR**) de *intnum* > ';'

Información relacionada

Resumen de interrupciones
Más información sobre la gestión
de las interrupciones

Descripción:

Resumen RAPID - *Interrupciones*
Características Básicas - *Interrupcio-
nes*

IWatch Habilitación de una interrupción

IWatch (Interrupt Watch) sirve para habilitar una interrupción que ha sido previamente ordenada pero que había sido desactivada mediante la instrucción *ISleep*.

Ejemplo

```
IWatch señalint1;
```

La interrupción *señalint1*, que había sido previamente desactivada, será activada.

Argumentos

IWatch Interrupción

Interrupción	Tipo de dato: <i>intnum</i>
---------------------	-----------------------------

La variable (identidad de la interrupción) de la interrupción.

Ejecución del programa

El evento conectado a esta interrupción genera nuevas interrupciones. Las que han sido generadas mientras la instrucción *ISleep* estaba activada, serán ignoradas.

Ejemplo

```
VAR intnum señalint1;
CONNECT señalint1 WITH rutinaint1;
ISignalDI di1,1,señalint1;

.
ISleep señalint1;
soldar1;
IWatch señalint1;
```

Durante la ejecución de la rutina *soldar1*, no se permitirá ninguna interrupción procedente de la señal *di1*.

Gestión de errores

Las interrupciones que no han sido ordenadas no serán permitidas. En el caso en que el número de interrupción sea desconocido, la variable del sistema **ERRNO** se activará en **ERR_UNKINO** (véase “Tipos de Datos - *errnum*”). El error podrá ser manipulado por el gestor de errores.

Sintaxis

IWatch

[Interrupción ‘:=’] < variable (**VAR**) de *intnum* > ‘;’

Información relacionada

Descripción:

Resumen de interrupciones

Resumen RAPID - *Interrupciones*

Desactivación de una interrupción

Instrucciones - *ISleep*

label**Nombre de línea**

Label sirve para dar nombre a una línea del programa. Utilizando la instrucción *GOTO*, este nombre podrá utilizarse para desplazar la ejecución del programa a esa línea.

Ejemplo

```
GOTO sig;
```

```
.
```

sig:

La ejecución del programa continúa con la instrucción que viene después de *sig*.

Argumentos**Label:**

Label

Identificador

El nombre que se desea dar a la línea.

Ejecución del programa

No pasa nada cuando se ejecuta esta instrucción.

Limitaciones

La etiqueta no debe ser nunca la misma que

- otra etiqueta dentro de la misma rutina,
- ningún nombre de dato dentro de la misma rutina.

Una etiqueta esconde datos globales y rutinas con el mismo nombre dentro de la rutina en que está localizada.

Sintaxis

(EBNF)
<identificador>':'

Información Relacionada

Identificadores

Mover la ejecución de un programa
a una etiqueta

Descripción:

Características Básicas -
Elementos Básicos

Instrucciones - *GOTO*

Load

Cargar un módulo de programa durante la ejecución

Load sirve para cargar un módulo de programa en la memoria de programa durante la ejecución.

El módulo de programa cargado será añadido a los módulos ya existentes de la memoria de programa.

Ejemplo

```
Load ram1disk \File:="PART_A.MOD";
```

Cargar el módulo de programa *PART_A.MOD* a partir de *ram1disk* a la memoria de programa. (*ram1disk* es una constante de cadena predefinida "ram1disk:").

Argumentos

Load TrayectoriadeArchivo [\Archivo]

TrayectoriadeArchivoTipo de dato: *string*

Es la trayectoria de archivo y el nombre del archivo que será cargado en la memoria de programa. El nombre de archivo deberá ser excluido cuando se utiliza el argumento *\Archivo*.

[\Archivo]Tipo de dato: *string*

Cuando el nombre del archivo es excluido del argumento *TrayectoriadeArchivo* entonces deberá ser definido con este argumento.

Ejecución del programa

La ejecución del programa espera que el módulo del programa haya acabado la secuencia de carga antes de proseguir con la siguiente instrucción.

Para obtener una buena estructura de programa, que sea fácil de entender y de mantener, todas las secuencias de carga y descarga de módulos de programa deberán realizarse desde el módulo principal que está siempre presente en la memoria del programa durante la ejecución.

Una vez que el programa ha sido cargado será enlazado e inicializado. La inicialización del módulo cargado tiene por efecto poner todas las variables al nivel del módulo a sus valores iniciales. Las referencias no resueltas serán aceptadas si el parámetro del sistema para *Load* ha sido activado (*BindRef* = NO). No obstante, cuando el programa es

arrancado, la función de la unidad de programación Programa/Archivo/Comprobar no comprobará las referencias no resueltas si el parámetro *BindRef*= NO. Se producirá un error de funcionamiento en el momento de la ejecución de una referencia no resuelta.

Ejemplos

Load "ram1disk:DOORDIR/DOOR1.MOD";

Cargar el módulo de programa *DOOR1.MOD* a partir de *ram1disk* al directorio *DOORDIR* de la memoria de programa.

Load "ram1disk:DOORDIR" \File:="DOOR1.MOD";

Igual que el ejemplo anterior pero con una sintaxis diferente.

Limitaciones

No se permitirá cargar un módulo de programa que contenga una rutina principal.

Las rutinas de tratamiento de interrupciones, los eventos de E/S del sistema y otras tareas de programas no podrán ser ejecutadas durante la secuencia de carga.

Se deberá evitar movimientos del robot durante la secuencia de carga.

Se deberá evitar utilizar el disco flexible para la secuencia de carga dado que la unidad de disco flexible tarda más tiempo.

Un paro de programa durante la ejecución de la instrucción *Load* provocará un paro de protección en los motores y la aparición del siguiente mensaje de error "20025 Stop order timeout" en el visualizador de la unidad de programación.

Gestión de errores

En el caso en que el archivo especificado en las instrucciones *Load* no se encuentre, entonces la variable del sistema *ERRNO* se activará en *ERR_FILENO*. En el caso en que el módulo esté ya cargado en la memoria de programa, entonces la variable del sistema *ERRNO* pasará a ser *ERR_LOADED* (véase "Tipos de Datos - errnum"). Estos errores podrán ser manipulados por el gestor de errores.

Sintaxis

```
Load
[FilePath':=']<expresión (IN) de string>
['\File':= '<expresión (IN) de string>'];'
```

Información relacionada

Descripción:

Descargar un módulo de programa
Aceptar referencias no resueltas

Instrucciones - *UnLoad*
Parámetros del Sistema - *Controller*
Parámetros del Sistema - *Tasks*
Parámetros del Sistema - *BindRef*

Load

Instrucciones

MoveAbsJ**Movimiento del robot a una posición de ejes absoluta**

MoveAbsJ (Move Absolute Joint) sirve para mover el robot a una posición de eje absoluta, definida en las posiciones de los ejes.

Se deberá utilizar esta instrucción únicamente en los siguientes casos:

- cuando el punto final es un punto singular
- para posiciones ambiguas en el IRB 6400C, por ejemplo, para movimientos realizados con la herramienta por encima del robot.

La posición final del robot, durante un movimiento realizado con *MoveAbsJ*, no se verá afectada por la herramienta dada y el objeto de trabajo determinados ni por un desplazamiento del programa activo. No obstante, el robot utiliza estos datos para calcular la carga, la velocidad del TCP, y la trayectoria esquina. Se podrán utilizar las mismas herramientas que las utilizadas en instrucciones de movimiento adyacentes.

El robot y los ejes externos se mueven a la posición de destino siguiendo una trayectoria que no es lineal. Todos los ejes alcanzan la posición de destino al mismo tiempo.

Ejemplos

MoveAbsJ p50, v1000, z50, herramienta2;

El robot con la herramienta *herramienta2*, se moverá siguiendo una trayectoria que no es lineal, a la posición de eje absoluta *p50*, con un dato de velocidad de *v1000* y un dato de zona de *z50*.

MoveAbsJ *, v1000\T:=5, fine, pinza3;

El robot con la herramienta *pinza3*, se moverá siguiendo una trayectoria que no es lineal, a un punto de paro que es almacenado como una posición de ejes absoluta en la instrucción (marcado con un asterisco *). El movimiento completo dura unos 5 s.

Argumentos

MoveAbsJ [\Conc] A Pos Eje Velocidad [\V] | [\T] Zona [\Z] Herramienta [\WObj]

[\Conc]

(Concurrente)

Tipo de dato: *switch*

Las instrucciones lógicas siguientes se ejecutarán mientras el robot está en movimiento. El argumento sirve para acortar el tiempo de ciclo, cuando, por ejemplo, se realiza una comunicación con el equipo externo, siempre y cuando no se esté utilizando la sincronización.

Al utilizar el argumento \Conc, el número de instrucciones de movimiento que se suceden está limitado a 5. En una sección de programa que incluye *StorePath-RestoPath*, no se permitirán instrucciones de movimiento que contengan el argumento \Conc.

En el caso de omitir este argumento, la instrucción siguiente se ejecutará únicamente después de que el robot haya alcanzado la zona especificada.

APosEje	(<i>A Posición Eje</i>)	Tipo de dato: <i>jointtarget</i>
----------------	---------------------------	----------------------------------

Es la posición eje absoluta de destino del robot y de los ejes externos. Es definida como una posición nombrada o almacenada directamente en la instrucción (marcada con un asterisco * en la instrucción).

Velocidad		Tipo de dato: <i>speeddata</i>
------------------	--	--------------------------------

Son los datos de velocidad que se aplican a los movimientos. Los datos de velocidad definen la velocidad del punto central de la herramienta, la reorientación de la herramienta y los ejes externos.

[\V]	(<i>Velocidad</i>)	Tipo de dato: <i>num</i>
---------------	----------------------	--------------------------

Este argumento sirve para especificar la velocidad del TCP en mm/s directamente en la instrucción. Luego, será sustituido por la velocidad correspondiente especificada en los datos de velocidad.

[\T]	(<i>Tiempo</i>)	Tipo de dato: <i>num</i>
---------------	-------------------	--------------------------

Este argumento sirve para especificar el tiempo total expresado en segundos durante el cual el robot se mueve. Luego será sustituido por los datos de velocidad correspondientes.

Zone		Tipo de dato: <i>zonedata</i>
-------------	--	-------------------------------

Datos de zona para el movimiento. Los datos de zona describen el tamaño de la trayectoria esquina generada.

[\Z]	(<i>Zona</i>)	Tipo de dato: <i>num</i>
---------------	-----------------	--------------------------

Este argumento sirve para especificar la precisión de la posición del TCP del robot directamente en la instrucción. La longitud de la trayectoria esquina está expresada en mm., y será sustituida por la zona correspondiente especificada en los datos de zona.

Tool		Tipo de dato: <i>tooldata</i>
-------------	--	-------------------------------

La herramienta utilizada durante el movimiento.

La posición del TCP y la carga en la herramienta están definidos en los datos de herramienta. La posición del TCP sirve para decidir la velocidad y la trayectoria esquina del movimiento.

[\WObj]

(Objeto de trabajo)

Tipo de dato: wobjdata

Es el objeto de trabajo utilizado durante el movimiento.

Este argumento podrá ser omitido si la herramienta es sujetada por el robot. No obstante, si el robot sujeta el objeto de trabajo, es decir, si la herramienta es estacionaria, o con ejes externos coordinados, entonces se deberá especificar este argumento.

En el caso de una herramienta estacionaria o de ejes externos coordinados, los datos utilizados por el sistema para decidir la velocidad y la trayectoria esquina del movimiento, se encuentran definidos en el objeto de trabajo.

Ejecución del programa

La herramienta se moverá a la posición eje absoluta de destino, con una interpolación de los ángulos de los ejes. Esto significa que cada eje se mueve a una velocidad de ejes constante y que todos los ejes alcanzan la posición eje de destino al mismo tiempo, lo que resulta en una trayectoria que no es lineal.

De forma general, el TCP se mueve a la velocidad programada aproximada. La herramienta es reorientada y los ejes externos se mueven al mismo tiempo que el TCP. En el caso en que el sistema no sea capaz de alcanzar la velocidad programada para la reorientación o para los ejes externos, la velocidad del TCP será reducida.

Una trayectoria esquina suele ser generada cuando un movimiento es transferido a la siguiente sección de la trayectoria. En el caso en que se haya especificado un punto de paro en los datos de zona, la ejecución del programa sólo continuará cuando el robot y los ejes externos hayan alcanzado la posición de eje aproximada.

Ejemplos

```
MoveAbsJ *, v2000\Y:=2200, z40 \Z:=45, pinza3;
```

La herramienta, *pinza3*, se moverá siguiendo una trayectoria que no es lineal a una posición de eje absoluta almacenada en la instrucción. El movimiento se llevará a cabo con los datos activos en *v2000* y *z40*, la velocidad y el tamaño de zona del TCP son 2200 mm/s y 45 mm respectivamente.

```
MoveAbsJ \Conc, *, v2000, z40, pinza3;
```

La herramienta, *pinza3*, se moverá siguiendo una trayectoria que no es lineal a una posición de eje absoluta almacenada en la instrucción. Las instrucciones lógicas siguientes son ejecutadas mientras el robot se está moviendo.

```
GripLoad obj_mass;
MoveAbsJ start, v2000, z40, grip3 \WObj:= obj;
```

El robot mueve el objeto de trabajo *obj*, respecto a la herramienta fija *pinza3* siguiendo una trayectoria que no es lineal, a una posición de eje absoluta de arranque (*start*).

Gestión de errores

Cuando se va a ejecutar el programa, el sistema realiza una comprobación de que los argumentos Herram y \WObj no contienen datos contradictorios respecto a una herramienta móvil o estacionaria respectivamente.

Limitaciones

Un movimiento realizado con la instrucción *MoveAbsJ* no se verá afectado por un desplazamiento activo del programa, sin embargo se verá afectado por un offset activo para los ejes externos.

Para ser capaz de retroceder la ejecución en una situación en que la instrucción *MoveAbsJ* esté implicada, y para evitar problemas con puntos singulares o áreas ambiguas, es esencial que las instrucciones siguientes cumplan ciertos requisitos, según se indica en la figura siguiente (véase la Figura 1).

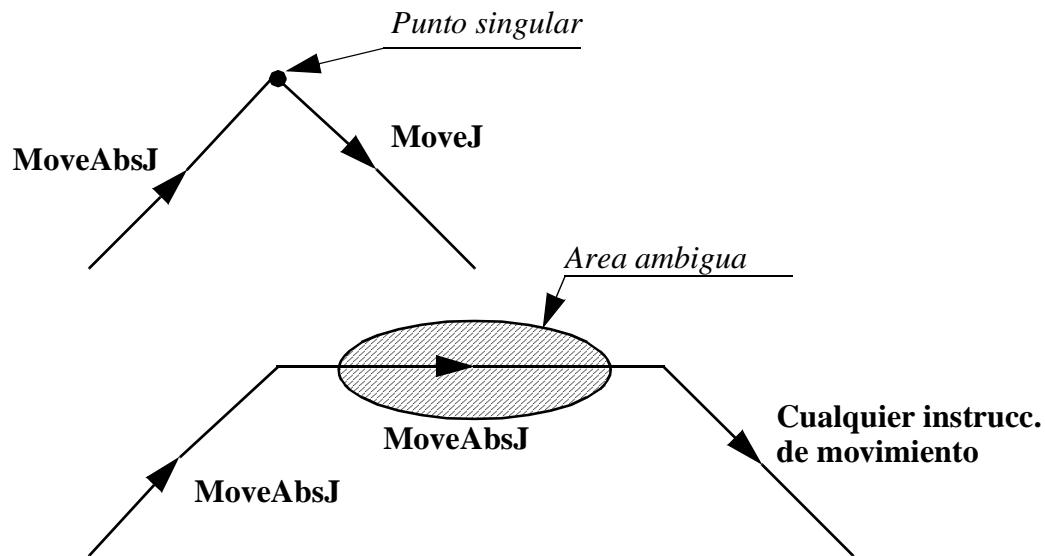


Figura 1 Limitación para la ejecución hacia atrás en un movimiento con la instrucción *MoveAbsJ*.

Sintaxis

```

MoveAbsJ
[ '\' Conc ',' ]
[ ToJointPos ':=' ] < expresión (IN) de jointtarget > ',' 
[ Speed ':=' ] < expresión (IN) de speeddata >
    [ '\' V ':=' < expresión (IN) de num > ]
    | [ '\' T ':=' < expresión (IN) de num > ] ',' 
[Zone ':=' ] < expresión (IN) de zonedata >
    [ '\' Z ':=' < expresión (IN) de num > ] ',' 
[ Tool ':=' ] < expresión (PERS) de tooldata >
[ '\' WObj ':=' < expresión (PERS) de wobjdata > ] ';'
```

Información relacionada

Descripción:

Otras instrucciones de posicionamiento	Resumen RAPID - <i>Movimiento</i>
Definición de jointtarget	Tipos de Datos - <i>jointtarget</i>
Definición de la velocidad	Tipos de Datos - <i>speeddata</i>
Definición de los datos de zona	Tipos de Datos - <i>zonedata</i>
Definición de las herramientas	Tipos de Datos - <i>tooldata</i>
Definición de los objetos de trabajo	Tipos de Datos - <i>wobjdata</i>
Movimiento en general	Principios de movimiento y de E/S
Ejecución concurrente del programa	Principios de movimiento y de E/S - <i>Sincronización utilizando Instrucciones Lógicas</i>

MoveAbsJ

Instrucciones

MoveC**Movimiento circular del robot**

MoveC (Move Circular) sirve para mover el punto central de la herramienta (TCP) de forma circular a un destino específico. Durante el movimiento, la orientación suele permanecer constante respecto al círculo.

Ejemplos

```
MoveC p1, p2, v500, z30, herram2;
```

El TCP de la herramienta, *herram2*, se moverá de forma circular a la posición *p2*, con un dato de velocidad de *v500* y un dato de zona de *z30*. El círculo se define a partir de la posición de arranque, el punto de círculo *p1* y el punto de destino *p2*.

```
MoveC *, *, v500 \T:=5, fine, pinza3;
```

El TCP de la herramienta, *pinza3*, se moverá de forma circular a un punto fino almacenado en la instrucción (marcado por el segundo asterisco *). El punto de círculo también está almacenado en la instrucción (marcado por el primer asterisco *). El movimiento completo tarda 5 segundos en realizarse.

```
MoveL p1, v500, fine, herram1;
MoveC p2, p3, v500, z20, herram1;
MoveC p4, p1, v500, fine, herram1;
```

Para realizar un círculo completo, las posiciones deben ser las mismas que las que se indican en la Figura 1.

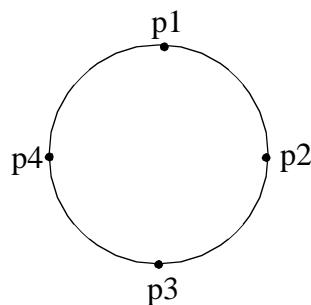


Figura 1 Un círculo completo se realiza mediante dos instrucciones MoveC.

Argumentos

MoveC [\Conc] PuntoCirculo AlPunto Velocidad [\V] | [\T]

Zona [\Z] Herramienta [\WObj] [\Corr]

[\Conc]	<i>(Concurrente)</i>	Tipo de dato: <i>switch</i>
-----------	----------------------	-----------------------------

Las instrucciones lógicas siguientes se ejecutarán inmediatamente. Este argumento sirve para acortar el tiempo de ciclo cuando, por ejemplo, se está comunicando con el equipo externo, si no se requiere ninguna sincronización.

Utilizando el argumento *\Conc*, el número de instrucciones de movimiento que se suceden está limitado a 5. En una sección de programa que incluye *StorePath-RestoPath*, no se permitirá la presencia de instrucciones de movimiento con el argumento *\Conc*.

Si se omite este argumento y que AlPunto no es un punto de paro, la instrucción siguiente se ejecutará poco tiempo antes de que el robot haya alcanzado la zona programada.

PuntoCírculo	Tipo de dato: <i>robtarget</i>
---------------------	--------------------------------

Es el punto de círculo del robot. El punto de círculo es una posición en el círculo, entre el punto de arranque y el punto de destino. Para obtener una mayor precisión, deberá estar situado aproximadamente a mitad de camino entre el punto de arranque y el punto de destino. Si está situado demasiado cerca del punto de arranque o del punto de destino, el robot puede generar un aviso. El punto de círculo está definido como una posición nombrada o es almacenado directamente en la instrucción (marcado con un asterisco * en la instrucción). Las posiciones de los ejes externos no son utilizadas.

AlPunto	Tipo de dato: <i>robtarget</i>
----------------	--------------------------------

Es el punto de destino del robot y de los ejes externos. Está definido como una posición nombrada o es almacenado directamente en la instrucción (marcado con un asterisco * en la instrucción).

Velocidad	Tipo de dato: <i>speeddata</i>
------------------	--------------------------------

Son los datos de velocidad que se aplican a los movimientos. Los datos de velocidad definen la velocidad del TCP, la reorientación de la herramienta y los ejes externos.

[\V]	<i>(Velocidad)</i>	Tipo de dato: <i>num</i>
--------	--------------------	--------------------------

Este argumento sirve para especificar la velocidad del TCP en mm/s directamente en la instrucción. Luego, será substituido por la velocidad correspondiente especificada en los datos de velocidad.

[\T]	<i>(Tiempo)</i>	Tipo de dato: <i>num</i>
--------	-----------------	--------------------------

Este argumento sirve para especificar el tiempo total en segundos durante el cual el robot y los ejes externos se mueven. Luego, será substituido por los datos de velocidad correspondientes.

Zona	Tipo de dato: <i>zonedata</i>
------	-------------------------------

Son los datos de zona para el movimiento. Los datos de zona describen el tamaño de la trayectoria esquina generada.

[\Z]	(Zona)	Tipo de dato: <i>num</i>
--------	--------	--------------------------

Este argumento sirve para especificar la precisión de la posición del TCP del robot directamente en la instrucción. La longitud de la trayectoria esquina está especificada en mm, y será sustituida por la zona correspondiente especificada en los datos de zona.

Herramienta	Tipo de dato: <i>tooldata</i>
-------------	-------------------------------

Es la herramienta en uso cuando el robot se mueve. El TCP es el punto que se mueve al punto de destino específico.

[\WObj]	(Objeto de trabajo)	Tipo de dato: <i>wobjdata</i>
-----------	---------------------	-------------------------------

Es el objeto de trabajo (sistema de coordenadas del objeto) al que se refiere la posición del robot en la instrucción.

Este argumento puede ser omitido, y en el caso en que se omita, la posición se referirá al sistema de coordenadas mundo. Si, por otra parte, se utiliza un TCP estacionario o ejes externos coordinados, este argumento deberá ser especificado para que un círculo pueda ejecutarse respecto a un objeto de trabajo.

[\Corr]	(Corrección)	Tipo de dato: <i>switch</i>
-----------	--------------	-----------------------------

Si este argumento está presente, los datos de corrección introducidos en una entrada de corrección mediante la instrucción *CorrWrite* serán añadidos a la trayectoria y a la posición de destino.

Ejecución del programa

El robot y las unidades externas se moverán al punto de destino de la siguiente manera:

- El TCP de la herramienta se moverá de forma circular a la velocidad constante programada.
- La herramienta es reorientada a una velocidad constante desde la orientación de la posición de arranque a la orientación del punto de destino.
- La reorientación se lleva a cabo respecto a la trayectoria circular. Así, si la orientación respecto a la trayectoria es la misma en el punto de arranque y en el punto de destino, la respectiva orientación permanecerá constante durante el movimiento (véase la Figura 2).

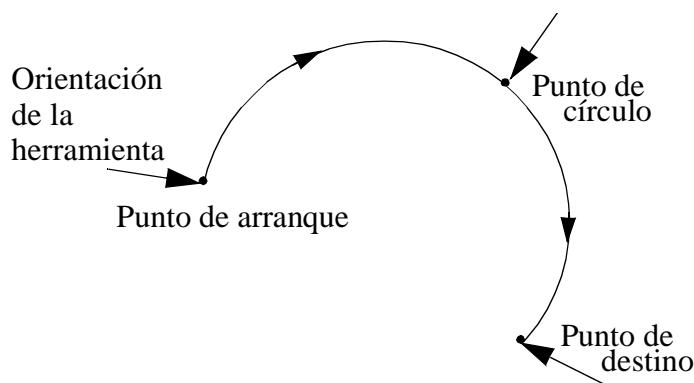


Figura 2 Orientación de la herramienta durante el movimiento circular.

- La orientación en el punto de círculo no es crítica; sirve únicamente para distinguir entre dos direcciones posibles de reorientación. La precisión de la reorientación sobre la trayectoria dependerá únicamente de la orientación en el punto de arranque y en el punto de destino.
- Los ejes externos no coordinados se ejecutan a una velocidad constante para que puedan llegar al punto de destino al mismo tiempo que los ejes del robot. La posición en el círculo no es utilizada.

En el caso en que la reorientación o los ejes externos no puedan alcanzar la velocidad programada, la velocidad del TCP será reducida.

Una trayectoria esquina suele ser generada cuando el movimiento es transferido a la sección siguiente de una trayectoria. Si se especifica un punto de paro en los datos de zona, la ejecución del programa únicamente continuará cuando el robot y los ejes externos hayan alcanzado la posición adecuada.

Ejemplos

```
MoveC *, *, v500 |V:=550, z40 |Z:=45, pinza3;
```

El TCP de la herramienta, *pinza3*, se moverá de forma circular a una posición almacenada en la instrucción. El movimiento se llevará a cabo con unos datos de *v500* y *z40*; la velocidad y el tamaño de la zona del TCP son de 550 mm/s y de 45 mm respectivamente.

MoveC \Conc, *, *, v500, z40, pinza3;

El TCP de la herramienta, *pinza3*, se moverá de forma circular a una posición almacenada en la instrucción. El punto de círculo está también almacenado en la instrucción. Las instrucciones lógicas siguientes se ejecutarán mientras el robot se mueve.

MoveC cir1, p15, v500, z40, pinza3 \WObj:=fijación;

El TCP de la herramienta, *pinza3*, se moverá de forma circular a la posición, *p15*, pasando por el punto de círculo *cir1*. Estas posiciones están especificadas en el sistema de coordenadas del objeto en la *fijación*.

Limitaciones

Un cambio del modo de ejecución, de delante hacia atrás o viceversa, mientras el robot está parado en una trayectoria circular no será permitido y por tanto se originará un mensaje de error.

La instrucción *MoveC* (o cualquier otra instrucción incluyendo un movimiento circular) no deberá nunca ser arrancada desde el principio, con el TCP entre el punto de círculo y el punto final. De lo contrario el robot no emprenderá la trayectoria programada (posicionamiento en torno a la trayectoria circular en otra dirección comparada con la programada).

El usuario deberá asegurarse de que el robot puede alcanzar el punto de círculo durante la ejecución del programa y dividir el segmento del círculo si necesario.

Sintaxis

```
MoveC
[ '\' Conc ',' ]
[ PuntoCírculo':=' ] < expresión (IN) de robtarget > ',' 
[ AlPunto':=' ] < expresión (IN) de robtarget > ',' 
[ Velocidad':=' ] < expresión (IN) de speeddata >
    [ '\' V ':=' < expresión (IN) de num > ]
    | [ '\' T ':=' < expresión (IN) de num > ] ',' 
[ Zona ':=' ] < expresión (IN) de zonedata >
    [ '\' Z ':=' < expresión (IN) de num > ] ',' 
[ Herramienta ':=' ] < persistente (PERS) de tooldata >
[ '\' WObj ':=' < persistente (PERS) de wobjdata > ]
[ '\' Corr ]';'
```

Información relacionada

Otras instrucciones de posicionamiento
Definición de la velocidad
Definición de los datos de zona
Definición de las herramientas
Definición de los objetos de trabajo
Escritura en una entrada de corrección
Movimiento en general
Sistemas de coordenadas

Ejecución de un programa concurrente

Descripción:

Resumen RAPID - *Movimiento*
Tipos de datos - *speeddata*
Tipos de datos - *zonedata*
Tipos de datos - *tooldata*
Tipos de datos - *wobjdata*
Instrucciones - *CorrWrite*
Principios de Movimiento y de E/S
Principios de Movimiento y de E/S -
Sistemas de coordenadas
Principios de Movimiento y de E/S -
*Sincronización utilizando instruccio-
nes lógicas*

MoveJ**Movimiento eje a eje del robot**

MoveJ (*Move Joint*) sirve para mover el robot rápidamente desde un punto a otro punto cuando este movimiento no tiene que seguir una línea recta.

El robot y los ejes externos se moverán a la posición de destino siguiendo una trayectoria que no es lineal. Todos los ejes alcanzarán la posición de destino al mismo tiempo.

Ejemplos

MoveJ p1, vmax, z30, herramienta2;

El punto central de la herramienta (TCP), *herramienta2*, se moverá siguiendo una trayectoria que no es lineal para alcanzar la posición, *p1*, con el dato de velocidad *vmax* y el dato de zona *z30*.

MoveJ *, vmax \T:=5, fine, pinza3;

El TCP de la herramienta, *pinza3*, se moverá siguiendo una trayectoria no lineal a un punto de paro almacenado en la instrucción (marcado con un asterisco *). El movimiento entero tardará 5 segundos en realizarse.

Argumentos

MoveJ [\Conc] AlPunto Velocidad [\V] | [\T] Zona [\Z] Herramienta [\WObj]

[\Conc]	<i>(Concurrente)</i>	Tipo de dato: <i>switch</i>
------------------	----------------------	-----------------------------

Las instrucciones lógicas siguientes se ejecutarán mientras el robot está en movimiento. Este argumento sirve para acortar el tiempo de ciclo cuando, por ejemplo, se está comunicando con el equipo externo, si no se requiere ninguna sincronización.

Utilizando el argumento *\Conc*, el número de instrucciones de movimiento que se suceden está limitado a 5. En una sección de programa que incluye *StorePath-RestorePath*, no se permitirá la presencia de instrucciones de movimiento con el argumento *\Conc*.

Si se omite este argumento, la instrucción siguiente se ejecutará solamente después de que el robot haya alcanzado la zona específica.

AlPunto	Tipo de dato: <i>robtarget</i>
----------------	--------------------------------

Es el punto de destino del robot y de los ejes externos. Está definido como una posición nombrada o es almacenado directamente en la instrucción (marcado con un asterisco * en la instrucción).

VelocidadTipo de dato: *speeddata*

Son los datos de velocidad que se aplican a los movimientos. Los datos de velocidad definen la velocidad del TCP, la reorientación de la herramienta y los ejes externos.

[\V]*(Velocidad)*Tipo de dato: *num*

Este argumento sirve para especificar la velocidad del TCP en mm/s directamente en la instrucción. Luego, será sustituido por la velocidad correspondiente especificada en los datos de velocidad.

[\T]*(Tiempo)*Tipo de dato: *num*

Este argumento sirve para especificar el tiempo total en segundos durante el cual el robot se mueve. Luego, será sustituido por los datos de velocidad correspondientes.

ZonaTipo de dato: *zonedata*

Son los datos de zona para el movimiento. Los datos de zona describen el tamaño de la trayectoria esquina generada.

[\Z]*(Zona)*Tipo de dato: *num*

Este argumento sirve para especificar la precisión de la posición del TCP del robot directamente en la instrucción. La longitud de la trayectoria esquina está especificada en mm, y será sustituida por la zona correspondiente especificada en los datos de zona.

HerramientaTipo de dato: *tooldata*

Es la herramienta en uso cuando el robot se mueve. El TCP es el punto que se mueve al punto de destino específico.

[\WObj]*(Objeto de trabajo)*Tipo de dato: *wobjdata*

Es el objeto de trabajo (sistema de coordenadas) al que se refiere la posición del robot en la instrucción.

Este argumento puede omitirse, y en el caso en que se omita, la posición se referirá al sistema de coordenadas mundo. Si, por otra parte, se utiliza un TCP estacionario o ejes externos coordinados, este argumento deberá ser especificado.

Ejecución del programa

El punto central de la herramienta se moverá al punto de destino con una interpolación de los ángulos de los ejes. Ello significa que cada eje se moverá a una velocidad de eje constante y que todos los ejes alcanzarán el punto de destino al mismo tiempo, lo cual originará una trayectoria que no es lineal.

De forma general, el TCP se moverá a la velocidad programada aproximada (independientemente de la velocidad de ejecución).

dientemente de si los ejes externos están coordinados o no). La herramienta será reorientada y los ejes externos se moverán al mismo tiempo que se mueve el TCP. En el caso en que la reorientación o los ejes externos no puedan alcanzar la velocidad programada, la velocidad del TCP será reducida.

Una trayectoria esquina suele ser generada cuando el movimiento es transferido a la sección siguiente de una trayectoria. Si se especifica un punto de paro en los datos de zona, la ejecución del programa únicamente continuará cuando el robot y los ejes externos hayan alcanzado la posición adecuada.

Ejemplos

MoveJ *, v2000\|V:=2200, z40 \|Z:=45, pinza3;

El TCP de la herramienta, *pinza3*, se moverá siguiendo una trayectoria que no es lineal, a una posición almacenada en la instrucción. El movimiento se llevará a cabo con los datos de *v2000* y *z40*; la velocidad y el tamaño de la zona del TCP son de 2200 mm/s y de 45 mm respectivamente.

MoveJ \Conc, *, v2000, z40, pinza3;

El TCP de la herramienta, *pinza3*, se moverá siguiendo una trayectoria que no es lineal, a una posición almacenada en la instrucción. Las instrucciones lógicas siguientes se ejecutarán mientras el robot se mueve.

MoveJ arranque, v2000, z40, pinza3 \|WObj:=fijación;

El TCP de la herramienta, *pinza3*, se moverá siguiendo una trayectoria que no es lineal, a una posición, *arranque*. Esta posición está especificada en el sistema de coordenadas del objeto, en la *fijación*.

Sintaxis

MoveJ

```
[ '\' Conc ',' ]
[ AlPunto'=' ] < expresión (IN) de robtarget > ',' 
[ Velocidad'=' ] < expresión (IN) de speeddata >
    [ '\' V ':=> < expresión (IN) de num > ]
    | [ '\' T ':=> < expresión (IN) de num > ] ',' 
[ Zona ':=> ] < expresión (IN) de zonedata >
    [ '\' Z ':=> < expresión (IN) de num > ] ',' 
[ Herramienta'=' ] < persistente (PERS) de tooldata >
[ '\' WObj ':=> < persistente (PERS) de wobjdata > ] ','
```

Información relacionada

Otras instrucciones de posicionamiento
Definición de la velocidad
Definición de los datos de zona
Definición de las herramientas
Definición de los objetos de trabajo
Movimiento en general
Sistemas de coordenadas

Ejecución de un programa concurrente

Descripción:

Resumen RAPID - *Movimiento*
Tipos de datos - *speeddata*
Tipos de datos - *zonedata*
Tipos de datos - *tooldata*
Tipos de datos - *wobjdata*
Principios de Movimiento y de E/S
Principios de Movimiento y de E/S-
Sistemas de Coordenadas
Principios de Movimiento y de E/S -
*Sincronización utilizando Instruccio-
nes Lógicas*

MoveL**Movimiento lineal del robot**

MoveL sirve para mover el punto central de la herramienta (TCP) de forma lineal a una posición de destino determinada. Cuando el TCP debe permanecer estacionario, esta instrucción podrá ser utilizada también para reorientar la herramienta.

Ejemplo

MoveL p1, v1000, z30, herramienta2;

El TCP de la herramienta, *herramienta2*, se moverá de forma lineal a la posición *p1*, con un dato de velocidad de *v1000* y un dato de zona de *z30*.

MoveL *, v1000\T:=5, fine, pinza3;

El TCP de la herramienta, *pinza3*, se moverá de forma lineal a un punto fino almacenado en la instrucción (marcado con un asterisco *). El movimiento completo tardará 5 segundos en llevarse a cabo.

Argumentos

**MoveL [\Conc] AlPunto Velocidad [\V] | [\T] Zona [\Z]
Herram [\WObj] [\Corr]**

[\Conc]	<i>(Concurrente)</i>	Tipo de dato: <i>switch</i>
------------------	----------------------	------------------------------------

Las instrucciones lógicas siguientes se ejecutarán inmediatamente. Este argumento sirve para acortar el tiempo de ciclo cuando, por ejemplo, se está comunicando con el equipo externo, si no se requiere ninguna sincronización.

Utilizando el argumento *\Conc*, el número de instrucciones de movimiento que se suceden está limitado a 5. En una sección de programa que incluye *StorePath-RestorePath*, no se permitirá la presencia de instrucciones de movimiento con el argumento *\Conc*.

Si se omite este argumento y que *AlPunto* no es un punto de paro, la instrucción siguiente se ejecutará poco tiempo antes de que el robot haya alcanzado la zona específica.

AlPunto

Tipo de dato: *robtarget*

Es el punto de destino del robot y de los ejes externos. Está definido como una posición nombrada o es almacenado directamente en la instrucción (marcado con un asterisco * en la instrucción).

Velocidad

Tipo de dato: *speeddata*

Son los datos de velocidad que se aplican a los movimientos. Los datos de velocidad definen la velocidad del TCP, la reorientación de la herramienta y los ejes

externos.

[\V]	(Velocidad)	Tipo de dato: num
--------	-------------	-------------------

Este argumento sirve para especificar la velocidad del TCP en mm/s directamente en la instrucción. Luego, será sustituido por la velocidad correspondiente especificada en los datos de velocidad.

[\T]	(Tiempo)	Tipo de dato: num
--------	----------	-------------------

Este argumento sirve para especificar el tiempo total en segundos durante el cual el robot se mueve. Luego, será sustituido por los datos de velocidad correspondientes.

Zona		Tipo de dato: zonedata
-------------	--	------------------------

Son los datos de zona para el movimiento. Los datos de zona describen el tamaño de la trayectoria esquina generada.

[\Z]	(Zona)	Tipo de dato: num
--------	--------	-------------------

Este argumento sirve para especificar la precisión de la posición del TCP del robot directamente en la instrucción. La longitud de la trayectoria esquina está especificada en mm, y será sustituida por la zona correspondiente especificada en los datos de zona.

Herramienta		Tipo de dato: tooldata
--------------------	--	------------------------

Es la herramienta en uso cuando el robot se mueve. El TCP es el punto que se mueve al punto de destino específico.

[\WObj]	(Objeto de trabajo)	Tipo de dato: wobjdata
-----------	---------------------	------------------------

El objeto de trabajo (sistema de coordenadas) al que se refiere la posición del robot en la instrucción.

Este argumento puede omitirse, y en el caso en que se omita, la posición se referirá al sistema de coordenadas mundo. Si, por otra parte, se utiliza un TCP estacionario o ejes externos coordinados, este argumento deberá ser especificado para poder realizar un movimiento lineal respecto al objeto de trabajo.

[\Corr]	(Corrección)	Tipo de dato: switch
-----------	--------------	----------------------

Si este argumento está presente, los datos de corrección introducidos en una entrada de corrección mediante la instrucción *CorrWrite* serán añadidos a la trayectoria y a la posición de destino.

Ejecución del programa

El robot y las unidades externas se moverán al punto de destino de la siguiente manera:

- El TCP de la herramienta se moverá de forma lineal a una velocidad programada constante.
- La herramienta es reorientada a intervalos iguales sobre la trayectoria.
- Los ejes externos no coordinados se moverán a una velocidad constante para que puedan llegar al punto de destino al mismo tiempo que los ejes del robot.

En el caso en que la reorientación o los ejes externos no puedan alcanzar la velocidad programada, la velocidad del TCP será reducida.

Una trayectoria esquina suele ser generada cuando el movimiento es transferido a la sección siguiente de una trayectoria. Si se especifica un punto de paro en los datos de zona, la ejecución del programa únicamente continuará cuando el robot y los ejes externos hayan alcanzado la posición adecuada.

Ejemplos

MoveL *, v2000 \V:=2200, z40 \Z:=45, pinza3;

El TCP de la herramienta, *pinza3*, se moverá siguiendo una trayectoria lineal, a una posición almacenada en la instrucción. El movimiento se llevará a cabo con los datos de *v2000* y *z40*; la velocidad y el tamaño de la zona del TCP son de 2200 mm/s y de 45 mm respectivamente.

MoveJ \Conc, *, v2000, z40, pinza3;

El TCP de la herramienta, *pinza3*, se moverá siguiendo una trayectoria lineal, a una posición almacenada en la instrucción. Las instrucciones lógicas siguientes se ejecutarán mientras el robot se mueve.

MoveJ arranque, v2000, z40, pinza3 \WObj:=fijación;

El TCP de la herramienta, *pinza3*, se moverá siguiendo una trayectoria lineal, a una posición, *arranque*. Esta posición está especificada en el sistema de coordenadas del objeto, en la *fijación*.

Sintaxis

MoveL

```
[ '\' Conc ',' ]  
[ AlPunto':=' ] < expresión (IN) de robtarget > ','  
[ Velocidad':=' ] < expresión (IN) de speeddata >  
    [ '\' V ':=' < expresión (IN) de num > ]  
    | [ '\' T ':=' < expresión (IN) de num > ] ','  
[ Zona ':=' ] < expresión (IN) de zonedata >  
    [ '\' Z ':=' < expresión (IN) de num > ] ','  
[ Herramienta ':=' ] < persistente (PERS) de tooldata >  
[ '\' WObj ':=' < persistente (PERS) de wobjdata > ]  
[ '\' Corr ]';'
```

Información relacionada

Otras instrucciones de posicionamiento
Definición de la velocidad
Definición de los datos de zona
Definición de las herramientas
Definición de los objetos de trabajo
Escritura en una entrada de corrección
Movimiento en general
Sistemas de coordenadas

Ejecución de un programa concurrente

Descripción:

Resumen RAPID - *Movimiento*
Tipos de datos - *speeddata*
Tipos de datos - *zonedata*
Tipos de datos - *tooldata*
Tipos de datos - *wobjdata*
Instrucciones - *CorrWrite*
Principios de Movimiento y de E/S
Principios de Movimiento y de E/S-
Sistemas de Coordenadas

Principios de Movimiento y de E/S -
Sincronización utilizando instruccio-
nnes lógicas

Open Apertura de un archivo o de un canal serie

Open sirve para abrir un archivo o un canal serie para la lectura o la escritura.

Ejemplo

```
VAR iodev archivodat;  
.  
Open "fpl1:LOGDIR" \File:= "ARCHIVO1.DOC",archivodat;
```

El archivo *ARCHIVO1.DOC* de la unidad *fpl1*: (disquete), directorio *LOGDIR*, está abierto para la escritura. El nombre de referencia *archivodat* se utilizará posteriormente en el programa cuando se escriba en el archivo.

Argumentos

Open Objeto [\File] Dispositivo E/S [\Read] | [\Write] | [\Append] | [\Bin]

Objeto Tipo de dato: *string*

El objeto de E/S que se desea abrir, por ejemplo, "fpl1:", "ram disk:".

[\File] Tipo de dato: *string*

El nombre del archivo. Este nombre puede también ser especificado en el argumento *Objeto*, por ejemplo, "fpl1:LOGDIR/ARCHIVO.DOC".

Dispositivo E/S Tipo de dato: *iodev*

Una referencia al archivo o al canal serie que se desea abrir. Esta referencia se utiliza luego para las operaciones de lectura y escritura en el archivo/canal.

Los argumentos *\Read*, *\Write*, *\Append* y *\Bin* son mutuamente exclusivos. En el caso en que ninguno de éstos haya sido especificado, la instrucción actuará en el mismo sentido que el argumento *\Write*.

[\Read] Tipo de dato: *switch*

Abre un archivo basado en caracteres o un canal serie para la lectura. Al leer un archivo, la lectura empieza desde el principio del archivo.

[\Write] Tipo de dato: *switch*

Abre un archivo basado en caracteres o un canal serie para la escritura. En el caso en que el archivo seleccionado ya exista, su contenido será borrado. Cualquier cosa que se escriba a partir de entonces, se escribirá al principio del archivo.

[Append]Tipo de dato: *switch*

Abre un archivo basado en caracteres o un canal serie para la escritura. En el caso en que el archivo seleccionado ya exista, todo lo que se escribirá a partir de entonces se escribirá al final del archivo.

[Bin]Tipo de dato: *switch*

Abre un canal serie binario para la lectura y escritura.
Funciona como *append*, es decir, indicador de archivo al final del archivo.

Ejemplo

```
VAR iodev impresora;
```

```
. Open "sio1:", impresora \Bin;
Write impresora, "Este es un mensaje para la impresora";
Close impresora;
```

El canal serie *sio1:* es abierto para la lectura y escritura en binario. El nombre de referencia *impresora* será utilizado posteriormente al escribir y al cerrar el canal serie.

Ejecución del programa

El canal serie/archivo especificado será activado de forma que se pueda leer o escribir en él. Se podrán abrir diferentes archivos en la misma unidad al mismo tiempo.

Gestión de errores

En el caso en que no se pueda abrir un archivo, la variable del sistema ERRNO se convertirá en ERR_FILEOPEN. Este error podrá ser procesado posteriormente en el gestor de errores.

Sintaxis

Open

```
[Objeto ':='] <expresión (IN) de string>
['\File':=' <expresión (IN) de string>'] ',' 
[Dispositivo E/S':=' <variable (VAR) de iodev>
['\Read] | ['\Write] | ['\Append] | ['\Bin] ';
```

Información relacionada

Descripción:

Lectura y escritura de
canales serie y archivos

Resumen RAPID - *Comunicación*

Open

Instrucciones

PDispOff**Desactivación de un desplazamiento
de programa**

PDispOff (*Program Displacement Off*) sirve para desactivar un desplazamiento de programa.

El programa de desplazamiento será activado por la instrucción *PDispSet* o *PDispOn* y se aplica a todos los movimientos hasta que se active o desactive otro desplazamiento de programa.

Ejemplos

PDispOff;

Desactivación de un desplazamiento de programa.

```
MoveL p10, v500, z10, herram1;  
PDispOn \ExeP:=p10, p11, herram1;  
MoveL p20, v500, z10, herram1;  
MoveL p30, v500, z10, herram1;  
PDispOff;  
MoveL p40, v500, z10, herram1;
```

Un desplazamiento de programa será definido como la diferencia entre las posiciones *p10* y *p11*. Este desplazamiento afectará el movimiento a *p20* y *p30*, pero no a *p40*.

Ejecución del programa

Se reseteará el desplazamiento de programa activo. Esto significa que el sistema de coordenadas del desplazamiento del programa será el mismo que el sistema de coordenadas del objeto y así todas las posiciones programadas se referirán a éste último.

Sintaxis

PDispOff ‘;’

Información relacionada

Definición de un desplazamiento de programa utilizando dos posiciones
Definición de un desplazamiento de programa utilizando valores

Descripción:

Instrucciones - *PDispOn*

Instrucciones - *PDispSet*

PDispOn**Activación de un desplazamiento
de programa**

PDispOn (Program Displacement On) sirve para definir y activar un desplazamiento de programa utilizando dos posiciones de robot.

El desplazamiento de programa sirve por ejemplo, después de haber realizado una búsqueda o cuando trayectorias de movimiento similares se van repitiendo en diferentes lugares en el programa.

Ejemplos

```
MoveL p10, v500, z10, herramienta1;
PDispOn \ExeP:=p10, p20, herramienta1;
```

Activación de un desplazamiento de programa (movimiento paralelo). Esto se calculará basándose en la diferencia entre las posiciones *p10* y *p20*.

```
MoveL p10, v500, fine, herramienta1;
PDispOn *, herramienta1;
```

Activación de un desplazamiento de programa (movimiento paralelo). Dado que se ha utilizado un punto de paro en la instrucción anterior, no será necesario utilizar el argumento *\ExeP*. El desplazamiento será calculado basándose en la diferencia entre la posición actual del robot y el punto programado (*) almacenado en la instrucción.

```
PDispOn \Rot \ExeP:=p10, p20, herramienta1;
```

Activación de un desplazamiento de programa incluyendo una rotación. Esto se calculará basándose en la diferencia entre las posiciones *p10* y *p20*.

Argumentos

PDispOn [\Rot] [\ExeP] PuntProg Herram [\WObj]

[\Rot]	<i>(Rotación)</i>	Tipo de dato: <i>switch</i>
-----------------	-------------------	-----------------------------

La diferencia de la orientación de la herramienta será tomada en cuenta y ello implicará una rotación del programa.

[\ExeP]	<i>(Punto Ejecutado)</i>	Tipo de dato: <i>robtarget</i>
------------------	--------------------------	--------------------------------

La nueva posición del robot en el momento de la ejecución del programa. Si este argumento es omitido, se utilizará la posición utilizada por el robot en el momento de la ejecución del programa.

PuntProg (*Punto Programado*) Tipo de dato: *robtarget*

La posición original del robot en el momento de la programación.

Herram Tipo de dato: *tooldata*

La herramienta utilizada durante la programación, es decir, el TCP al que se refiere la posición *PuntProg*.

[\WObj] (*Objeto de trabajo*) Tipo de dato: *wobjdata*

El objeto de trabajo (sistema de coordenadas) al que se refiere la posición *PuntProg*.

Este argumento podrá ser omitido, y en el caso en que lo sea, la posición se referirá al sistema de coordenadas mundo. Sin embargo, si se utiliza un TCP estacionario o ejes externos coordinados, este argumento deberá ser especificado.

Los argumentos *Herram* y *\WObj* se utilizan tanto para calcular el *PuntProg* durante la programación como para calcular la posición actual durante la ejecución del programa, siempre y cuando no haya ningún argumento *ExeP* programado.

Ejecución del programa

Un desplazamiento de programa significa que el sistema de coordenadas ProgDisp será trasladado respecto al sistema de coordenadas del objeto. Dado que todas las posiciones se refieren al sistema de coordenadas ProgDisp, todas las posiciones programadas serán también desplazadas. Véase la Figura 1.

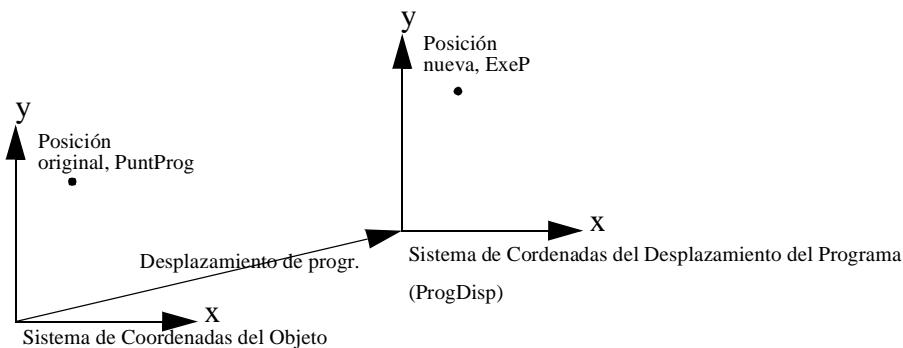


Figura 1 Desplazamiento de una posición programada utilizando un desplazamiento de programa.

El desplazamiento de programa se activará cuando la instrucción *PDispOn* haya sido ejecutada y permanecerá activado hasta que se active otro desplazamiento de programa (la instrucción *PDispSet* o *PDispOn*) o hasta que se desactive otro desplazamiento de programa (la instrucción *PDispOff*).

Sólo se podrá activar un desplazamiento a la vez. No obstante, se podrá programar varias instrucciones *PDispOn*, una tras otra y, en este caso, los diferentes desplazamientos del programa serán añadidos.

El desplazamiento de programa será calculado como la diferencia entre *ExeP* y *PuntProg*. En el caso en que *ExeP* no haya sido especificado, se utilizará la posición utilizada del robot en este momento de la ejecución del programa. Dado que se utiliza la posición actual del robot, el robot no deberá moverse cuando se ejecuta la instrucción *PDispOn*.

Si se utiliza el argumento *|Rot*, la rotación será calculada también basándose en la orientación de la herramienta en las dos posiciones. El desplazamiento será calculado de forma que la posición nueva (*ExeP*) tenga la misma posición y orientación, respecto al sistema de coordenadas desplazado *ProgDisp*, que la posición antigua (*PuntProg*) respecto al sistema de coordenadas original (véase la Figura 2).

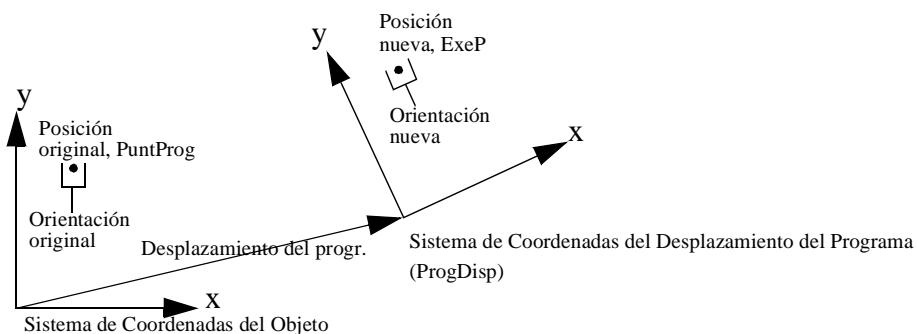


Figura 2 Translación y rotación de una posición programada.

El desplazamiento del programa será reinicializado automáticamente:

- a la puesta en marcha
- cuando se carga un programa nuevo
- cuando se ejecuta el programa desde el principio

Ejemplo

```

PROC dibuja_cuadrado()
    PDispOn *, herramienta;
    MoveL *, v500, z10, herramienta;
    PDispOff;
ENDPROC

MoveL p10, v500, fine, herramienta;
dibuja_cuadrado;
MoveL p20, v500, fine, herramienta;
dibuja_cuadrado;
MoveL p30, v500, fine, herramienta;

```

dibuja_cuadrado;

La rutina *dibuja_cuadrado* sirve para ejecutar la misma estructura de movimiento en tres posiciones diferentes, basándose en las posiciones *p10*, *p20* y *p30*. Véase la Figura 3.

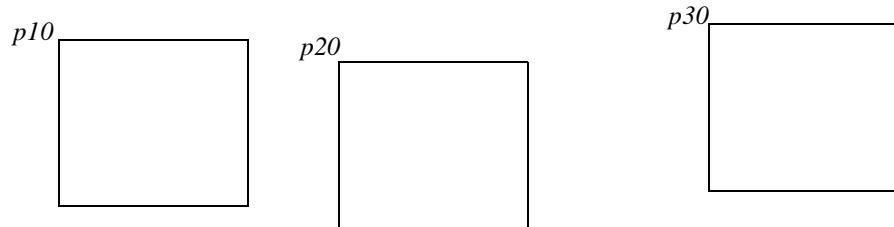


Figura 3 Utilizando un desplazamiento de programa, las estructuras de movimiento podrán ser reutilizadas.

```
SearchL sen1, pbusc, p10, v100, herram1\WObj:=fijación1;
PDispOn \ExeP:=pbusc, *, herram1 \WObj:=fijación1;
```

Se realiza una búsqueda en la que la posición buscada del robot está almacenada en la posición *psearch*. Cualquier movimiento llevado a cabo después de esto empezará a partir de esta posición utilizando un desplazamiento de programa (movimiento paralelo). Este último será calculado basándose en la diferencia entre la posición buscada y el punto programado (*) almacenado en la instrucción. Todas las posiciones están basadas en el sistema de coordenadas del objeto *fixture1*.

Sintaxis

```
PDispOn
[ '\' Rot ',' ]
[ '\' ExeP ':=' ] < expresión (IN) de robtarget > ',' ]
[ ProgPoint ':=' ] < expresión (IN) de robtarget > ',' ,
[ Tool ':=' ] < persistente (PERS) de tooldata>
[ '\'WObj ':=' < persistente (PERS) de wobjdata> ] ';'
```

Información relacionada

Descripción:

Desactivación del desplazamiento del programa

Instrucciones - *PDispOff*

Definición del desplazamiento del programa utilizando valores

Instrucciones - *PDispSet*

Sistemas de coordenadas

Principios de Movimiento y de E/S -
Sistemas de Coordenadas

Definición de las herramientas

Tipos de datos - *tooldata*

Definición de los objetos de trabajo

Tipos de datos - *wobjdata*

Más ejemplos

Instrucciones - *PDispOff*

PDispSet**Activación de un desplazamiento del programa utilizando un valor**

PDispSet (*Program Displacement Set*) sirve para definir y activar un desplazamiento de programa utilizando valores.

Se utilizará un desplazamiento de programa cuando, por ejemplo, se repiten estructuras de movimiento similares a diferentes lugares del programa.

Ejemplo

```
VAR pose xp100 := [ [100, 0, 0], [1, 0, 0, 0] ];
```

```
PDispSet xp100;
```

La activación del desplazamiento de programa *xp100* significa que:

- El sistema de coordenadas ProgDisp será desplazado de 100 mm del sistema de coordenadas del objeto, en dirección del eje x- positivo (véase la Figura 1).
- Mientras este desplazamiento de programa está activado, todas las posiciones serán desplazadas de 100 mm en la dirección del eje x-.

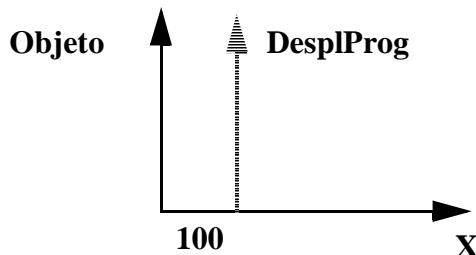


Figura 1 Un desplazamiento de programa de 100 mm en el eje x-.

Argumentos**PDispSet BaseDesp**

BaseDesp(*Base de Desplazamiento*)

Tipo de dato: *pose*

El desplazamiento de programa será definido como un dato del tipo *pose*.

Ejecución del programa

El desplazamiento de programa implica la translación y/o la rotación del sistema de coordenadas ProgDisp respecto al sistema de coordenadas del objeto. Dado que todas las posiciones se refieren al sistema de coordenadas ProgDisp, todas las posiciones programadas serán también desplazadas. Véase la Figura 2.

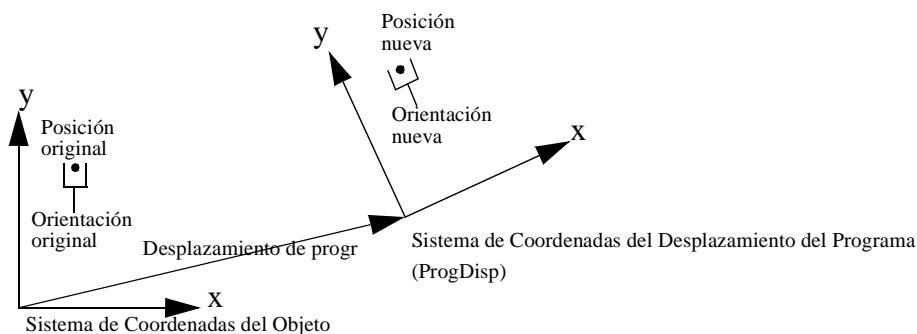


Figura 2 Translación y rotación de una posición programada.

El desplazamiento de programa se activará cuando la instrucción *PDispSet* se ejecute y permanecerá activado hasta que se active otro desplazamiento de programa (la instrucción *PDispSet* o *PDispOn*) o hasta que se desactive el desplazamiento de programa (la instrucción *PDispOff*).

Sólo se podrá activar un desplazamiento de programa a la vez. Los programas de desplazamiento no podrán añadirse el uno al otro utilizando *PDispSet*.

El desplazamiento del programa será reinicializado automáticamente:

- a la puesta en marcha
- cuando se carga un programa nuevo
- cuando se ejecuta la primera instrucción en el programa.

Sintaxis

```
PDispSet
[ BaseDesp':=' ] <expresión (IN) de pose> ';
```

Información relacionada

Desactivación de un desplazamiento de programa

Descripción:
Instrucciones - *PDispOff*

Definición de un desplazamiento de programa utilizando dos posiciones

Instrucciones - *PDispOn*

Definición de los tipos de dato *pose*

Tipos de Datos - *pose*

Sistemas de Coordenadas

Principios de Movimiento y de E/S -
Sistemas de Coordenadas

Ejemplos de utilización de un desplazamiento de programas

Instrucciones - *PDispOn*

PathResol Ajuste de la resolución de trayectoria

PathResol (Path Resolution) sirve para ajustar la resolución de la trayectoria geométrica configurada (tiempo muestra geométrico) definida en los parámetros del sistema del manipulador.

Descripción

La resolución de la trayectoria afecta a la precisión de la trayectoria interpolada y el tiempo de ciclo del programa. La precisión de la trayectoria resulta mejorada y el tiempo de ciclo es a menudo reducido cuando se disminuye el argumento de *PathResol*. No obstante, un valor demasiado bajo del argumento de *PathResol* puede provocar problemas de carga de la CPU en ciertas aplicaciones exigentes. Pero téngase en cuenta que la utilización de una resolución estándar de trayectoria configurada (*PathResol 100%*) evitará problemas de carga de la CPU y proporcionará la precisión de trayectoria suficiente en la mayoría de las situaciones.

Ejemplo de utilización de *PathResol*:

Movimientos críticamente dinámicos (carga útil máxima, velocidad elevada, movimientos de ejes combinados cerca del límite del área de trabajo) pueden provocar problemas de carga de la CPU. Se deberá aumentar el argumento de *PathResol*.

Una baja capacidad de los ejes externos puede provocar problemas de carga de la CPU durante la coordinación. Se deberá aumentar el argumento de *PathResol*.

La soldadura al arco con una elevada frecuencia de onda puede requerir una elevada resolución de trayectoria interpolada. Se deberá disminuir el argumento de *PathResol*.

Los círculos pequeños o los pequeños movimientos combinados con cambios de dirección pueden disminuir la calidad de la capacidad de la trayectoria y aumentar el tiempo de ciclo. Se deberá disminuir el argumento de *PathResol*.

La aplicación de adhesivo con grandes reorientaciones y las zonas esquina pequeñas pueden provocar variaciones de la velocidad. Se deberá disminuir el argumento de *PathResol*.

Ejemplo

```
MoveJ p1,v1000,fine,tool1;  
PathResol 150;
```

Con el robot en un punto de paro, la resolución de trayectoria es disminuida del 150% respecto a la configurada.

Argumentos

PathResol Valor

Valor	Tipo de dato: <i>num</i>
--------------	--------------------------

El ajuste como un porcentaje de la resolución de la trayectoria configurada (los valores elevados disminuyen la resolución de la trayectoria).

El 100% corresponde a la resolución de trayectoria configurada.

Dentro de los límites 25-400%.

Ejecución del programa

La resolución de trayectoria de todas las instrucciones de posicionamiento subsiguientes están afectadas hasta que se ejecute una nueva instrucción *PathResol*. Ello afectará a la resolución de trayectoria durante toda la ejecución de movimientos del programa (nivel de trayectoria por defecto y nivel de trayectoria después de *StorePath*) y también durante el movimiento manual.

El valor por defecto del ajuste de la resolución de trayectoria es del 100%. Este valor es activado automáticamente:

- a la puesta en marcha en frío
- cuando se carga un programa nuevo
- cuando se inicia la ejecución del programa desde el principio.

El ajuste corriente de la resolución de la trayectoria puede leerse a partir de la variable *C_MOTSET* (tipo de dato *motsetdata*) en el componente *pathresol*.

Limitaciones

El robot debe estar inmovilizado en un punto de paro antes de realizar el ajuste de la resolución de la trayectoria. Cuando hay una trayectoria esquina en el programa, el sistema creará en lugar de ello un punto de paro (aviso 50146) y no será posible rearrancar en esta instrucción después de un corte del suministro de corriente.

Sintaxis

```
PathResol
[Valor ':=' ] <expresión (IN) de num> ';
```

Información relacionada

Descripción:

Instrucciones de posicionamiento

Principios de Movimiento y de E/S -
Movimientos

Características de movimiento

Resumen RAPID - *Características de Movimiento*

Ajuste actual de la resolución de trayectoria

Tipos de datos - *Datos del Sistema*

ProcCall Llamada de un procedimiento nuevo

Una llamada procedimiento sirve para transferir la ejecución del programa a otro procedimiento. Cuando el procedimiento ha sido ejecutado completamente, la ejecución del programa continua con la instrucción que sigue la llamada de procedimiento.

Normalmente se puede enviar una serie de argumentos al procedimiento nuevo. Estos tendrán por misión controlar el comportamiento del procedimiento y hacer que sea posible utilizar el mismo procedimiento para diferentes utilidades.

Ejemplos

```
weldpipe1;  
                  Llamada del procedimiento weldpipe1.
```

```
errormessage;  
Set do1;
```

```
.  
PROC errormessage()  
  TPWrite "ERROR";  
ENDPROC
```

El procedimiento *errormessage* será llamado. Cuando este procedimiento esté listo, la ejecución del programa regresará a la intrucción que sigue la llamada de procedimiento, *Set do1*.

Argumentos

Procedure { Argument }

Procedure	Identificador
-----------	---------------

El nombre del procedimiento que se desea llamar.

Argument	Tipo de dato: De acuerdo con la declaración de procedimiento
----------	--

Los argumentos del procedimiento (de acuerdo con los parámetros del procedimiento).

Ejemplo

```
weldpipe2 10, lowspeed;
```

Llamará al procedimiento *weldpipe2*, incluyendo dos argumentos.

```
weldpipe3 10 \speed:=20;
```

Llamará al procedimiento *weldpipe3*, incluyendo un argumento obligatorio y otro opcional.

Limitaciones

Los argumentos del procedimiento deben coincidir con sus parámetros:

- Todos los argumentos obligatorios deberán ser incluidos.
- Deberán ser colocados siguiendo el mismo orden.
- Deberán ser del mismo tipo de dato.
- Deberán ser del tipo correcto en relación al modo de acceso (entrada, variable o persistente).

Una rutina puede llamar a otra rutina que, a su vez, llama a otra rutina, y así sucesivamente. Una rutina también podrá llamarse a sí misma, es decir, que realizará una llamada recursiva. El número de niveles de rutina permitido dependerá del número de parámetros, pero por lo general se permiten más de 10 niveles.

Sintaxis

(EBNF)

```
<procedimiento> [ <lista argumento> ] ';'
```

```
<procedimiento> ::= <identificador>
```

Información relacionada

Argumentos, parámetros
Más ejemplos

Descripción:

Características Básicas - *Rutinas*
Ejemplos

PulseDO

Generación de un pulso en una señal de salida digital

PulseDO sirve para generar un pulso en una señal de salida digital.

Ejemplos

PulseDO do15;

Se generará un pulso de 0,2 s en la señal de salida *d015*.

PulseDO \PLength:=1.0, conexión;

Se generará un pulso de $1,0\text{ s}$ en la señal *conexión*.

Argumentos

PulseDO [\PLength] Señal

[\PLength]

(Duración del pulso)

Tipo de dato: *num*

La duración del pulso en segundos (0,1 - 32 seg.).

Si se omite el argumento, se generará un pulso de 0,2 segundos.

Señal

Tipo de dato: *signaldo*

El nombre de la señal en la que deberá generarse un pulso.

Ejecución del programa

Se generará un pulso con una longitud de pulso específica (véase la Figura 1).

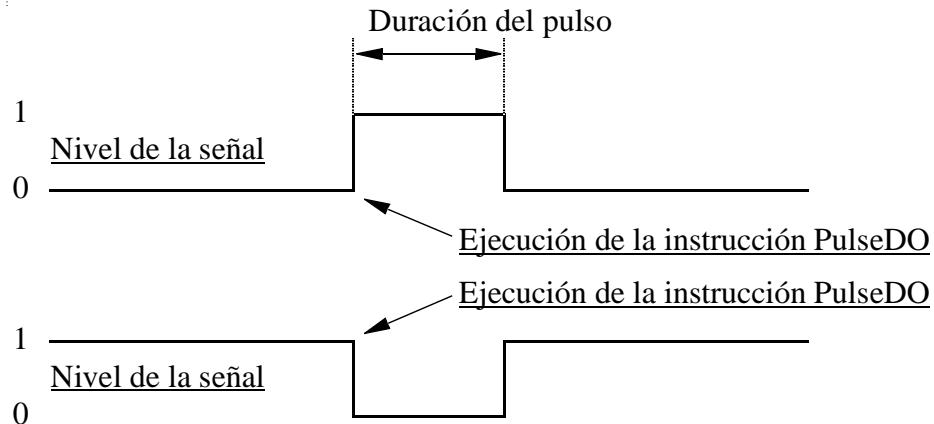


Figura 1 Generación de un pulso en una señal de salida digital.

La instrucción siguiente se ejecutará después del inicio del pulso. El pulso podrá entonces ser activado/reinicializado sin afectar el resto de la ejecución del programa.

Limitaciones

La duración del pulso tiene una resolución de 0,01 segundos. Los valores programados que difieran de éste serán redondeados.

Sintaxis

```
PulseDO
[ '\' PLength ':=' <expresión (IN) de num > ',' ]
[ Señal ':=' ] <variable (VAR) de signaldo > ';'
```

Información relacionada

Instrucciones de E/S

Descripción:

Resumen RAPID -
Señales de Entrada y Salida

Funciones de las E/S en general

Principios de Movimiento y de E/S-
Principios de E/S

Configuración de las E/S

Guía del Usuario - Parámetros del
Sistema

RAISE**Llamada al gestor de error**

RAISE sirve para crear un error en el programa y luego llamar el gestor de errores de la rutina. *RAISE* también podrá utilizarse en el gestor de errores para propagar el error actual al gestor de error de la rutina que llama.

La instrucción puede, por ejemplo, utilizarse para saltar a un nivel superior en la estructura del programa, por ejemplo, al gestor de errores de la rutina principal, cuando ocurre un error en un nivel inferior.

Ejemplo

```
IF ...
  IF ...
    IF ...
      RAISE escape1;

ERROR
  IF ERRNO=escape1 RAISE;
```

La rutina será interrumpida para permitir retirarse de un nivel bajo del programa.
Se produce un salto al gestor de errores de la rutina llamada.

Argumentos**RAISE [Nº Error]****Nº Error**Tipo de dato: *errnum*

Número de error: Cualquier número entre 1-90 que el gestor de errores puede utilizar para localizar el error que se ha producido (la variable del sistema *ERRNO*).

También se podrá utilizar un número de error fuera de 1-90 mediante la instrucción *BookErrNo*.

El número de error deberá ser especificado fuera del gestor de errores en una instrucción *RAISE* para ser capaz de transferir la ejecución al gestor de errores de esta rutina.

Si la instrucción está presente en un gestor de errores de una rutina, no es necesario que el número de error esté especificado. En tal caso, el error será propagado al gestor de errores de la rutina que llama.

Ejecución del programa

La ejecución del programa continúa en el gestor de errores de la rutina. Cuando el ges-

tor de errores haya sido ejecutado, la ejecución del programa proseguirá con:

- la rutina que ha llamado la rutina correspondiente (RETURN),
- el gestor de errores de la rutina que ha llamado la rutina correspondiente (RAISE).

En el caso en que la instrucción RAISE esté presente en un gestor de errores de la rutina, la ejecución del programa continúa en el gestor de errores de la rutina que ha llamado la rutina correspondiente. El mismo número de error permanece activo.

En el caso en que la instrucción RAISE esté presente en una rutina de tratamiento de interrupciones, el error será procesado por el gestor de errores del sistema.

Gestión de errores

En el caso en que el número de error esté fuera de alcance, la variable del sistema ERRNO se activará en ERR_ILLRAISE (véase "Tipos de Datos - errnum"). Este error puede ser manipulado en el gestor de error.

Sintaxis

(EBNF)

RAISE [<número de error>] ;

<número de error> ::= <expresión>

Información relacionada

Descripción:

Gestión de errores

Características Básicas -
Recuperación de errores

Utilizar números de error

Instrucciones - *BookErrNo*

Reset Puesta a cero de una señal de salida digital

Reset sirve para poner una salida digital a cero.

Ejemplos

Reset do15;

La señal *do15* se pondrá a 0.

Reset sold;

La señal *sold* se pondrá a 0.

Argumentos

Reset Señal

Señal

Tipo de dato: *signaldo*

El nombre de la señal que se desea poner a 0.

Ejecución del programa

El valor verdadero depende de la configuración de la señal. Si la señal ha sido invertida en los parámetros del sistema, la instrucción hará que el canal físico se ponga en 1.

Sintaxis

Reset
[Señal ':='] < variable (**VAR**) de *signaldo* > ';

Información Relacionada

	<u>Descrita en:</u>
Activación de una señal de salida digital	Instrucciones - <i>Activación</i>
Instrucciones de E/S	Resumen RAPID - <i>Señales de Entrada y Salida</i>
Funciones de las E/S en general	Principios de Movimiento y de E/S - <i>Principios de E/S</i>
Configuración de las E/S	Guía del Usuario - <i>Parámetros del Sistema</i>

RestoPath

Restauración de la trayectoria después de una interrupción

RestoPath sirve para restaurar una trayectoria que ha sido almacenada previamente mediante la instrucción *StorePath*.

Ejemplo

RestoPath;

Restaurar la trayectoria que ha sido almacenada anteriormente mediante la instrucción *StorePath*.

Ejecución del programa

La trayectoria de movimiento utilizada del robot y de los ejes externos será borrada y la trayectoria almacenada anteriormente mediante la instrucción *StorePath* será restaurada. No se moverá nada hasta que se ejecute la instrucción *StartMove* o que se realice un retorno utilizando la instrucción *RETRY* a partir de un gestor de errores.

Ejemplo

```
ArcL p100, v100, insol1, sold5, oscil1, z10, boq1;  
...  
ERROR  
  IF ERRNO=AW_WELD_ERR THEN  
    limpia_boq;  
    RETRY;  
  ENDIF  
...  
PROC limpia_boq()  
  VAR robtarget p1;  
  StorePath;  
  p1 := CRobT();  
  MoveL plimp, v100, fine, boq1;  
...  
  MoveL p1, v100, fine, boq1;  
  RestoPath;  
ENDPROC
```

En el caso de ocurrir un error de soldadura, la ejecución del programa continuará en el gestor de errores de la rutina, que, a su vez llamará *limpia_boq*. La trayectoria de movimiento que se está ejecutando en este momento será entonces almacenada y el robot se moverá a la posición *plimp* donde el error será rectificado. Una vez esto haya sido realizado, el robot regresará a la posición

donde ocurrió el error, *p1*, y volverá a almacenar el movimiento original. En este momento, la soldadura rearrancará automáticamente, lo que significa que el robot regresará en la trayectoria antes de iniciar la soldadura y de proseguir con la ejecución normal del programa.

Limitaciones

El movimiento que se está ejecutando en el momento deberá en primer lugar ser parado, utilizando por ejemplo un punto de paro, antes de que la instrucción *RestoPath* pueda ser ejecutada.

La instrucción de movimiento que precede a esta instrucción deberá terminar con un punto de paro para que, en esta instrucción, sea posible realizar un rearranque después de un corte de potencia.

Sintaxis

`RestoPath‘;’`

Información relacionada

Almacenamiento de trayectorias
Más ejemplos

Descripción:

Instrucciones - *StorePath*
Instrucciones - *StorePath*

RETRY**Rearranque después de un error**

RETRY sirve para rearrancar la ejecución del programa después de que se haya producido un error.

Ejemplo

```
reg2 := reg3/reg4;  
.  
ERROR  
IF ERRNO = ERR_DIVZERO THEN  
    reg4 := 1;  
    RETRY;  
ENDIF
```

Se ha intentado dividir *reg3* por *reg4*. Si *reg4* es igual a 0 (división por cero), se realizará un salto al gestor de errores, que inicializará *reg4*. La instrucción *RETRY* se utilizará entonces para saltar del gestor de errores y se realizará otro intento para llevar a cabo la división.

Ejecución del programa

La ejecución del programa (vuelve a ejecutar) continúa con la instrucción que ha provocado el error.

Gestión de errores

En el caso en que el número máximo de reintentos (4) haya sido excedido, la ejecución del programa se detendrá con un mensaje de error y la variable del sistema ERRNO se activará en ERR_EXCRTYMAX (véase Tipos de datos - *errnum*).

Limitaciones

La instrucción solamente podrá existir en un gestor de errores de rutina. En el caso de que el error haya sido creado al utilizar una instrucción *RAISE*, la ejecución del programa no podrá ser rearrancada mediante una instrucción *RETRY*, en vez de ésta se deberá usar la función *TRYNEXT*.

Sintaxis

RETRY ';

Información relacionada

Descripción:

Gestores de errores

Características Básicas -
Recuperación de Errores

Continuar con la siguiente instrucción

Instrucciones - *TRYNEXT*

RETURN**Fin de ejecución de una rutina**

RETURN sirve para finalizar la ejecución de una rutina. Si la rutina es una función, el valor de la función será devuelto.

Ejemplos

```
mensajerror;  
Set do1;
```

```
.  
PROC mensajerror()  
    TPWrite "ERROR";  
    RETURN;  
ENDPROC
```

El procedimiento *mensajerror* será llamado. Cuando el procedimiento llega a la instrucción *RETURN*, la ejecución del programa continúa en la instrucción que sigue la llamada del procedimiento, *Set do1*.

```
FUNC num valor_abs (num valor)  
    IF valor<0 THEN  
        RETURN -valor;  
    ELSE  
        RETURN valor;  
    ENDIF  
ENDFUNC
```

La función devuelve el valor absoluto de un número.

Argumentos**RETURN [Valor devuelto]****Valor devuelto**

Tipo de dato: Según la declaración de la función

El valor devuelto de una función.

El valor devuelto debe ser especificado en una instrucción *RETURN* presente en la función.

Si la instrucción está en un procedimiento o en una rutina de tratamiento de interrupciones, puede ocurrir que un valor devuelto no esté especificado.

Ejecución del programa

El resultado de la instrucción *RETURN* podrá variar, dependiendo del tipo de rutina en la que está utilizada:

- Rutina principal: En el caso en que el arranque del programa haya sido ordenado al final del ciclo, el programa se para. De lo contrario, la ejecución del programa continúa con la primera instrucción de la rutina principal.
- Procedimiento: La ejecución del programa continúa con la instrucción que sigue a la llamada del procedimiento.
- Función: Devuelve el valor de la función.
- Rutina tratamiento de interrupciones: La ejecución del programa continúa a partir de donde se ha producido la interrupción.
- Gestor de errores:
 - En un procedimiento:
La ejecución del programa continúa con la rutina que llamó la rutina con el gestor de error (con la instrucción que sigue la llamada del procedimiento).
 - En una función:
El valor de la función es devuelto.

Sintaxis

(EBNF)

RETURN [<expresión>]';'

Información Relacionada

Funciones y Procedimientos

Descripción:

Características Básicas - *Rutinas*

Rutinas de tratamiento de interrupciones

Características Básicas -
Interrupciones

Gestores de error

Características Básicas - *Recuperación de errores*

Rewind Reiniciar la posición del archivo

Rewind sitúa la posición del archivo al principio del mismo.

Ejemplo

Rewind iodev1;

El archivo referido con el nombre de *iodev1* tendrá su posición situada al principio del archivo.

Argumentos**Rewind DispositivoE/S****DispositivoE/S**Tipo de dato: *iodev*

Es el nombre (referencia) del archivo que debe ser reiniciado.

Ejecución del programa

El archivo especificado es reposicionado al principio.

Ejemplo

```
! IO device and numeric variable for use together with a binary file
VAR iodev dev;
VAR num bindata;

! Open the binary file with \Write switch to erase old contents
Open "flp1:"\File := "bin_file",dev \Write;
Close dev;

! Open the binary file with \Bin switch for binary read and write access
Open "flp1:"\File := "bin_file",dev \Bin;
WriteStrBin dev,"Hello world";

! Rewind the file pointer to the beginning of the binary file
! Read contents of the file and write the binary result on TP
! (gives 72 101 108 108 111 32 119 111 114 108 100 )
Rewind dev;
bindata := ReadBin(dev);
WHILE bindata <> EOF_BIN DO
    TPWrite " " \Num:=bindata;
    bindata := ReadBin(dev);
ENDWHILE

! Close the binary file
Close dev;
```

La instrucción *Rewind* sirve para reposicionar un archivo binario al principio de forma que el contenido del mismo pueda ser leído con la instrucción *ReadBin*.

Sintaxis

```
Rewind
[DispositivoE/S ':='] <variable (VAR) de iodev>;
```

Información relacionada

Apertura (etc.) de archivos

Descripción:

Resumen RAPID - *Comunicación*

SearchC

Búsqueda circular utilizando el robot

SearchC (Search Circular) sirve para buscar una posición moviendo el punto central de la herramienta (TCP) de forma circular.

Durante el movimiento, el robot supervisa una señal de entrada digital. Cuando el valor de la señal pasa al deseado, el robot inmediatamente lee la posición actual.

Esta instrucción suele utilizarse cuando la herramienta sujetada por el robot es un sensor para la detección de la superficie. Utilizando la instrucción *SearchC*, se podrá obtener las coordenadas generales de un objeto de trabajo.

Ejemplos

SearchC sen1, polmc, puncirc, p10, v100, sensor;

El TCP del *sensor* se moverá de forma circular hacia la posición *p10* a una velocidad de *v100*. Cuando el valor de la señal *sen1* pasa a ser activa, la posición se almacenará en *polmc*.

SearchC \Stop, sen1, polmc, puncirc, p10, v100, sensor;

El TCP del *sensor* se moverá de forma circular hacia la posición *p10*. Cuando el valor de la señal *sen1* pasa a ser activa, la posición se almacenará en *polmc* y el robot se detendrá inmediatamente.

Argumentos

**SearchC [\Stop] | [\PStop] | [\Sup] Señal [\Flanks] PuntBusc
PuntCirc AlPunto Veloc [\V] | [\T] Herram [\WObj]
[\Corr]**

[\Stop]

Tipo de dato: *switch*

El movimiento del robot se detendrá, lo más rápidamente posible sin mantener el TCP en la trayectoria (paro fuerte), cuando el valor de la señal de búsqueda pasa a ser activa. No obstante, el robot se moverá de una distancia pequeña antes de pararse y no regresará a la posición buscada, es decir, a la posición donde la señal ha cambiado.

[\PStop]

(Paro trayectoria)

Tipo de dato: *switch*

El movimiento del robot es parado, lo más rápidamente posible, manteniendo el TCP en la trayectoria (paro suave), cuando el valor de la señal de búsqueda pasa a ser activa. No obstante, el robot se mueve de una pequeña distancia antes de pararse y no regresará a la posición buscada, es decir, a la posición donde la señal ha cambiado.

[\Sup] *(Supervisión)* Tipo de dato: *switch*

La instrucción de búsqueda es sensible a la activación de señales durante el movimiento completo (búsqueda sobre la marcha), es decir, incluso después de que el primer cambio de señal haya sido registrado. Si se produce más de una detección durante una búsqueda, la ejecución del programa se para.

En el caso en que se omitan los argumentos *\Stop*, *\PStop* o *\Sup*, el movimiento continua (búsqueda sobre la marcha) a la posición especificada en el argumento *AlPunto* (lo mismo que con el argumento *\Sup*).

Señal Tipo de dato: *signaldi*

Es el nombre de la señal que se debe supervisar.

[\Flanks] Tipo de dato: *switch*

El lado positivo y negativo de la señal es válido para una búsqueda.

Si se omite el argumento *\Flanks*, sólo el lado positivo de la señal será válido para una búsqueda y se activará una supervisión de la señal al principio del proceso de búsqueda. Esto significa que si la señal todavía tiene un valor positivo al principio de un proceso de búsqueda, el movimiento del robot será detenido lo más rápidamente posible, mientras que el TCP se mantiene en la trayectoria (paro suave). No obstante, el robot se moverá de una pequeña distancia antes de detenerse y no regresará a la posición de inicio. Se generará un error de recuperación del usuario (ERR_SIGSUPSEARCH) que podrá ser tratado por el gestor de errores.

PuntBusc Tipo de dato: *robtarget*

La posición del TCP y de los ejes externos cuando la señal de búsqueda ha sido disparada. La posición será especificada en el sistema de coordenadas más exterior teniendo en cuenta la herramienta especificada, el objeto de trabajo y el sistema de coordenadas ProgDisp/ExtOffs activo.

PuntCirc Tipo de dato: *robtarget*

Es el punto de círculo del robot. Véase la instrucción MoveC para una descripción más detallada del movimiento circular. El punto de círculo está definido como una posición con nombre o es almacenado directamente en la instrucción (marcado con un asterisco * en la instrucción).

AlPunto Tipo de dato: *robtarget*

Es el punto de destino del robot y de los ejes externos. Está definido como una posición con nombre o es almacenado directamente en la instrucción (marcado con un asterisco * en la instrucción). SearchC siempre utiliza un punto de paro como dato de zona para el destino.

Velocidad Tipo de dato: *speeddata*

Son los datos de velocidad que se aplican a los movimientos. El dato de velocidad

define la velocidad del punto central de la herramienta, de los ejes externos y de la reorientación de la herramienta.

[\V]	<i>(Velocidad)</i>	Tipo de dato: <i>num</i>
---------------	--------------------	--------------------------

Este argumento sirve para especificar la velocidad del TCP en mm/s directamente en la instrucción. Más adelante, este argumento será sustituido por la velocidad correspondiente especificada en los datos de velocidad.

[\T]	<i>(Tiempo)</i>	Tipo de dato: <i>num</i>
---------------	-----------------	--------------------------

Este argumento sirve para especificar el tiempo total en segundos durante el cual se mueve el robot. Más adelante, este argumento será sustituido por los datos de velocidad correspondientes.

Herram		Tipo de dato: <i>tooldata</i>
---------------	--	-------------------------------

Es la herramienta en uso cuando el robot se mueve. El punto central de la herramienta es el punto que se mueve a la posición de destino especificada.

[\WObj]	<i>(Objeto de trabajo)</i>	Tipo de dato: <i>wobjdata</i>
------------------	----------------------------	-------------------------------

Es el objeto de trabajo (sistema de coordenadas) al que se refieren las posiciones del robot en la instrucción.

Este argumento puede omitirse, y en el caso en que lo sea, la posición se referirá al sistema de coordenadas mundo. Si, por otra parte, se utiliza un TCP estacionario o ejes externos coordinados, este argumento deberá estar especificado para un movimiento lineal referido al objeto de trabajo correspondiente.

[\Corr]	<i>(Corrección)</i>	Tipo de dato: <i>switch</i>
------------------	---------------------	-----------------------------

Cuando este argumento está presente, los datos de corrección escritos en una entrada mediante la instrucción *CorrWrite* serán añadidos a la trayectoria y al punto de destino.

Ejecución del programa

Véase la instrucción *MoveC* para más información respecto al movimiento circular.

El movimiento se termina siempre con un punto de paro, es decir, que el robot se para en el punto de destino.

Si se utiliza una función de búsqueda sobre la marcha, es decir, cuando el argumento *\Sup* ha sido especificado, el movimiento del robot siempre continuará hacia el punto de destino programado. Si la búsqueda se realiza utilizando el argumento *\Stop*, o *\PStop*, el movimiento del robot se parará cuando se detecte la primera señal.

La instrucción *SearchC* devuelve la posición del TCP cuando el valor de la señal digital pasa a ser la requerida, según se indica en la Figura 1.

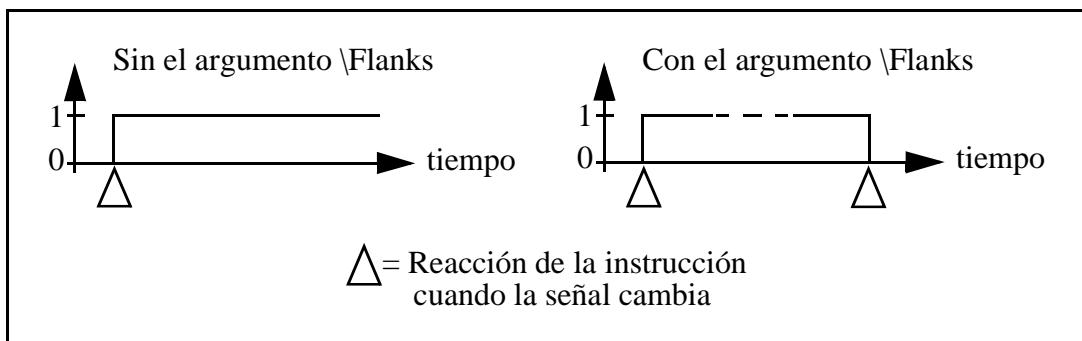


Figura 1 Detección de señal disparada por el flanco (la posición es almacenada cuando la señal cambia por primera vez únicamente).

Ejemplo

```
SearchC \Sup, sen1, \Flanks, polmc, puncirc, p10, v100, sensor;
```

El TCP del *sensor* se moverá de forma circular hacia la posición *p10*. Cuando el valor de la señal *sen1* pasa a ser activa o pasiva, la posición será almacenada en *polmc*. En el caso en que el valor de la señal cambie dos veces, la ejecución del programa se detiene.

Limitaciones

Los datos de zona de la instrucción de posicionamiento que preceden *SearchC* deberán utilizarse con mucho cuidado. El arranque de la búsqueda, es decir, cuando la señal de E/S está lista para reaccionar, no corresponde en este caso con el punto de destino programado de la instrucción de posicionamiento anterior, sino que es un punto situado en la trayectoria real del robot. En la Figura 2 se indica un ejemplo de una situación en que puede haber problemas por utilizar otros datos de zona que no sea *fine*.

La instrucción *SearchC* no deberá nunca ser rearrancada después de haber pasado el punto de círculo. De lo contrario el robot no emprenderá la trayectoria programada (posicionamiento en torno a la trayectoria circular en otra dirección comparada con la programada).

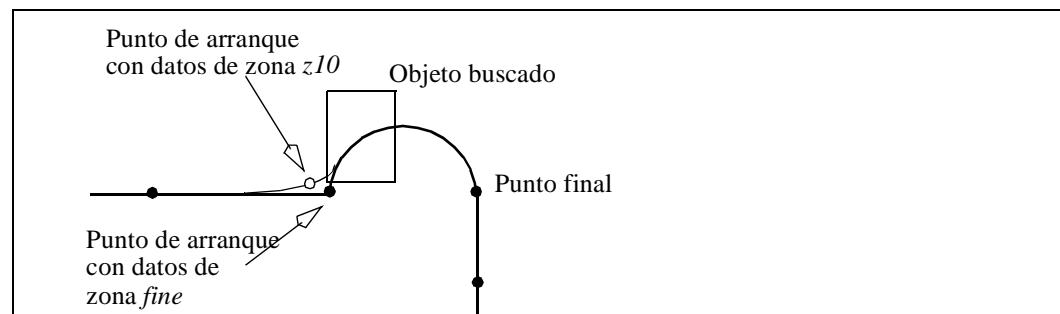


Figura 2 La detección se realiza en el lado incorrecto del objeto debido a la utilización de datos de zona incorrectos.

La distancia de paro normal cuando se utiliza una velocidad de búsqueda de 50 mm/s es de:

- 1-3 mm sin el TCP en la trayectoria (argumento *\Stop*).
- 12-16 mm con el TCP en la trayectoria (argumento *\PStop*).

Gestión de errores

Se generará un error durante una búsqueda cuando:

- no ha ocurrido ninguna detección de señal - esto genera el error **ERR_WHLSEARCH**.
- ha ocurrido más de una detección de señal - esto genera el error **ERR_WHLSEARCH** únicamente si el argumento *\Sup* ha sido utilizado.
- la señal ya ha generado un valor positivo al principio del proceso de búsqueda
- esto genera el error **ERR_SIGSUPSEARCH** únicamente si se ha omitido el argumento *\Flanks*.

Los errores serán procesados de distintas maneras según el modo de funcionamiento utilizado:

Continuo hacia adelante / ERR_WHLSEARCH

No se devuelve ninguna posición y el movimiento siempre continua hacia el punto de destino programado. La variable del sistema **ERRNO** se activará en **ERR_WHLSEARCH** y el error podrá ser procesado en el gestor de errores de la rutina.

Continuo hacia adelante / Instrucción hacia adelante / ERR_SIGSUPSEARCH

No se devuelve ninguna posición y el movimiento siempre se detiene en cuanto antes al principio de la trayectoria de búsqueda. La variable del sistema **ERRNO** se activa en **ERR_SIGSUPSEARCH** y el error puede ser tratado en el gestor de errores de la rutina.

Instrucción hacia adelante / ERR_WHLSEARCH

No se devuelve ninguna posición y el movimiento siempre continúa hacia el punto de destino programado. La ejecución del programa se para con un mensaje de error.

Instrucción hacia atrás

Durante la ejecución hacia atrás, la instrucción se limita en llevar a cabo el movimiento sin realizar ninguna supervisión de señales.

Sintaxis

SearchC
 ['\ Stop',] || ['\ PStop ',] | ['\ Sup ',]
 [Señal ':='] <variable (**VAR**) de *signaldi* >

```

[ '\' Flanks],'
[ PuntBusc':=' ] < variable o persistente (INOUT) de robtarget > ,'
[ PuntCirc':=' ] < expresión (IN) de robtarget > ,'
[ AlPunto':=' ] < expresión (IN) de robtarget > ,'
[ Veloc':=' ] < expresión (IN) de speeddata >
    [ '\' V ':=' < expresión (IN) de num > ]
    | [ '\' T ':=' < expresión (IN) de num > ] ,'
[ Herram':=' ] < persistente (PERS) de tooldata >
[ '\' WObj ':=' < persistente (PERS) de wobjdata > ]
[ '\' Corr];'

```

Información relacionada

	<u>Describo en:</u>
Búsqueda lineal	Instrucciones - <i>SearchL</i>
Escritura en una entrada de corrección	Instrucciones - <i>CorrWrite</i>
Movimiento circular	Principios de Movimiento y de E/S - <i>Posicionamiento durante la ejecución del programa</i>
Definición de la velocidad	Tipos de datos - <i>speeddata</i>
Definición de las herramientas	Tipos de datos - <i>tooldata</i>
Definición de los objetos de trabajo	Tipos de datos - <i>wobjdata</i>
Utilización de los gestores de error	Resumen RAPID - <i>Recuperación de errores</i>
Movimientos en general	Principios de Movimiento y de E/S
Más ejemplos de búsqueda	Instrucciones - <i>SearchL</i>

SearchL Búsqueda lineal utilizando el robot

SearchL (Search Linear) sirve para buscar una posición moviendo el punto central de la herramienta (TCP) de forma lineal.

Durante el movimiento, el robot supervisa una señal de entrada digital. Cuando el valor de la señal pasa al deseado, el robot inmediatamente lee la posición actual.

Esta instrucción suele utilizarse cuando la herramienta sujetada por el robot constituye un sensor para la detección de la superficie. Con esta instrucción *SearchL*, se podrán obtener las coordenadas generales de un objeto de trabajo.

Ejemplos

SearchL *sen1*, *polmc*, *p10*, *v100*, *sensor*;

El TCP del *sensor* se moverá de forma lineal hacia la posición *p10* a una velocidad de *v100*. Cuando el valor de la señal *sen1* pasa a ser activa, la posición queda almacenada en *polmc*.

SearchL \Stop, *sen1*, *polmc*, *p10*, *v100*, *sensor*;

El TCP del *sensor* se moverá de forma lineal hacia la posición *p10*. Cuando el valor de la señal *sen1* pasa a ser activa, la posición queda almacenada en *polmc* y el robot se para inmediatamente.

Argumentos

**SearchL [\Stop] | [\PStop] | [\Sup] Señal PuntBusc [\Flanks]
AlPunto Veloc [\V] | [\T] Herram [\WObj] [\Corr]**

[\Stop]

Tipo de dato: *switch*

El movimiento del robot se detiene, lo más rápidamente posible sin mantener el TCP en la trayectoria (paro fuerte), cuando el valor de la señal de búsqueda pasa a ser activa. No obstante, el robot se moverá de una pequeña distancia antes de detenerse y no regresará a la posición buscada, es decir, a la posición donde ha cambiado la señal.

[\PStop]

(Paro trayectoria)

Tipo de dato: *switch*

El movimiento del robot es parado, lo más rápidamente posible, manteniendo el TCP en la trayectoria (paro suave), cuando el valor de la señal de búsqueda pasa a ser activa. No obstante, el robot se mueve de una pequeña distancia antes de pararse y no regresará a la posición buscada, es decir, a la posición donde la señal ha cambiado.

[Sup]	(Supervisión)	Tipo de dato: <i>switch</i>
La instrucción de búsqueda es sensible a la activación de señales durante el movimiento completo (búsqueda sobre la marcha), es decir, incluso después de que el primer cambio de señal haya sido registrado. Si se produce más de una detección durante una búsqueda, la ejecución del programa se detiene.		
En el caso en que se omitan los argumentos <i>\Stop</i> , <i>\PStop</i> o <i>\Sup</i> , el movimiento continúa (búsqueda sobre la marcha) a la posición especificada en el argumento <i>AlPunto</i> (el mismo que con el argumento <i>\Sup</i>).		
Señal		Tipo de dato: <i>signaldi</i>
Es el nombre de la señal que se debe supervisar.		
[\Flanks]		Tipo de dato: <i>switch</i>
El lado positivo y negativo de la señal es válido para una búsqueda.		
Si se omite el argumento <i>\Flanks</i> , sólo el lado positivo de la señal será válido para una búsqueda y se activará una supervisión de la señal al principio del proceso de búsqueda. Esto significa que si la señal todavía tiene un valor positivo al principio de un proceso de búsqueda, el movimiento del robot será detenido lo más rápidamente posible, mientras que el TCP se mantiene en la trayectoria (paro suave). Se generará un error de recuperación del usuario (ERR_SIGSUPSEARCH) que podrá ser tratado por el gestor de errores.		
PuntBusc		Tipo de dato: <i>robtarget</i>
Es la posición del TCP y de los ejes externos cuando la señal de búsqueda se ha activado. La posición será especificada en el sistema de coordenadas más exterior, teniendo en cuenta la herramienta especificada, el objeto de trabajo y el sistema de coordenadas ProgDisp/Extoffs activo.		
AlPunto		Tipo de dato: <i>robtarget</i>
Es el punto de destino del robot y de los ejes externos. Está definido como una posición con nombre o es almacenado directamente en la instrucción (marcado con un asterisco * en la instrucción). SearchL siempre utiliza un punto de paro como dato de zona para el destino.		
Velocidad		Tipo de dato: <i>speeddata</i>
Son los datos de velocidad que se aplican a los movimientos. Los datos de velocidad definen la velocidad del punto central de la herramienta, de los ejes externos y de la reorientación de la herramienta.		
[\V]	(Velocidad)	Tipo de dato: <i>num</i>
Este argumento sirve para especificar la velocidad del TCP en mm/s directamente en la instrucción. Más adelante, será sustituido por la velocidad correspondiente especificada en los datos de velocidad.		

[\T] *(Tiempo)* Tipo de dato: *num*

Este argumento sirve para especificar el tiempo total en segundos durante el cual el robot se mueve. Más adelante, será sustituido por los datos de velocidad correspondientes.

Herram Tipo de dato: *tooldata*

Es la herramienta en uso cuando el robot se mueve. El punto central de la herramienta es el punto que se mueve a la posición de destino especificada.

[\WObj] *(Objeto de Trabajo)* Tipo de dato: *wobjdata*

Es el objeto de trabajo (sistema de coordenadas) al que se refiere la posición del robot en la instrucción.

Este argumento podrá ser omitido, y en el caso en que así sea, la posición se referirá al sistema de coordenadas mundo. Si, por otra parte, se utiliza un TCP estacionario o ejes externos coordinados, este argumento deberá ser especificado para un movimiento lineal relativo al objeto de trabajo correspondiente.

[\Corr] *(Corrección)* Tipo de dato: *switch*

Cuando este argumento está presente, los datos de corrección escritos en una entrada mediante la instrucción *CorrWrite* serán añadidos a la trayectoria y al punto de destino.

Ejecución del programa

Véase la instrucción *MoveL* para más información referente al movimiento lineal.

El movimiento siempre termina con un punto de paro, es decir, que el robot se detiene en el punto de destino.

Si se utiliza una función de búsqueda sobre la marcha, es decir, si el argumento *\Sup* ha sido especificado, el movimiento del robot siempre seguirá hacia el punto de destino programado. Si se realiza una búsqueda utilizando *\Stop* o *\PStop*, el movimiento del robot se detendrá cuando se detecte la primera señal.

La instrucción *SearchL* almacena la posición del TCP cuando el valor de la señal digital pasa a ser la requerida, según se indica en la Figura 1.

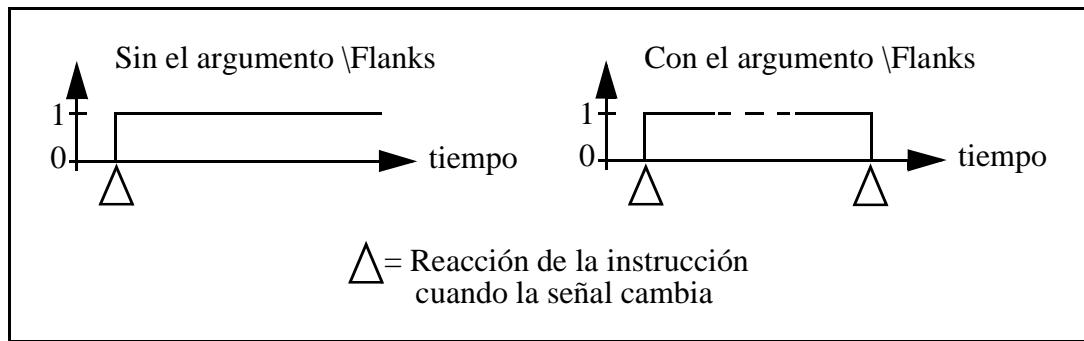


Figura 1 Detección de señal disparada por el flanco (la posición es almacenada cuando la señal cambia por primera vez únicamente).

Para obtener una respuesta rápida, se deberá utilizar las señales de sensor *sen1*, *sen2* o *sen3* activadas por interrupciones, de la tarjeta de sistema.

Ejemplos

```
SearchL \Sup, sen1, \Flanks, polmc, p10, v100, sensor;
```

El TCP del *sensor* se moverá de forma lineal hacia la posición *p10*. Cuando el valor de la señal *sen1* pasa a ser activa o pasiva, la posición se almacenará en *polmc*. En el caso en que el valor de la señal cambie dos veces, la ejecución del programa se detendrá después de que haya terminado el proceso de búsqueda.

```
SearchL \Stop, sen1, polmc, p10, v100, herramienta1;
MoveL polmc, v100, fine, herramienta1;
PDispOn *, herramienta1;
MoveL p100, v100, z10, herramienta1;
MoveL p110, v100, z10, herramienta1;
MoveL p120, v100, z10, herramienta1;
PDispoff;
```

Al principio del proceso de búsqueda, se realizará una comprobación de la señal *sen1* y si la señal todavía tiene un valor positivo, la ejecución del programa se para. De lo contrario el TCP de la herramienta *herramienta1* se moverá de forma lineal hacia la posición *p10*. Cuando el valor de la señal *sen1* pasa a ser activa, la posición será almacenada en *polmc* y el robot regresará a este punto. Cuando se utiliza el desplazamiento del programa, el robot se moverá respecto a la posición buscada, *polmc*.

Limitaciones

Los datos de zona de la instrucción de posicionamiento que preceden a *SearchL* deberán utilizarse con mucho cuidado. El inicio de la búsqueda, es decir, cuando la señal de E/S esté lista para reaccionar, no corresponderá en este caso al punto de destino programado de la instrucción de posicionamiento anterior, sino a un punto situado en la trayectoria real del robot. En la Figura 2 a la Figura 4 se indican ejemplos de situaciones en que puede haber problemas por utilizar otros datos de zona que no sea *fine*.

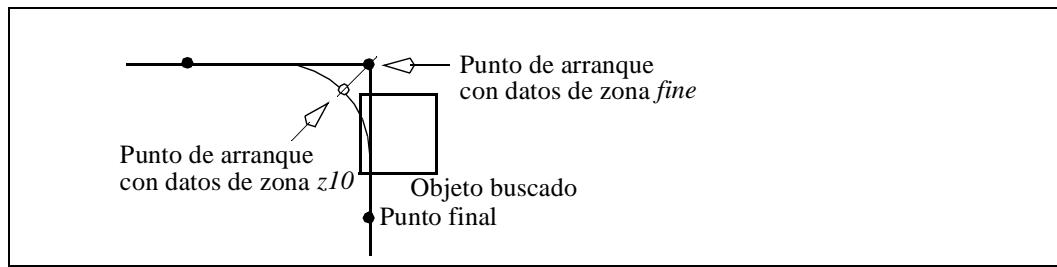


Figura 2 La detección se ha realizado en el lado incorrecto del objeto debido a la utilización de datos de zona incorrectos.

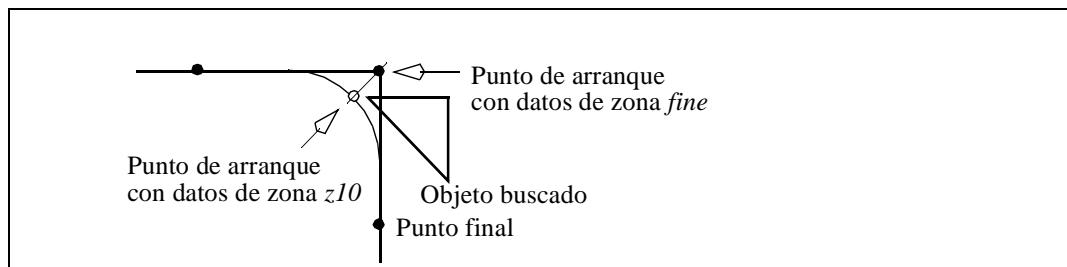


Figura 3 No se detecta el objeto debido a la utilización de datos de zona incorrectos.

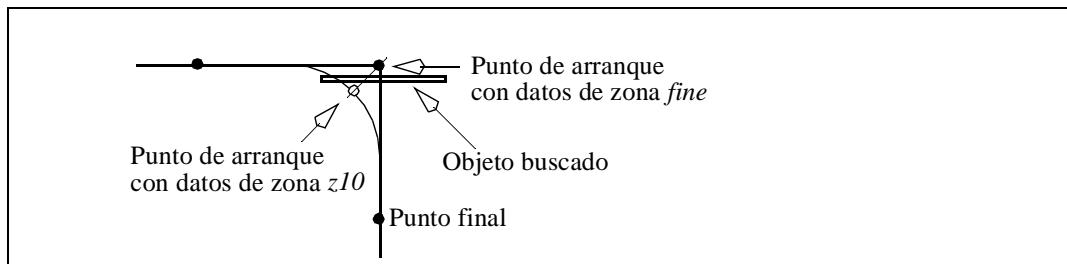


Figura 4 No se detecta el objeto debido a la utilización de datos de zona incorrectos.

La distancia de paro normal cuando se utiliza una velocidad de búsqueda de 50 mm/s es de:

- 1-3 mm sin el TCP en la trayectoria (argumento \Stop).
- 12-16 mm con el TCP en la trayectoria (argumento \PStop).

Gestión de errores

Se generará un error durante una búsqueda cuando:

- no ha ocurrido ninguna detección de señal esto genera el error ERR_WHLSEARCH.
- ha ocurrido más de una detección de señal - esto genera el error ERR_WHLSEARCH únicamente si el argumento \Sup ha sido utilizado.
- la señal ya ha generado un valor positivo al principio del proceso de búsqueda
 - esto genera el error ERR_SIGSUPSEARCH únicamente si se ha omitido el argumento \Flanks.

Los errores serán procesados de distintas maneras según el modo de funcionamiento utilizado:

Continuo hacia adelante / ERR_WHLSEARCH

No se devuelve ninguna posición y el movimiento siempre continua hacia el punto de destino programado. La variable del sistema ERRNO se activará en ERR_WHLSEARCH y el error podrá ser procesado en el gestor de errores de la rutina.

Continuo hacia adelante / Instrucción hacia adelante / ERR_SIGSUPSEARCH

No se devuelve ninguna posición y el movimiento siempre se detiene en cuanto antes al principio de la trayectoria de búsqueda. La variable del sistema ERRNO se activa en ERR_SIGSUPSEARCH y el error puede ser tratado en el gestor de errores de la rutina.

Instrucción hacia adelante / ERR_WHLSEARCH

No se devuelve ninguna posición y el movimiento siempre continúa hacia el punto de destino programado. La ejecución del programa se para con un mensaje de error.

Instrucción hacia atrás

Durante la ejecución hacia atrás, la instrucción se limita en llevar a cabo el movimiento sin realizar ninguna supervisión de señales.

Ejemplo

```
VAR num fk;  
. MoveL p10, v100, fine, herram1;  
SearchL \Stop, sen1, polmc, p20, v100, herram1;  
. ERROR  
IF ERRNO=ERR_WHLSEARCH THEN  
MoveL p10, v100, fine, herram1;  
RETRY;  
ELSEIF ERRNO=ERR_SIGSUPSEARCH THEN
```

```

TPWrite "La señal de la instrucción SearchL todavía está activada!";
TPReadFK fk,"Try again after manual reset of signal ?","YES","","","","NO";
IF fk = 1 THEN
    MoveL p10, v100, fine, tool1;
    RETRY;
ELSE
    Stop;
ENDIF
ENDIF

```

Si la señal está todavía activa al principio del proceso de búsqueda, se activará un diálogo del usuario (TPReadFK...;). Reinicializar la señal y apretar SI en el diálogo del usuario y el robot regresa a *p10* y vuelve a intentar otra vez. De lo contrario la ejecución del programa se detendrá.

Si la señal está pasiva al principio del proceso de búsqueda, el robot realiza una búsqueda a partir de la posición *p10* a *p20*. Si no ocurre ninguna detección de señal, el robot regresa a *p10* y vuelve a intentar otra vez.

Sintaxis

```

SearchL
[ '\' Stop ',' ] | [ '\' PStop ',' ] | [ '\' Sup ',' ]
[ Señal ':=' ] < variable (VAR) de signaldi > ',' 
    [ '\' Flanks ] ',' 
[ PuntBusc':=' ] < variable o persistente (INOUT) de robtarget > ',' 
[ AlPunto':=' ] < expresión (IN) de robtarget > ',' 
[ Veloc':=' ] < expresión (IN) de speeddata > 
    [ '\' V ':=' < expresión (IN) de num > ] 
    | [ '\' T ':=' < expresión (IN) de num > ] ',' 
[ Herram':=' ] < persistente (PERS) de tooldata > 
[ '\' WObj ':=' < persistente (PERS) de wobjdata > ] 
[ '\' Corr ];'

```

Información relacionada

Búsqueda circular

Escritura en una entrada de corrección

Movimiento lineal

Definición de la velocidad

Definición de las herramientas

Definición de los objetos de trabajo

Utilización de los gestores de error

Movimiento en general

Descripción:

Instrucciones - *SearchC*

Instrucciones - *CorrWrite*

Principios de Movimiento y de E/S -
*Posicionamiento durante la ejecución
del programa*

Tipos de datos - *speeddata*

Tipos de datos - *tooldata*

Tipos de datos - *wobjdata*

Resumen RAPID - *Recuperación de
errores*

Principios de Movimiento y de E/S

Set**Activación de una señal de salida digital**

Set sirve para colocar el valor de la señal de salida digital a uno.

Ejemplos

Set do15;

La señal *do15* se pondrá en 1.

Set sold;

La señal *sold* se pondrá en 1.

Argumentos**Set Señal****Señal**

Tipo de señal: *signaldo*

El nombre de la señal que se desea poner a 1.

Ejecución del programa

El valor verdadero dependerá de la configuración de la señal. En el caso en que la señal haya sido invertida en los parámetros del sistema, la instrucción hará que el canal físico se ponga en cero.

Sintaxis

Set
[Señal ':='] <variable (**VAR**) de *signaldo* > ';

Información Relacionada

Descripción:

Activación de una señal de salida digital a cero

Instrucciones - *Reiniciar*

Instrucciones de Entradas/Salidas

Resumen RAPID -
Señales de Entrada y Salida

Funciones de las Entrada/Salida en general

Principios de Movimiento y de E/S-
Principios de E/S

Configuración de las E/S

Guía del Usuario - *Parámetros del Sistema*

SetAO Cambio del valor de una señal de salida analógica

SetAO sirve para cambiar el valor de una señal de salida analógica.

Ejemplo

SetAO ao2, 5.5;

La señal *ao2* se pone a 5.5.

Argumentos

SetAO Señal Valor

Señal

Tipo de dato: *signalao*

El nombre de la señal de salida analógica que se desea cambiar.

Valor

Tipo de dato: *num*

El valor deseado de la señal.

Ejecución del programa

El valor programado es escalado (de acuerdo con los parámetros del sistema) antes de ser enviado a un canal físico. Véase la Figura 1.

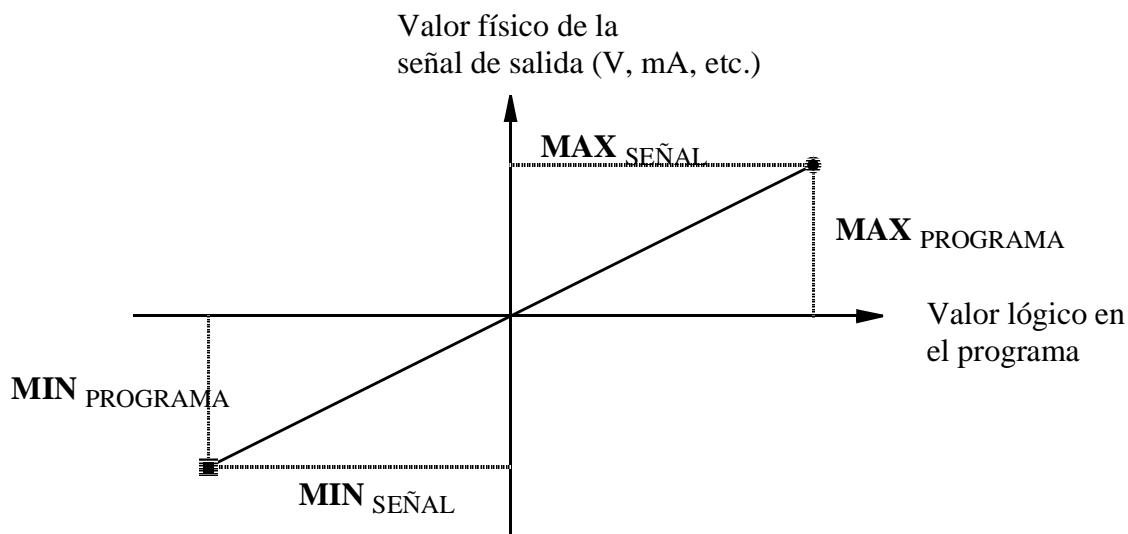


Figura 1 Diagrama indicando como se escalan los valores de las señales analógicas.

Ejemplo

```
SetAO corrsold, sal_corr;
```

La señal *corrsold* adoptará con el mismo valor que el valor utilizado de la variable *sal_corr*.

Sintaxis

SetAO

[Señal ':='] < variable (**VAR**) de *signalao* > ','
[Valor ':='] < expresión (**IN**) de *num* > ','

Información Relacionada

Descripción:

Instrucciones de las Entradas/Salidas

Resumen RAPID -
Señales de Entrada y Salida

Funciones de las Entradas/Salidas en general

Principios de Movimiento y de E/S -
Principios de E/S

Configuración de las E/S

Guía del Usuario - *Parámetros del Sistema*

SetDO Cambio del valor de una señal de salida digital

SetDO sirve para cambiar el valor de una señal de salida digital, con o sin un retraso.

Ejemplos

SetDO do15, 1;

La señal *do15* se activará en *1*.

SetDO sold, off;

La señal *sold* pasará a *off*.

SetDO \SDelay := 0,2, sold, alto;

La señal *sold* se activará en *alto* con un retraso de 0,2 s. La ejecución del programa continuará, sin embargo, con la instrucción siguiente.

Argumentos

SetDO [\SDelay] Señal Valor

[\SDelay] *(Retraso de la Señal)* Tipo de dato: *num*

Retrasa el cambio del tiempo especificado en segundos (0,1 - 32 seg.).
La ejecución del programa continúa directamente con la instrucción siguiente.
Una vez transcurrido el tiempo de retraso, la señal cambia sin afectar por ello el resto de la ejecución del programa.

Si se omite el argumento, el valor de la señal cambia directamente.

Señal Tipo de dato: *signaldo*

El nombre de la señal que se desea cambiar.

Valor Tipo de dato: *dionum*

El valor deseado de la señal.

El valor se especifica a 0 o a 1.

Ejecución del programa

El valor verdadero dependerá de la configuración de la señal. En el caso en que la señal haya sido invertida en los parámetros del sistema, el valor del canal físico será el opuesto.

Sintaxis

SetDO

['\' SDelay ':=' < expresión (**IN**) de *num* > ',']
[Señal ':='] < variable (**VAR**) de *signaldo* > ','
[Valor ':='] < expresión (**IN**) de *dionum* > ','

Información relacionada

Descripción:

Instrucciones de las Entradas/Salidas

Resumen RAPID -
Señales de Entrada y Salida

Funciones de las Entradas/Salidas en general

Principios de Movimiento y de E/S -
Principios de E/S

Configuración de las E/S

Guía del Usuario - *Parámetros del Sistema*

SetGO**Cambio del valor de un grupo
de señales de salidas digitales**

SetGO sirve para cambiar el valor de un grupo de señales de salidas digitales, con o sin un retraso de tiempo.

Ejemplo

SetGO grup2, 12;

La señal *grup2* se activará en *12*. Si *grup2* comprende 4 señales, por ejemplo, las salidas 6-9, las salidas 6 y 7 se pondrán a cero, mientras que la 8 y la 9 se activarán en 1.

SetGO \SDelay :=0,4, grup2, 10;

La señal *grup2* se activará en *10*. Si *grup2* comprende 4 señales, por ejemplo, las salidas 6-9, las salidas 6 y 8 se pondrán a cero, mientras que la 7 y la 9 se activarán en 1, con un retraso de *0,4s*. La ejecución del programa, no obstante, continuará con la instrucción siguiente.

Argumentos**SetGO [\\SDelay] Señal Valor**

[\\SDelay]	<i>(Retraso de la Señal)</i>	<i>Tipo de dato: num</i>
-------------------	------------------------------	--------------------------

Retrasa el cambio del tiempo especificado en segundos (0,1 - 32 seg.).
La ejecución del programa continúa directamente con la instrucción siguiente.
Una vez transcurrido el tiempo de retraso, el valor de las señales cambia sin afectar por ello el resto de la ejecución del programa.

Si se omite el argumento, el valor de la señal cambia directamente.

Señal		<i>Tipo de dato: signalgo</i>
--------------	--	-------------------------------

El nombre del grupo de señales que se desea cambiar.

Valor		<i>Tipo de dato: num</i>
--------------	--	--------------------------

El valor deseado para el grupo de señales (un número entero positivo).

El valor permitido depende del número de señales del grupo:

<u>Nº de señales</u>	<u>Valor permitido</u>	<u>Nº de señales</u>	<u>Valor permitido</u>
1	0 - 1	9	0 - 511
2	0 - 3	10	0 - 1023
3	0 - 7	11	0 - 2047
4	0 - 15	12	0 - 4095
5	0 - 31	13	0 - 8191
6	0 - 63	14	0 - 16383
7	0 - 127	15	0 - 32767
8	0 - 255	16	0 - 65535

Ejecución del programa

El valor programado se convertirá en un número binario sin signo. Este número binario será enviado al grupo de señales con el resultado que las señales individuales del grupo se pondrán a 0 o a 1. Debido a retrasos internos, el valor de la señal puede estar indefinida durante un corto periodo de tiempo.

Sintaxis

```
SetGO
[ '\' SDelay ':=' <expresión (IN) de num > ',' ]
[ Señal ':=' ] <variable (VAR) de signalgo > ',' 
[ Valor ':=' ] <expresión (IN) de num > ','
```

Información relacionada

Otras instrucciones de Entrada/Salida

Descrita en:

Resumen RAPID -
Señales de Entrada y Salida

Funciones de las Entradas/Salidas

Principios de Movimiento y de
E/S -
Principios de E/S

Configuración de las E/S (parámetros del sistema)

Guía del Usuario - *Parámetros
del Sistema*

SingArea

Definición de la interpolación en torno a puntos singulares

SingArea (Singularity Area) sirve para definir como el robot debe moverse en la proximidad de puntos singulares.

Ejemplos

`SingArea \Wrist;`

La orientación de la herramienta puede ser modificada ligeramente para evitar un punto singular (eje 4 y 6 alineados).

`SingArea \Off;`

La orientación de la herramienta no puede diferir de la orientación programada. Si el robot pasa por un punto singular, uno o más ejes pueden realizar un cambio de orientación, originando así una reducción de la velocidad.

Argumentos

SingArea [\Wrist] | [\Off]

[\Wrist]

Tipo de dato: *switch*

Se tolera una pequeña diferencia en la orientación de la herramienta a fin de evitar la singularidad de la muñeca. Se usa cuando los ejes 4 y 6 están alineados (eje 5 a 0 grados).

[\Off]

Tipo de dato: *switch*

No se tolera ninguna diferencia en la orientación de la herramienta. Se usa cuando no se pasa ningún punto singular o cuando no se permite que la orientación sea cambiada en puntos singulares.

Ejecución del programa

Si se especifica el argumento `\Wrist`, la orientación es interpolada eje a eje para evitar los puntos singulares. De esta forma, el TCP sigue la trayectoria correcta aunque la orientación de la herramienta se desvíe un poco. Esto ocurrirá también cuando no se pase por un punto singular.

La interpolación especificada se aplica a todos los movimientos siguientes hasta que se ejecute una nueva instrucción *SingArea*.

El movimiento se verá afectado únicamente cuando se ejecuta una interpolación lineal o circular.

Por defecto, o si no se especifica ningún argumento, la ejecución del programa utiliza automáticamente el argumento */Off*. Esto se activa automáticamente

- a la puesta en marcha
- cuando se carga un programa nuevo
- cuando se ejecuta el programa desde el principio.

Limitaciones

Sólo se podrá especificar un argumento.

Sintaxis

`SingArea
[\' Wrist] | [\' Off] ;'`

Información Relacionada

Descripción:

Singularidad

Principios de Movimiento y de E/S -
Singularidad

Interpolación

Principios de Movimiento y de E/S -
*Posicionamiento durante la Ejecución
del Programa*

SoftAct**Activación del servo suave**

SoftAct (Soft Servo Activate) sirve para activar lo que se denomina servo «suave» en cualquiera de los ejes del robot.

Ejemplo

SoftAct 3, 20;

Activación del servo suave del eje 3, con un valor de suavidad del 20%.

SoftAct 1, 90 \Ramp:=150;

Activación del servo suave del eje 1, con un valor de suavidad del 90% y un factor de rampa del 150%.

Argumentos

SoftAct Eje Suavidad [\Rampa]

Eje

Tipo de dato: *num*

Es el número del eje utilizado en el robot (1 - 6).

Suavidad

Tipo de dato: *num*

Es el valor de suavidad en porcentaje (0 - 100%). El 0% indica una suavidad mínima (rigidez máxima) y el 100 % indica una suavidad máxima.

Rampa

Tipo de dato: *num*

Es el Factor Rampa en porcentaje ($\geq 100\%$). El factor rampa sirve para controlar el grado de implicación del servo suave. Un factor del 100% denota el valor normal; si se utilizan valores más elevados, significa que el servo funcionará más lentamente (rampa más larga). El valor por defecto del factor rampa es del 100 %.

Ejecución del programa

La suavidad es activada en el valor especificado para el eje utilizado. El valor de suavidad es válido para todos los movimientos, hasta que se programe un valor de suavidad nuevo para el eje utilizado o hasta que el servo suave sea desactivado por una instrucción.

Limitaciones

El mismo eje no deberá ser activado dos veces, a menos que se encuentre una instrucción de movimiento entre ellas. Así, la secuencia siguiente del programa deberá ser evitada a fin de no provocar una sacudida en el movimiento del robot:

```
SoftAct    n , x ;
SoftAct    n , y ;
(n = eje de robot n, x e y valores de suavidad)
```

Sintaxis

```
SoftAct
[Eje ':=' ] <expresión (IN) de num> ',' 
[Suavidad ':=' ] <expresión (IN) de num>
[ '\'Rampa ':=' <expresión (IN) de num> ]';'
```

Información relacionada

Descripción:

Comportamiento con el servo suave activado

Principios de movimiento y de E/S -
*Posicionamiento durante la ejecución
de los movimientos del programa*

SoftDeact

Desactivación del servo suave

SoftDeact (Soft Servo Deactivate) sirve para desactivar lo que se denomina servo «suave» en todos los ejes del robot.

Ejemplo

SoftDeact;

Desactivación del servo suave en todos los ejes.

SoftDeact \Ramp:=150;

Desactivación del servo suave en todos los ejes, con un factor de rampa del 150%

Argumentos

SoftDeact [\Ramp]

Ramp

Tipo de dato: *num*

Es el factor rampa expresado en un porcentaje ($\geq 100\%$). El factor rampa sirve para controlar la desactivación del servo suave. Un factor del 100% denota el valor normal; con valores más elevados, el servo suave es desactivado más lentamente (rampa más larga). El valor por defecto del factor de rampa es del 100%

Ejecución del programa

El servo suave es desactivado en todos los ejes.

Sintaxis

SoftDeact';'
[‘\’Ramp ‘:=’ <expresión (IN) de num] ‘;’

Información relacionada

Descripción:

Activación del servo suave

Instrucciones - *SoftAct*

StartMove

Rearranque del movimiento del robot

StartMove sirve para reanudar el movimiento del robot y de los ejes externos después de un paro ejecutado mediante la instrucción *StopMove*.

Ejemplo

```
StopMove;  
WaitDI entrada_preparado, 1;  
StartMove;
```

El robot empezará a moverse de nuevo cuando la *entrada_preparado* se activa.

Ejecución del programa

Cualquier proceso asociado con el movimiento parado será rearrancado al mismo tiempo que rearanca el movimiento.

Gestión de errores

En el caso en que el robot esté demasiado lejos de la trayectoria (más de 10 mm o 20 grados) como para realizar un arranque del movimiento interrumpido, la variable del sistema *ERRNO* se activará en *ERR_PATHDIST*. Este error podrá entonces ser manipulado por el gestor de errores.

Sintaxis

```
StartMove';'
```

Información relacionada

Descripción:

Paro de movimientos
Más ejemplos

Instrucciones - *StopMove*
Instrucciones - *StorePath*

StartMove

Instrucciones

Stop**Paro de la ejecución del programa**

Stop sirve para detener temporalmente la ejecución del programa.

La ejecución del programa también podrá ser detenida utilizando la instrucción *EXIT*. Esto, no obstante, sólo deberá realizarse si se ha terminado una tarea, o si se produce un error muy grave, ya que la ejecución del programa no podrá ser rearrancada con *EXIT*.

Ejemplo

TPWrite "La línea con el computador principal se ha interrumpido";
Stop;

La ejecución del programa será detenida después de que aparezca un mensaje en la unidad de programación.

Argumentos

Stop [\NoRegain]

[\NoRegain]

Tipo de dato: *switch*

Este argumento especifica para el siguiente arranque del programa en el modo manual, si el robot y los ejes externos deben o no regresar a la posición de paro. En el modo automático el robot y los ejes externos siempre regresan a la posición de paro.

En el caso en que el argumento *NoRegain* esté activado, el robot y los ejes externos no regresarán a la posición de paro (si han sido desviados de dicha posición).

En el caso en que se omita el argumento y que el robot o los ejes externos hayan sido desviados de la posición de paro, el robot visualizará una pregunta en la unidad de programación. Entonces, el usuario deberá contestar a la pregunta especificando si desea que el robot regrese a la posición de paro o no.

Ejecución del programa

La instrucción detendrá la ejecución del programa en cuanto el robot y los ejes externos alcancen el punto de destino programado del movimiento que se está realizando en el momento. La ejecución del programa podrá entonces ser rearrancada a partir de la siguiente instrucción.

Si hay una instrucción de *Paro* en alguna rutina de evento, la rutina será ejecutada a partir del principio del siguiente evento.

Ejemplo

```
MoveL p1, v500, fine, tool1;  
TPWrite "Mover el robot a la posición para esquina pallet 1";  
Stop \NoRegain;  
p1_read := CRobT();  
MoveL p2, v500, z50, tool1;
```

La ejecución del programa se detiene con el robot situado en la posición *p1*. El operador mueve el robot a *p1_read*. Para el siguiente arranque de programa, el robot no regresará a *p1*, ya que la posición *p1_read* puede ser almacenada en el programa.

Limitaciones

La instrucción de movimiento que precede esta instrucción deberá terminar con un punto de paro para que en esta instrucción sea posible realizar un rearranque después de un corte de potencia.

Sintaxis

```
Stop';'  
[ '\' NoRegain ]';'
```

Información relacionada

Paro después de un error grave
Paro de la ejecución del programa
Paro de los movimientos del robot

Descripción:

Instrucciones - *EXIT*
Instrucciones - *EXIT*
Instrucciones - *StopMove*

StopMove Paro del movimiento del robot

StopMove sirve para detener los movimientos del robot y de los ejes externos de forma temporal. Si se ejecuta la instrucción *StartMove*, el movimiento será reanudado.

Esta instrucción podrá, por ejemplo, utilizarse en una rutina de tratamiento de interrupciones para detener el robot de forma temporal cuando ocurre una interrupción.

Ejemplo

```
StopMove;  
WaitDI entrada_preparado, 1;  
StartMove;
```

El movimiento del robot será detenido hasta que la *entrada_preparado* sea activada.

Ejecución del programa

Los movimientos del robot y de los ejes externos se detendrán aunque los frenos estén aplicados. Cualquier proceso asociado con el movimiento en curso quedará detenido al mismo tiempo que se detiene el movimiento.

La ejecución del programa continúa sin esperar que el robot y los ejes externos se detengan (estén inmovilizados).

Ejemplos

```
VAR intnum intno1;  
...  
CONNECT intno1 WITH go_to_home_pos;  
ISignalDI di1,1,intno1;  
  
TRAP go_to_home_pos  
  VAR robtarget p10;  
  
  StopMove;  
  StorePath;  
  p10:=CRobT();  
  MoveL home,v500,fine,tool1;  
  WaitDI di1,0;  
  Move L p10,v500,fine,tool1;  
  RestoPath;  
  StartMove;
```

ENDTRAP

Cuando la entrada *di1* está activada en 1, se activa una interrupción que a su vez activa la rutina de interrupciones *go_to_home_pos*. El movimiento actual es parado inmediatamente y el robot se mueve a la posición de inicio *home*. Cuando *di1* está activada en 0, el robot regresa a la posición donde ocurrió la interrupción y continúa a moverse a lo largo de la trayectoria programada.

```
VAR intnum intno1;  
...  
CONNECT intno1 WITH go_to_home_pos;  
ISignalDI di1,1,intno1;  
  
TRAP go_to_home_pos ()  
    VAR robtarget p10;  
  
    StorePath;  
    p10:=CRobT();  
    MoveL home,v500,fine,tool1;  
    WaitDI di1,0;  
    Move L p10,v500,fine,tool1;  
    RestoPath;  
    StartMove;  
ENDTRAP
```

Este ejemplo es similar al anterior, pero el robot no se mueve a la posición inicial *home* hasta que la instrucción de movimiento actual haya acabado.

Sintaxis

StopMove';'

Información relacionada

Continuar un movimiento
Interrupciones

Descripción:

Instrucciones - *StartMove*
Resumen RAPID - *Interrupciones*
Características Básicas - *Interrupciones*

StorePath

Almacenamiento de una trayectoria cuando se produce una interrupción

StorePath sirve para almacenar la trayectoria de movimiento que se está ejecutando en el momento en que se produce un error o una interrupción. El manipulador de errores o la rutina de tratamiento de interrupciones podrá entonces iniciar un movimiento nuevo y, a partir de aquí, rearrancar el movimiento que ha sido almacenado anteriormente.

Esta instrucción podrá utilizarse para ir a una posición de servicio o para limpiar la pinza, por ejemplo, cuando se produce un error.

Ejemplo

StorePath;

La trayectoria de movimiento actual será almacenada para ser utilizada en una etapa posterior.

Ejecución del programa

La trayectoria de movimiento utilizada del robot y de los ejes externos será guardada. Después de esto, se podrá iniciar otro movimiento en una rutina de tratamiento de interrupciones o en un manipulador de errores. Cuando el motivo que ha provocado el error o la interrupción haya sido resuelto, la trayectoria de movimiento almacenada podrá ser rearrancada.

Ejemplo

```
TRAP máquina_OK;  
  VAR robtarget p1;  
  StopMove;  
  StorePath;  
  p1 := CRobT();  
  MoveL p100, v100, fine, herramienta1;  
  ...  
  MoveL p1, v100, fine, herramienta1;  
  RestoPath;  
  StartMove;  
ENDTRAP
```

Cuando ocurre una interrupción que activa la rutina de tratamiento de interrupciones *máquina_OK*, la trayectoria de movimiento que el robot está ejecutando en este momento será parada al final de la instrucción (AlPunto) y almacenada. Después de esto, el robot resolverá el problema que ha provocado la interrupción, reemplazando por ejemplo, una pieza de la máquina y el movimiento normal será rearrancado.

Limitaciones

Sólo se podrá almacenar una trayectoria de movimiento a la vez.

Sintaxis

StorePath‘;’

Información relacionada

Descripción:

Restaurar una trayectoria

Instrucciones - *RestoPath*

Más ejemplos

Instrucciones - *RestoPath*

TEST**Dependiendo del valor de una expresión...**

TEST sirve cuando diferentes instrucciones deben ser ejecutadas dependiendo del valor de una expresión o de un dato.

En el caso en que no haya demasiadas alternativas, se podrá usar también la instrucción **IF..ELSE**.

Ejemplo

```
TEST reg1
CASE 1,2,3 :
    rutina1;
CASE 4 :
    rutina2;
DEFAULT :
    TPWrite "Elección ilegal";
    Stop;
ENDTEST
```

Diferentes instrucciones serán ejecutadas dependiendo del valor de *reg1*. En el caso en que el valor sea 1-3, la *rutina1* será ejecutada. Si el valor es 4, la *rutina2* será ejecutada. De lo contrario, aparecerá un mensaje de error visualizado y la ejecución se detendrá.

Argumentos

TEST Dato de Test {CASE Valor de Test {, Valor de Test} :...} [DEFAULT: ...] ENDTEST

Dato de Test

Tipo de dato: Todos

El dato o expresión con el que se desea que el valor test sea comparado.

Valor de Test

Tipo de dato: El mismo que el anterior

El valor que el dato test debe tener para que las instrucciones asociadas se ejecuten.

Ejecución del programa

El dato test será comparado con los valores test de la primera condición CASE. En el caso en que la comparación sea verdadera, las instrucciones asociadas se ejecutarán. Después de ello, la ejecución del programa continúa con la instrucción que sigue a ENDTEST.

Si la primera condición CASE no se cumple, se comprobarán las otras condiciones CASE y así, sucesivamente. En el caso en que no se cumpla ninguna de las condiciones, las instrucciones asociadas con DEFAULT (si existe) serán ejecutadas.

Sintaxis

(EBNF)

TEST <expresión>
{ (**CASE** <valor de test> { ',' <valor de test> } ':'
 <lista instrucciones>) | <**CSE**> }
[**DEFAULT** ':' <lista instrucciones>]
ENDTEST

<valor de test> ::= <expresión>

Información Relacionada

Expresiones

Descripción:

Características Básicas - *Expresiones*

TPErase**Borrado del texto impreso en la unidad de programación**

TPErase (Teach Pendant Erase) sirve para borrar el contenido del visualizador de la unidad de programación.

Ejemplo

TPErase;
TPWrite "Ejecución iniciada";

El contenido del visualizador de la unidad de programación será borrado antes de que aparezca *Ejecución iniciada*.

Ejecución del programa

El visualizador de la unidad de programación quedará libre de todo texto. La próxima vez que se introduzca texto, aparecerá en la línea superior del visualizador.

Sintaxis

TPErase;

Información Relacionada

Descripción:

Escribir en la unidad de programación

Resumen RAPID - *Comunicación*

TPReadFK Lectura de las teclas de función

TPReadFK (Teach Pendant Read Function Key) sirve para introducir texto encima de las teclas de función y para descubrir la tecla que ha sido pulsada.

Ejemplo

```
TPReadFK reg1, "Más?", "", "", "", "Sí", "No";
```

El texto *Más?* aparecerá en el visualizador de la unidad de programación y las teclas de función 4 y 5 se activarán mediante las cadenas de texto *Sí* y *No* respectivamente (véase la Figura 1). La ejecución del programa esperará hasta que se pulse una de las teclas de función, 4 o 5. En otras palabras, *reg1* tendrá asignado el 4 o el 5 dependiendo de la tecla que se pulse.

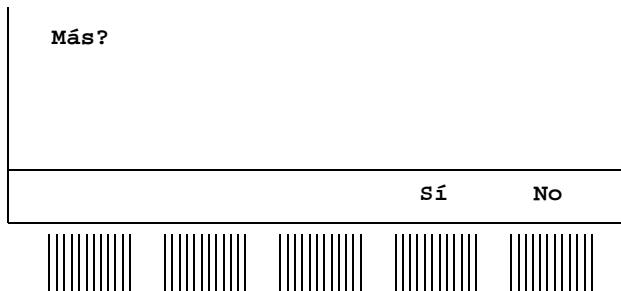


Figura 1 El usuario podrá introducir información mediante las teclas de función.

Argumentos

TPReadFK Contestación Texto FK1 FK2 FK3 FK4 FK5 [\\Max-Time] [DIBreak] [\\BreakFlag]

Contestación

Tipo de dato: *num*

La variable para la cual, dependiendo de la tecla que se pulse, se obtiene el valor numérico 1..5. Si se pulsa la tecla de función 1, se obtendrá 1 y así sucesivamente.

Texto

Tipo de dato: *string*

El texto informativo que se desea introducir en el visualizador (de 80 caracteres como máximo).

FKx

(*Texto de la tecla de función*) Tipo de dato: *string*

El texto que se desea introducir como indicación propia de la tecla de función (de 7 caracteres como máximo). FK1 es la tecla situada más a la izquierda.

Las teclas de función desprovistas de cualquier indicación están especificadas

con una cadena vacía "".

[\\MaxTime]

Tipo de dato: *num*

Es el intervalo máximo de tiempo [s] que la ejecución del programa espera. Si no se pulsa ninguna tecla de función durante este tiempo, el programa continua a ejecutarse en el gestor de errores, a menos que se utilice *BreakFlag* (véase a continuación). La constante *ERR_TP_MAXTIME* podrá ser utilizada para comprobar si el intervalo máximo de tiempo ha transcurrido o no.

[\\DIBreak]

(*Interrupción Entrada Digital*) Tipo de dato: *signaldi*

Es la señal digital que podría interrumpir el diálogo del usuario. Si no se pulsa ninguna tecla de función cuando la señal se activa en el valor 1 (o si ya está activada en 1), el programa continuará a ejecutarse en el gestor de errores, a menos que se utilice *BreakFlag* (véase a continuación). La constante *ERR_TP_DIBREAK* podrá ser utilizada para comprobar si esto ha ocurrido o no.

[\\BreakFlag]

Tipo de dato: *errnum*

Es una variable que mantiene el código de error si se utiliza maxtime o dibreak. Si se omite esta variable opcional, el gestor de errores será ejecutado. Las constantes *ERR_TP_MAXTIME* y *ERR_TP_DIBREAK* podrán utilizarse para seleccionar el motivo.

Ejecución del programa

El texto de información aparece siempre escrito en una nueva línea. Si el visualizador está lleno, este texto se moverá de una línea hacia arriba. Las cadenas de caracteres más largas que la anchura de la unidad de programación (40 caracteres) serán divididas en dos líneas.

El texto indicativo estará escrito encima de las teclas de función correspondientes. Las teclas de función que no lleven ninguna indicación serán desactivadas.

La ejecución del programa esperará hasta que se pulse una de las teclas de función activadas.

Descripción de la petición concurrente *TPReadFK* o *TPReadNum* en la unidad de programación (petición de la unidad de programación) a partir de la misma o de otras tareas de programa:

- La petición nueva de la unidad de programación procedente de otra tarea de programa no está accesible en este momento (la petición nueva queda retenida en la cola)
- La petición nueva de la unidad de programación procedente de una rutina de tratamiento de interrupciones en la misma tarea de programa está accesible en este momento (la petición antigua queda retenida en la cola)
- El paro de programa está accesible en este momento (el antiguo queda retenido en la cola)
- La petición nueva de la unidad de programación en el estado de paro de programa está accesible en este momento (la antigua queda retenida en la cola).

Ejemplo

```

VAR errnum errvar;
...
TPReadFK reg1, "¿Ir a la posición de servicio?", "", "", "", "Yes", "No" \Max-
Time:=
          600 \DIBreak:= di5\BreakFlag:= errvar;
IF reg1 = 4 or OR errvar = ERR_TP_DIBREAK THEN
    MoveL service, v500, fine, tool1;
    Stop;
END IF
IF errvar = ERR_TP_MAXTIME EXIT;

```

El robot se mueve a la posición de servicio si la cuarta tecla de función ("Sí") está pulsada, o si se activa la entrada 5. Si no se da ninguna respuesta en los 10 minutos siguientes, la ejecución se termina.

Sintaxis

```

TPReadFk
[Contestación':=] <variable o persistente (INOUT) de num>,
[Texto':=] <expresión (IN) de string>,
[FK1 ':=] <expresión (IN) de string>,
[FK2 ':=] <expresión (IN) de string>,
[FK3 ':=] <expresión (IN) de string>,
[FK4 ':=] <expresión (IN) de string>,
[FK5 ':=] <expresión (IN) de string>
['\MaxTime ':= <expresión (IN) de num>]
['\DIBreak ':= <variable (VAR) de signaldi>];
['\BreakFlag ':= <variable o persistente (INOUT) de errnum>];

```

Información relacionada

Descripción:

Lectura y escritura a partir de la unidad de programación

Respuestas a través de la unidad de programación

Resumen RAPID - *Comunicación*

Funcionamiento de producción

TPReadNum

Lectura de un número en la unidad de programación

TPReadNum (Teach Pendant Read Numerical) sirve para la lectura de un número en la unidad de programación.

Ejemplo

TPReadNum reg1, “¿Cuántas unidades deben producirse? “;

El texto *¿Cuántas unidades deben producirse?* aparece en el visualizador de la unidad de programación. La ejecución del programa espera hasta que se haya introducido un número a partir del teclado numérico de la unidad de programación. Este número quedará almacenado en *reg1*.

Argumentos

**TPReadNum Contestación Texto [\\MaxTime] [DIBreak]
[\\BreakFlag]**

Contestación

Tipo de dato: *num*

Es la variable que es devuelta para el número introducido en la unidad de programación.

Texto

Tipo de dato: *string*

Es el texto de información que se desea introducir en la unidad de programación (de 80 caracteres como máximo).

[\\MaxTime]

Tipo de dato: *num*

Es el intervalo máximo de tiempo que la ejecución del programa espera. Si no se introduce ningún número durante este tiempo, el programa continúa a ejecutarse en el gestor de errores a menos que se haya utilizado *BreakFlag* (véase a continuación). La constante *ERR_TP_MAXTIME* podrá ser utilizada para comprobar si el intervalo máximo de tiempo ha transcurrido o no.

[\\DIBreak]

(*Interrupción Entrada Digital*) Tipo de dato: *signaldi*

Es la señal digital que podría interrumpir el diálogo del usuario. Si no se introduce ningún número cuando la señal se activa en 1 (o si ya está activada en 1), el programa continúa ejecutándose en el gestor de errores a menos que se haya utilizado *BreakFlag* (véase a continuación). La constante *ERR_TP_DIBREAK* podrá ser utilizada para comprobar si ello ha ocurrido o no.

[BreakFlag]

Tipo de dato: *errnum*

Es una variable que mantiene el código de error si se ha utilizado *maxtime* o *dibreak*. Si se omite esta variable opcional, el gestor de error será ejecutado. Las constantes **ERR_TP_MAXTIME** y **ERR_TP_DIBREAK** podrán utilizarse para seleccionar el motivo.

Ejecución del programa

El texto de información siempre aparece en una línea nueva. Si el visualizador está lleno, el texto se moverá de una línea hacia arriba. Las cadenas de caracteres que son más largas que la anchura de la unidad de programación quedarán divididas (40 caracteres) en dos líneas.

La ejecución del programa esperará hasta que se teclee un número en el teclado numérico (pulsando luego la tecla Retorno o la tecla **OK**).

Véase la instrucción *TPReadFK* referente a la descripción de una petición concurrente *TPReadFK* o *TPReadNum* en la unidad de programación, procedente de la misma tarea de programa o de otras tareas.

Ejemplo

```
TPReadNum reg1, “¿Cuántas unidades deben producirse?“;
FOR i FROM 1 TO reg1 DO
    produc_unid;
ENDFOR
```

El texto *¿Cuántas unidades deben producirse?* aparece escrito en el visualizador de la unidad de programación. La rutina *produc_unid* se repetirá entonces el número de veces que se ha indicado a través de la unidad de producción.

Sintaxis

```
TPReadNum
[Contestación':='] <variable o persistente (INOUT) de num>,’
[Texto':='] <expresión (IN) de string>
[’\MaxTime ':=’ <expresión (IN) de num>]
[’\DIBreak ':=’ <variable (VAR) de signaldi>]
[’\BreakFlag ':=’ <variable o persistente (INOUT) de errnum>] ’;
```

Información relacionada

Lectura y escritura en la unidad de programación

Introducción de un número en la unidad de programación

Ejemplos sobre como utilizar los argumentos MaxTime, DIBreak y BreakFlag

Descripción:

Resumen RAPID - *Comunicación*

Guía del Usuario - *Proceso de producción*

Instrucciones - *TPReadFK*

TPReadNum

Instrucciones

TPShow

Cambio de ventana de la unidad de programación

TPShow (Teach Pendant Show) sirve para seleccionar la ventana de la unidad de programación desde aplicaciones en RAPID.

Ejemplos

TPShow TP_PROGRAM;

La *Ventana de Producción* estará activa si el sistema se encuentra en el modo *AUTO* y la *Ventana de Programa* estará activa si el sistema está en el modo *MAN* después de la ejecución de esta instrucción.

TPShow TP_LATEST;

La última ventana utilizada de la unidad de programación antes de la *Ventana de Entradas&Salidas del Operador*, estará activa después de la ejecución de esta instrucción.

Argumentos

TPShow Ventana

Ventana

Tipo de dato: *tpnum*

La ventana que se desea visualizar:

TP_PROGRAM = *Ventana de Producción* si está en el modo *AUTO*. *Ventana de Programa* si está en el modo *MAN*.

TP_LATEST = Última ventana utilizada en la unidad de programación antes de *Ventana de Entrada&Salidas del Operador*.

Datos predefinidos

CONST tpnum TP_PROGRAM := 1;
CONST tpnum TP_LATEST := 2;

Ejecución del programa

La ventana seleccionada de la unidad de programación será activada.

Sintaxis

TPShow
[Ventana':='] <expresión (**IN**) de *tpnum*> ‘;’

Información relacionada

Descripción en:

Comunicación mediante la unidad de programación

Resumen RAPID - *Comunicación*

Número de Ventana de la unidad de programación

Tipos de dato - *tpnum*

-

TPWrite

Escritura en la unidad de programación

TPWrite (*Teach Pendant Write*) sirve para escribir un texto en la unidad de programación. El valor de ciertos datos así como texto podrán ser introducidos.

Ejemplos

TPWrite "Ejecución iniciada";

El texto *Ejecución iniciada* aparecerá visualizada en la unidad de programación.

TPWrite "Nº de piezas producidas="|Num:=reg1;

Por ejemplo, a la respuesta a *Nº de piezas producidas*= 5, se deberá introducir 5 en vez de *reg1* en la unidad de programación.

Argumentos

TPWrite **Texto** [**Num**] | [**Bool**] | [**\Pos**] | [**\Orient**]

La cadena de texto que se debe escribir (80 caracteres como máximo).

[Num] (Numeric) Tipo de dato: *num*

El dato cuyo valor numérico debe ser introducido después de la cadena de texto.

[Bool] *(Boolean)* Tipo de dato: *bool*

El dato cuyo valor lógico debe ser introducido después de la cadena de texto.

[|Pos] *(Position)* Tipo de dato: *pos*

El dato cuya posición debe ser introducida después de la cadena de texto.

[Orient] *(Orientation)* Tipo de dato: *orient*

El dato cuya orientación debe ser introducida después de la cadena de texto.

Ejecución del programa

El texto introducido en la unidad de programación empieza siempre en una línea nueva. Cuando el visualizador está lleno, el texto se moverá de una línea hacia arriba. Las cadenas de texto que son más largas que la anchura de la unidad de programación (40 caracteres) se dividirán en dos líneas.

Si se utiliza uno de los argumentos `\Num`, `\Bool`, `\Pos` o `\Orient`, su valor será primero convertido en una cadena de texto antes de ser añadido a la primera cadena. La conversión del valor en cadena de texto se realiza según lo siguiente:

<u>Argumento</u>	<u>Valor</u>	<u>Cadena de texto</u>
<code>\Num</code>	23	"23"
<code>\Num</code>	1.141367	"1.141367"
<code>\Bool</code>	VERDADERO	"VERDADERO"
<code>\Pos</code>	[1817.3,905.17,879.11]	"[1817.3,905.17,879.11]"
<code>\Orient</code>	[0.96593,0,0.25882,0]	"[0.96593,0,0.25882,0]"

El valor será convertido en una cadena con el formato estándar en RAPID. Esto significa en principio 6 cifras significantes. Si el número decimal es inferior a 0,000005 o mayor que 0,999995, el número será redondeado a un número entero.

Limitaciones

Los argumentos `\Num`, `\Bool`, `\Pos` y `\Orient` son mutuamente exclusivos y por lo tanto no podrán ser utilizados simultáneamente en la misma instrucción.

Sintaxis

```
TPWrite
[Texto':='] <expresión (IN) de string>
['\Num':='] <expresión (IN) de num>
| ['\Bool':='] <expresión (IN) de bool>
| ['\Pos':='] <expresión (IN) de pos>
| ['\Orient':='] <expresión (IN) de orient> ]';
```

Información relacionada

Borrado y lectura en la unidad de programación

Descripción:

Resumen RAPID - Comunicación

TriggC**Movimiento circular del robot con eventos**

TriggC (Trigg Circular) sirve para activar señales de salida y/o ejecutar rutinas de interrupción en posiciones fijas, al mismo tiempo que el robot se mueve en una trayectoria circular.

Mediante las instrucciones *TriggIO*, *TriggEquip* o *TriggInt* se podrán definir uno o más (4 como máximo) eventos. A continuación, estas definiciones serán referidas en la instrucción *TriggC*.

Ejemplos

```
VAR triggdata gunon;
TriggIO gunon, 0 \Start \DOp:=gun, on;
MoveL p1, v500, z50, gun1;
TriggC p2, p3, v500, gunon, fine, gun1;
```

La señal de salida digital *gun* se activa cuando el TCP del robot pasa en el punto que se encuentra en el medio de la trayectoria esquina del punto *p1*.

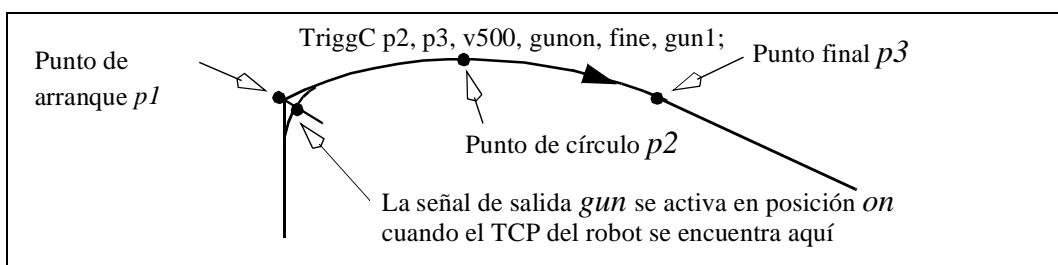


Figura 1 Ejemplo de un evento de E/S de posición fija.

Argumentos

TriggC [**\Conc**] **PuntoCir** **AlPunto** **Veloc** [**\T**]
Trigg1 [**\T2**] [**\T3**] [**\T4**] **Zona** **Herram** [**\WObj**]
[Corr]

[**\Conc**]

(Concurrente)

Tipo de dato: *switch*

Las instrucciones lógicas siguientes son ejecutadas inmediatamente. Este argumento sirve para acortar el tiempo de ciclo cuando, por ejemplo, se está comunicando con el equipo externo y no se requiere ninguna sincronización. También podrá utilizarse para ajustar la ejecución de la trayectoria del robot, para evitar el mensaje de aviso 50024 Fallo de trayectoria esquina o el error 40082 Límite

de deceleración.

Cuando se usa el argumento \Conc, el número de instrucciones de movimiento sucesivas será limitado a 5. En una sección de programa que incluye una secuencia StorePath-RestoPath, las instrucciones de movimiento con el argumento \Conc no están permitidas.

Si se omite este argumento y que AlPunto no es un punto de paro, la instrucción siguiente se ejecutará poco tiempo antes de que el robot haya alcanzado el punto de paro especificado o 100 ms antes de la zona especificada.

PuntoCir

Tipo de dato: *robtarget*

Es el punto circular del robot. Véase la instrucción *MoveC* para una descripción más detallada del movimiento circular. El punto circular está definido como una posición nombrada o es almacenado directamente en la instrucción (marcado con un asterisco * en la instrucción).

AlPunto

Tipo de dato: *robtarget*

Es el punto de destino del robot y de los ejes externos. Esta definido como una posición nombrada o es almacenado directamente en la instrucción (marcado con un asterisco * en la instrucción).

Veloc

Tipo de dato: *speeddata*

Son los datos de velocidad que se aplican a los movimientos. Los datos de velocidad definen la velocidad del punto central de la herramienta, de los ejes externos y de la reorientación de la herramienta.

[\T]

(*Tiempo*)

Tipo de dato: *num*

Este argumento sirve para especificar el tiempo total en segundos durante el cual el robot se mueve. Más adelante será sustituido por el dato de velocidad correspondiente.

Trigg1

Tipo de dato: *triggdata*

Es una variable que se refiere a las condiciones y actividades de disparo, definidas anteriormente en el programa mediante la utilización de las instrucciones *TriggIO*, *TriggEquip* o *TriggInt*.

[\T2]

(*Trigg 2*)

Tipo de dato: *triggdata*

Es una variable que se refiere a las condiciones y actividades de disparo, definidas anteriormente en el programa mediante la utilización de las instrucciones *TriggIO*, *TriggEquip* o *TriggInt*.

[\T3]

(*Trigg 3*)

Tipo de dato: *triggdata*

Es una variable que se refiere a las condiciones y actividades de disparo, definidas anteriormente en el programa mediante la utilización de las instrucciones *TriggIO*, *TriggEquip* o *TriggInt*.

[\T4] *(Trigg 4)* Tipo de dato: *triggedata*

Es una variable que se refiere a las condiciones y actividades de disparo, definidas anteriormente en el programa mediante la utilización de las instrucciones *TriggIO*, *TriggEquip* o *TriggInt*.

Zona Tipo de dato: *zonedata*

Son los datos de zona para el movimiento. Los datos de zona describen el tamaño de la trayectoria esquina generada.

Herram Tipo de dato: *tooldata*

Es la herramienta utilizada cuando el robot se está moviendo. El punto central de la herramienta es el punto que es movido a la posición de destino especificada.

[\WObj] *(Objeto de trabajo)* Tipo de dato: *wobjdata*

Es el objeto de trabajo (sistema de coordenadas) al que se refiere la posición del robot en la instrucción.

Este argumento puede ser omitido, y en el caso en que lo sea, la posición se referirá al sistema de coordenadas mundo. Si, por otra parte, se está usando un TCP estacionario o unos ejes externos coordinados, este argumento deberá ser especificado para un movimiento lineal relativo al objeto de trabajo correspondiente.

[\Corr] *(Corrección)* Tipo de dato: *switch*

Si este argumento está presente, los datos de corrección introducidos en una entrada de corrección mediante la instrucción *CorrWrite* serán añadidos a la trayectoria y a la posición de destino.

Ejecución del programa

Véase la instrucción *MoveC* para más información sobre el movimiento circular.

Dado que las condiciones de disparo se han cumplido a medida que el robot se posiciona cada vez más cerca del punto final, las actividades de disparo definidas se ejecutarán. Las condiciones de disparo se cumplen a cierta distancia antes de alcanzar el punto final de la instrucción, o a cierta distancia después del punto de arranque de la instrucción, o a cierto punto definido en el tiempo (limitado a un periodo corto) antes de alcanzar el punto final de la instrucción.

Durante la ejecución paso a paso hacia adelante, las actividades de las E/S se llevarán a cabo pero las rutinas de interrupción no serán ejecutadas. Durante la ejecución paso a paso hacia atrás, no se llevará a cabo ninguna actividad de disparo.

Ejemplos

VAR intnum intno1;

```

VAR triggdata trigg1;
...
CONNECT intno1 WITH trap1;
TriggInt trigg1, 0.1 \Time , intno1;
...
TriggC p1, p2, v500, trigg1, fine, gun1;
TriggC p3, p4, v500, trigg1, fine, gun1;
...
IDelete intno1;

```

La rutina de tratamiento de interrupción *trap1* se ejecuta cuando el punto de trabajo está en una posición que se encuentra a 0,1 s antes del punto *p2* o *p4* respectivamente.

Limitaciones

En el caso en que el punto de arranque utilizado se haya desviado de su posición habitual, de forma que la longitud de posicionamiento total de la instrucción *TriggC* sea más corta que la habitual, puede ocurrir que algunas o todas las condiciones de disparo se cumplan en una única y misma posición. En tales casos, la secuencia en la que se llevan a cabo las actividades de disparo no estará definida. La lógica del programa contenida en el programa del usuario puede no estar basada en una secuencia normal de actividades de disparo para un "movimiento incompleto".

No se deberá nunca arrancar la instrucción *TriggC* desde el principio con el robot en una posición después de un punto de círculo. De lo contrario, ocurriría que el robot no emprenderá la trayectoria programada (posicionamiento en torno a una trayectoria circular en otra dirección comparada con la programada).

Sintaxis

```

TriggC
[\' Conc ',']
[PuntoCirc':='] < expresión (IN) de robtarget > ',' 
[AlPunto':='] < expresión (IN) de robtarget > ',' 
[Veloc':='] < expresión (IN) de speeddata > 
[\' T ':='] < expresión (IN) de num > ',' 
[Trigg1 ':='] < variable (VAR) de triggdata > 
[\' T2 ':='] < variable (VAR) de triggdata > ] 
[\' T3 ':='] < variable (VAR) de triggdata > ] 
[\' T4 ':='] < variable (VAR) de triggdata > ] ',' 
[Zona ':='] < expresión (IN) de zonedata > ',' 
[Herram':='] < persistente (PERS) de tooldata > 
[\' WObj ':='] < persistente (PERS) de wobjdata > ] 
[\' Corr ],'

```

Información relacionada

Movimiento lineal con disparos

Descripción en:

Instrucciones - *TriggL*

Movimiento de ejes con disparos

Instrucciones - *TriggJ*

Definición de los disparos

Instrucciones - *TriggIO, TriggEquip, TriggInt*

Escritura en una entrada de corrección

Instrucciones - *CorrWrite*

Movimiento circular

Principios de Movimiento y de E/S -
Posicionamiento durante la Ejecución del Programa

Definición de la velocidad

Tipos de datos - *speeddata*

Definición de las herramientas

Tipos de datos - *tooldata*

Definición de los objetos de trabajo

Tipos de datos - *wobjdata*

Movimiento en general

Principios de Movimiento y de E/S

TriggEquip**Definición de un evento de E/S
de tiempo-posición fijas**

TriggEquip (*Trigg Equipment*) sirve para definir las condiciones y acciones para la activación de una señal digital, de un grupo de señales digitales o de una señal de salida analógica en una posición fija en la trayectoria de movimiento del robot con la posibilidad de llevar a cabo una compensación de tiempo para el retraso ocurrido en el equipo externo.

Los datos definidos serán utilizados en una o más instrucciones *TriggL*, *TriggC* o *TriggJ* siguientes.

Ejemplos

```
VAR triggdata gunon;
```

```
TriggEquip gunon, 10, 0.1 \DOp:=gun, 1;
```

```
TriggL p1, v500, gunon, z50, gun1;
```

La herramienta *gun1* se abre en el punto *p2*, cuando el TCP está a 10 mm antes del punto *p1*. Para alcanzar esto, la señal de salida digital *gun* está puesta en el valor 1, cuando el TCP está a 0,1 seg antes del punto *p2*. La pinza está totalmente abierta cuando el TCP alcanza el punto *p2*.

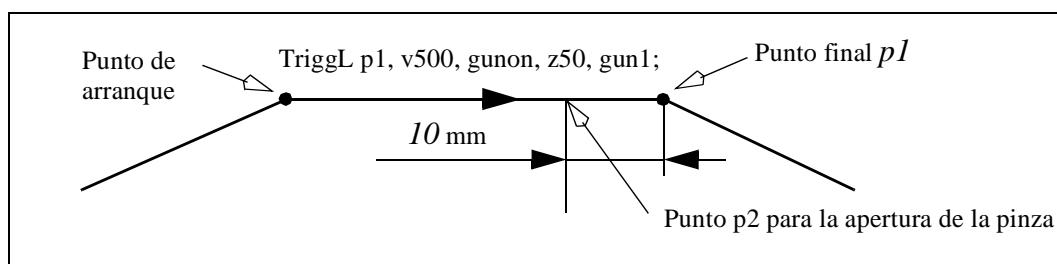


Figura 1 Ejemplo de un evento de E/S de tiempo-posición fijas.

Argumentos

**TriggEquip TrggData Distancia [\Start] EquipLag
[\DOp] | [\GOp] | [\AOp] SetValue [\Inhib]**

TrggData

Tipo de dato: *triggdata*

Es una variable utilizada para almacenar los datos *triggdata* retornados de esta instrucción. Estos datos *triggdata* serán utilizados luego en las instrucciones *TriggL*, *TriggC* o *TriggJ* siguientes.

DistanciaTipo de dato: *num*

Define la posición en la trayectoria donde debe ocurrir el evento de E/S.

Especificada como siendo la distancia en mm (valor positivo) desde el punto final de la trayectoria de movimiento (aplicable si el argumento `\Start` no ha sido activado).

Véase la sección titulada Ejecución del programa para más detalles.

[\Start]Tipo de dato: *switch*

Se utiliza cuando la distancia del argumento *Distancia* empieza en el punto de arranque del movimiento en lugar de empezar en el punto final.

EquipLag

(Retraso del equipo)

Tipo de dato: *num*

Sirve para especificar el retraso del equipo externo en segundos.

Para la compensación del retraso del equipo externo, se deberá utilizar un valor de argumento positivo. Un valor de argumento positivo significa que la señal de E/S es activada por el sistema robot en un momento específico antes de que el TCP físico alcance la distancia específica respecto al punto de arranque del movimiento o al punto final.

Un valor de argumento negativo significa que la señal de E/S es activada por el sistema robot a un momento específico después de que el TCP físico haya pasado la distancia especificada respecto al punto de arranque del movimiento o al punto final.

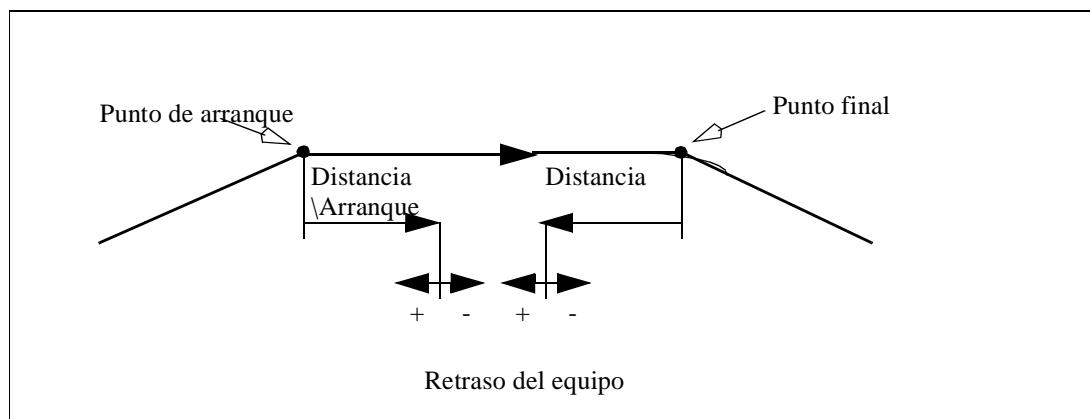


Figura 2 Utilización del argumento *EquipLag*.

[\DOp]

(Salida Digital)

Tipo de dato: *signaldo*

Es el nombre de la señal, cuando se desea cambiar una señal de salida digital.

[\GOp]

(Grupo Salida)

Tipo de dato: *signalgo*

Es el nombre de la señal, cuando se desea cambiar un grupo de señales de salida digital.

[\AOp] *(Salida analógica)* Tipo de dato: *signalao*

Es el nombre de la señal, cuando se desea cambiar una señal de salida analógica.

SetValue Tipo de dato: *num*

Es el valor deseado de la señal de salida (dentro del límite permitido para la señal correspondiente).

[\Inhib] *(Inhibición)* Tipo de dato: *bool*

Es el nombre de la bandera variable persistente para la inhibición de la activación de la señal durante el funcionamiento.

Si se utiliza este argumento opcional y que el valor actual de la bandera especificada está en TRUE en el tiempo-posición para la activación de la señal, entonces la señal especificada (*DOp*, *GOp* o *AOp*) se activará en 0 en lugar de activarse en el valor deseado.

Ejecución del programa

Cuando se ejecuta la instrucción *TriggEquip*, la condición de disparo es almacenada en la variable específica para el argumento *TriggData*.

Más adelante, cuando una de las instrucciones *TriggL*, *TriggC* o *TriggJ* es ejecutada, lo que se indica a continuación se aplicará, con respecto a las definiciones contenidas en *TriggEquip*:

La distancia especificada en el argumento *Distancia*:

Movimiento lineal La distancia en línea recta

Movimiento circular La longitud del arco de círculo

Movimiento no-lineal La longitud aproximada del arco a lo largo de la trayectoria (para obtener la precisión adecuada, la distancia no debe exceder la mitad de la longitud del arco).

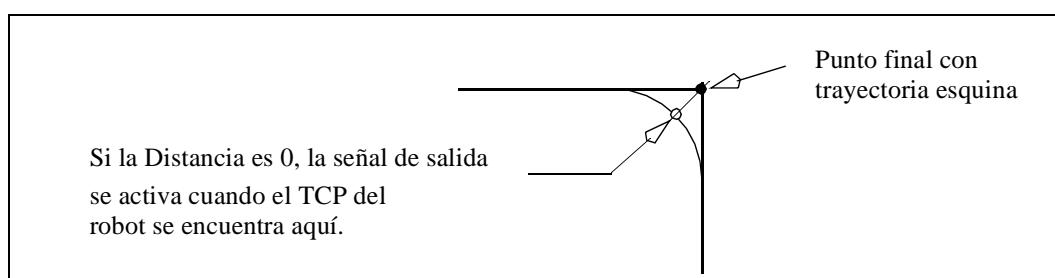


Figura 3 E/S de tiempo-posición fijas en una trayectoria esquina.

El evento relativo al tiempo-posición será generado cuando se haya pasado el punto de arranque (punto final), si la distancia especificada a partir del punto final (punto de arranque) no se encuentra dentro de la longitud de movimiento de la instrucción utili-

zada (*Trigg...*). Si se utiliza el argumento *EquipLag* con un tiempo negativo (retraso), la señal de E/S podrá ser activada después del punto final.

Ejemplos

```
VAR triggdata glueflow;  
TriggEquip glueflow, 1 \Start, 0.05 \AOp:=glue, 5.3;  
MoveJ p1, v1000, z50, tool1;  
TriggL p2, v500, glueflow, z50, tool1;
```

La señal de salida analógica *glue* se activa en el valor 5,3 cuando el TCP pasa por un punto situado a 1 mm después del punto de arranque *p1* con una compensación para el retraso del equipo de 0,05 segundos.

...
TriggL p3, v500, glueflow, z50, tool1;

La señal de salida analógica *glue* está activada de nuevo en el valor 5,3 cuando el TCP pasa por un punto situado a 1 mm después del punto de arranque *p2*.

Limitaciones

Los eventos de E/S con distancia (sin el argumento *\Time*) están concebidos para los puntos de paso (trayectoria esquina). Los eventos de E/S con distancia, que utilizan puntos de paro, tienen una precisión peor de lo que se especifica a continuación.

Respecto a la precisión de los eventos de E/S con distancia y que utilizan puntos de paso, se aplica lo siguiente cuando se activa una salida digital a una distancia específica a partir del punto de arranque o final dentro de las instrucciones *TriggL* o *TriggC*:

- La precisión que se especifica a continuación es válida para un parámetro *EquipLag* positivo < 60 ms, equivalente al retraso del servo del robot (sin cambiar el parámetro del sistema *Event Preset Time*).
- La precisión que se especifica a continuación es válida para un parámetro *EquipLag* positivo < *Event Preset Time* (parámetro del sistema) configurado.
- La precisión que se especifica a continuación no es válida para un parámetro *EquipLag* positivo > *Event Preset Time* (parámetro del sistema) configurado. En este caso, se usa un método aproximado en el que las limitaciones dinámicas del robot no son tomadas en consideración. Se deberá usar *SingArea\Wrist* a fin de obtener una precisión aceptable.
- La precisión que se especifica a continuación es válida para un parámetro *EquipLag* negativo.

Los eventos de E/S con tiempo (con el argumento *\Time*) están destinados a ser utilizados con los puntos de paro. Los eventos de E/S con tiempo, utilizados con puntos de paso, tienen una precisión peor que la que se especifica a continuación.

Los eventos de E/S con tiempo sólo podrán ser especificados a partir del punto final del movimiento. Este tiempo no deberá exceder el tiempo de frenado utilizado por el robot, que es de aproximadamente 0,5 s (valores típicos a una velocidad de 500 mm/seg para el IRB2400: 150 ms y para el IRB6400: 250 ms). En el caso en que el tiempo especificado sea mayor que el tiempo de frenado utilizado, la interrupción seguirá siendo generada, aunque no lo será antes de que haya comenzado el tiempo de frenado (más tarde de lo especificado). No obstante, el tiempo total de movimiento del movimiento utilizado podrá ser usado en movimientos rápidos y pequeños.

Valores de precisión absoluta típicos para activación de salidas digitales +/- 5 ms.

Valores de precisión de repetición típicos para activación de salidas digitales +/- 2 ms.

Las señales de salida digitales utilizadas (*DOP* o *GOP*) no podrán ser enlazadas a otras señales.

Sintaxis

```
TriggEquip
[ TriggData ':=' ] <variable (VAR) de triggdata> ',' 
[ Distance ':=' ] <expresión (IN) de num>
[ '\' Start ] ',' 
[ EquipLag ':=' ] <expresión (IN) de num>
[ '\' DOp ':=' <variable (VAR) de signaldo> ]
| [ '\' GOp ':=' <variable (VAR) de signalgo> ]
| [ '\' AOp ':=' <variable (VAR) de signalao> ] ',' 
[ SetValue ':=' ] <expresión (IN) de num>
[ '\' Inhibit ':=' <persistente (PERS) de bool>] ','
```

Información relacionada

Uso de los disparos

Descripción en:

Instrucciones - *TriggL*, *TriggC*, *TriggJ*

Definición de otros disparos

Instrucción - *TriggIO*, *TriggInt*

Más ejemplos

Tipos de datos - *triggdata*

Activación de E/S

Instrucciones - *SetDO*, *SetGO*, *SetAO*

TriggInt**Definición de una interrupción
relativa a una posición**

TriggInt sirve para definir las condiciones y acciones para la ejecución de una rutina de interrupción en una posición sobre la trayectoria de movimiento del robot.

Los datos definidos serán utilizados en una o más instrucciones *TriggL*, *TriggC* o *TriggJ* siguientes.

Ejemplos

```
VAR intnum intno1;
VAR triggdata trigg1;
...
CONNECT intno1 WITH trap1;
TriggInt trigg1, 5, intno1;
...
TriggL p1, v500, trigg1, z50, gun1;
TriggL p2, v500, trigg1, z50, gun1;
...
IDelete intno1;
```

La rutina de tratamiento de interrupción *trap1* se ejecuta cuando el TCP está en una posición situada a 5 mm antes del punto *p1* o *p2* respectivamente.

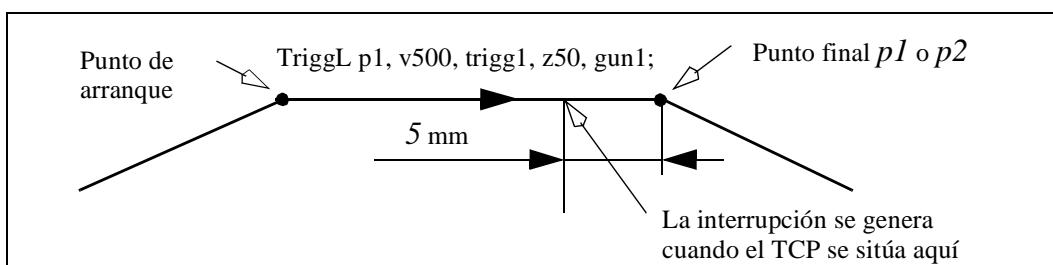


Figura 1 Ejemplo de una interrupción relativa a una posición.

Argumentos

**TriggInt TriggData Distancia [\Start] | [\Time]
Interrupción**

TriggData

Tipo de dato: *triggdata*

Es una variable utilizada para el almacenamiento de los datos *triggdata* retornados de esta instrucción. Estos datos *triggdata* serán luego utilizados en las instrucciones *TriggL*, *TriggC* o *TriggJ* siguientes.

DistanciaTipo de dato: *num*

Define la posición en la trayectoria, en el lugar donde la interrupción deberá ser generada.

Se especifica como siendo la distancia en mm (valor positivo) desde la posición final de la trayectoria de movimiento (aplicable siempre y cuando el argumento *\Start* o *\Time* no haya sido activado).

Véase la sección titulada Ejecución del Programa para más detalles.

[\Start]Tipo de dato: *switch*

Se utiliza cuando la distancia del argumento *Distancia* empieza en el punto de arranque del movimiento en lugar de empezar en el punto final.

[\Time]Tipo de dato: *switch*

Se utiliza cuando el valor especificado del argumento *Distancia* es un tiempo en segundos (valor positivo) en vez de una distancia.

Las interrupciones relativas a posiciones en el tiempo sólo podrán utilizarse para tiempos cortos (< 0,5 s) antes de que el robot alcance el punto final de la instrucción. Véase la sección titulada Limitaciones para más detalles.

InterrupciónTipo de dato: *intnum*

Es una variable utilizada para la identificación de una interrupción.

Ejecución del programa

Cuando se ejecuta la instrucción *TriggInt*, los datos son almacenados en una variable específica para el argumento *TriggData* y la interrupción que está especificada en la variable para el argumento *Interrupt* es activada.

Más adelante, cuando se ejecuta una de las instrucciones *TriggL*, *TriggC* o *TriggJ*, se aplica lo que se indica a continuación referente a las definiciones contenidas en *TriggInt*:

La distancia especificada en el argumento *Distancia*:

Movimiento lineal	La distancia en línea recta
Movimiento circular	La longitud del arco de círculo
Movimiento no-lineal	La longitud aproximada del arco sobre la trayectoria (para obtener una precisión adecuada, la distancia no deberá exceder la mitad de la longitud del arco).

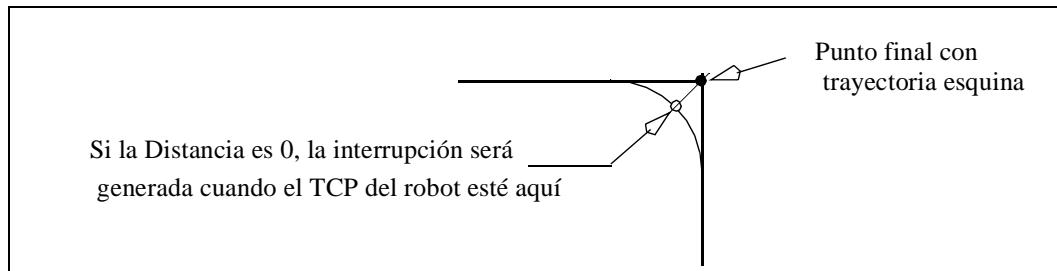


Figura 2 Interrupción relativa a una posición sobre una trayectoria esquina.

La interrupción relativa a la posición será generada cuando se haya pasado el punto de arranque (punto final), siempre y cuando la distancia especificada desde el punto final (punto de arranque) no esté dentro de la longitud de movimiento de la instrucción utilizada (*Trigg...*).

Ejemplos

Este ejemplo describe la programación de las instrucciones que intervienen para la generación de interrupciones relativas a la posición:

```
VAR intnum intno2;
VAR triggdata trigg2;
```

- Declaración de las variables *intno2* y *trigg2* (no deberán ser iniciadas).

```
CONNECT intno2 WITH trap2;
```

- Atribución de los números de interrupción que están almacenados en la variable *intno2*
- El número de interrupción es atribuido a la rutina de tratamiento de interrupción *trap2*

```
TriggInt trigg2, 0, intno2;
```

- Aparece la bandera del número de interrupción de la variable *intno2* cuando es utilizada
- La interrupción es activada
- El número de interrupción y las condiciones de disparo definidas son almacenadas en la variable *trigg2*

TriggL p1, v500, trigg2, z50, gun1;

- El robot es movido al punto *p1*.
- Cuando el TCP ha alcanzado el punto *p1*, se genera una interrupción y se ejecuta la rutina de tratamiento de interrupción *trap2*.

TriggL p2, v500, trigg2, z50, gun1;

- El robot es movido al punto *p2*
- Cuando el TCP alcanza el punto *p2*, se genera una interrupción y la rutina de tratamiento de interrupción *trap2* se ejecuta otra vez.

IDelete intno2;

- El número de interrupción contenido en la variable *intno2* deja de ser atribuido.

Limitaciones

Los eventos de interrupciones relativos a la distancia (sin el argumento \Time) están destinados a ser utilizados con puntos de paso (trayectoria esquina). Los eventos de interrupciones relativos a la distancia, utilizados con puntos de paro tienen una peor precisión que la especificada a continuación.

Los eventos de interrupciones relativos al tiempo (con el argumento \Time) están destinados a ser utilizados con los puntos de paro. Los eventos de interrupciones relativos al tiempo, utilizados con puntos de paso, tienen una peor precisión que la especificada a continuación.

Los eventos de E/S relativos al tiempo sólo podrán ser especificadas desde el punto final del movimiento. Este tiempo no deberá exceder el tiempo de frenado utilizado por el robot, que es de aproximadamente 0,5 s (valores típicos a una velocidad de 500 mm/seg para el IRB2400: 150 ms y para el IRB6400: 250 ms). En el caso en que el tiempo especificado sea mayor que el tiempo de frenado utilizado, el evento seguirá siendo generado, aunque no lo será antes de que haya empezado el tiempo de frenado (más tarde de lo especificado). No obstante, el tiempo total de movimiento del movimiento utilizado podrá ser usado en movimientos rápidos y pequeños.

Valores de precisión absoluta típicos para la generación de interrupciones +/- 5 ms.
Valores de precisión de repetición típicos la generación de interrupciones +/- 2 ms.

Normalmente suele haber un retraso que oscila entre 5 y 120 ms. entre la generación de la interrupción y la respuesta, dependiendo del tipo de movimiento que se está realizando en el momento de la interrupción. (Referirse a Características Básicas RAPID - Interrupciones).

Para obtener la mayor precisión cuando se activa una salida en una posición fija sobre la trayectoria del robot, se deberá utilizar preferentemente la instrucción *TriggIO* o *TriggEquip* en vez de las instrucciones *TriggInt* con *SetDO/SetGO/SetAO* en una rutina de interrupción.

Sintaxis

```
TriggInt
  [ TriggData ':=' ] < variable (VAR) de triggdata > ','
  [ Distance ':=' ] < expresión (IN) de num >
  [ '\' Start ] | [ '\' Time ] ',''
  [ Interrupt ':=' ] < variable (VAR) de intnum > ';
```

Información relacionada

Descripción:

- Utilización de disparos
- Definición de E/S de posición fija
- Más ejemplos
- Interrupciones

- Instrucciones - *TriggL, TriggC, TriggJ*
- Instrucción - *TriggIO, TriggEquip*
- Tipos de datos - *triggdata*
- Características básicas - *Interrupciones*

TriggIO**Definición de un evento de E/S
de posición fija**

TriggIO sirve para definir las condiciones y acciones para activar una señal digital, un grupo de señales digitales o una señal de salida analógica en una posición fija en la trayectoria de movimiento del robot.

Para obtener un evento de E/S de posición fija, *TriggIO* compensa el retraso en el sistema de control (retraso entre el robot y el servo) pero no compensará ningún retraso en el equipo externo. Para una compensación de ambos retrasos, se deberá utilizar *TriggEquip*.

Los datos definidos serán utilizados en una o más instrucciones *TriggL*, *TriggC* o *TriggJ* siguientes.

Ejemplos

```
VAR triggdata gunon;
```

```
TriggIO gunon, 10 \DOp:=gun, 1;
```

```
TriggL p1, v500, gunon, z50, gun1;
```

La señal de salida digital *gun* está activada en el valor *I* cuando el TCP está a *10 mm* antes del punto *p1*.

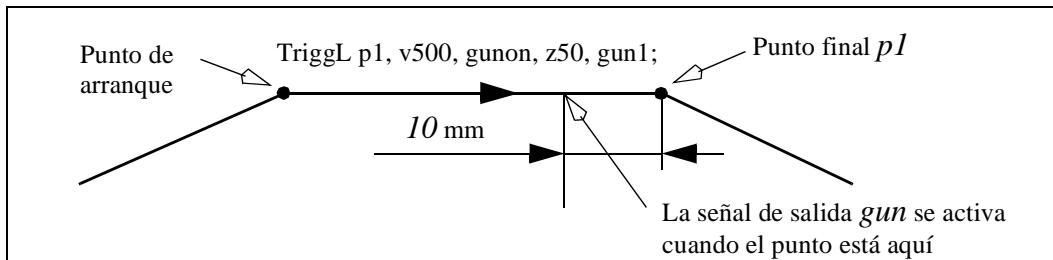


Figura 1 Ejemplo de un evento de E/S de posición fija.

Argumentos

**TriggIO TriggData Distancia [\Start] | [\Time]
[\DOp] | [\GOp] | [\AOp] | [\ProcID] SetValue
[\DODelay]**

TriggData

Tipo de dato: *triggdata*

Es una variable utilizada para almacenar los datos *triggdata* retornados de esta instrucción. Estos datos *triggdata* serán utilizados luego en las instrucciones *TriggL*, *TriggC* o *TriggJ* siguientes.

Distancia	Tipo de dato: <i>num</i>
Define la posición en la trayectoria donde debe ocurrir el evento de E/S.	
Especificada como siendo la distancia en mm (valor positivo) desde el punto final de la trayectoria de movimiento (aplicable si el argumento <i>\Start</i> o <i>\Time</i> no ha sido activado).	
Véase la sección titulada Ejecución del programa para más detalles.	
[\Start]	
Tipo de dato: <i>switch</i>	
Se utiliza cuando la distancia del argumento <i>Distancia</i> empieza en el punto de arranque del movimiento en lugar de empezar en el punto final.	
[\Time]	
Tipo de dato: <i>switch</i>	
Se utiliza cuando el valor especificado del argumento <i>Distancia</i> es un tiempo en segundos (valor positivo) en lugar de una distancia.	
Las E/S de posición fija en el tiempo sólo podrán ser utilizadas para períodos de tiempo cortos (< 0,5 s) antes de que el robot haya alcanzado el punto final de la instrucción. Véase la sección titulada Limitaciones para más detalles.	
[\DOp]	(<i>Salida Digital</i>)
Tipo de dato: <i>signaldo</i>	
Es el nombre de la señal, cuando se desea cambiar una señal de salida digital.	
[\GOp]	(<i>Grupo Salida</i>)
Tipo de dato: <i>signalgo</i>	
Es el nombre de la señal, cuando se desea cambiar un grupo de señales de salida digital.	
[\AOp]	(<i>Salida analógica</i>)
Tipo de dato: <i>signalao</i>	
Es el nombre de la señal, cuando se desea cambiar una señal de salida analógica.	
[\ProcID]	(<i>Identidad del Proceso</i>)
Tipo de dato: <i>num</i>	
No implementado para el usuario.	
(La identidad del proceso IPM para recibir el evento. El selector está especificado en el argumento <i>SetValue</i>).	
SetValue	Tipo de dato: <i>num</i>
Es el valor deseado de la señal de salida (dentro del límite permitido para la señal correspondiente).	
[\DODelay]	(<i>Retraso Salida Digital</i>)
Tipo de dato: <i>num</i>	
Es el retraso expresado en segundos (valor positivo) para una señal de salida digital o para un grupo de señales de salida digitales.	
Se utiliza únicamente para retrasar la activación de señales de salida digitales,	

después de que el robot haya alcanzado la posición especificada. No habrá retraso si se ha omitido el argumento.

El retraso no está sincronizado con el movimiento.

Ejecución del programa

Cuando se ejecuta la instrucción *TriggIO*, la condición de disparo es almacenada en una variable específica para el argumento *TriggData*.

Más adelante, cuando una de las instrucciones *TriggL*, *TriggC* o *TriggJ* es ejecutada, lo que se indica a continuación se aplicará, con respecto a las definiciones contenidas en *TriggIO*:

La distancia especificada en el argumento *Distancia*:

Movimiento lineal	La distancia en línea recta
Movimiento circular	La longitud del arco de círculo
Movimiento no-lineal	La longitud aproximada del arco a lo largo de la trayectoria (para obtener la precisión adecuada, la distancia no debe exceder la mitad de la longitud del arco).

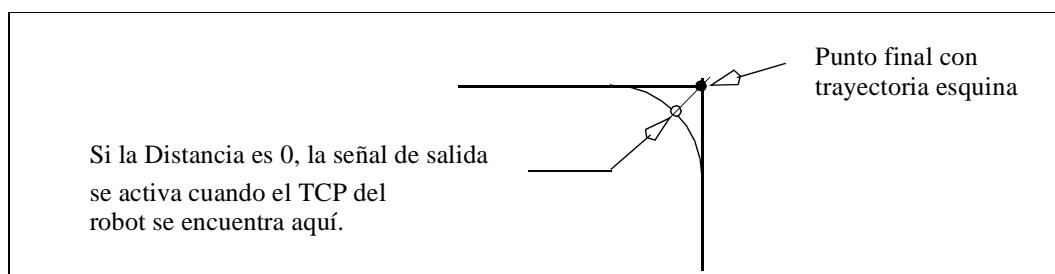


Figura 2 E/S de posición fija en una trayectoria esquina.

La E/S de posición fija será generada cuando se haya pasado el punto de arranque (punto final), si la distancia especificada a partir del punto final (punto de arranque) no se encuentra dentro de la longitud de movimiento de la instrucción utilizada (*Trigg...*).

Ejemplos

VAR triggdata glueflow;

TriggIO glueflow, 1 \Start \AOp:=glue, 5.3;

MoveJ p1, v1000, z50, tool1;
TriggL p2, v500, glueflow, z50, tool1;

La señal de salida analógica *glue* está activada en el valor 5,3 cuando el punto de trabajo pasa por un punto situado a 1 mm después del punto de arranque *p1*.

...
TriggL p3, v500, glueflow, z50, tool1;

La señal de salida analógica *glue* está activada de nuevo en el valor 5,3 cuando el punto de trabajo pasa por un punto situado a 1 mm después del punto de arranque *p2*.

Limitaciones

Los eventos de E/S con distancia (sin el argumento *\Time*) están concebidos para los puntos de paso (trayectoria esquina). Los eventos de E/S con distancia, que utilizan puntos de paro, tienen una precisión peor de lo que se especifica a continuación.

Los eventos de E/S con tiempo (con el argumento *\Time*) están destinados a ser utilizados con los puntos de paro. Los eventos de E/S con tiempo, utilizados con puntos de paso, tienen una precisión peor que la que se especifica a continuación.

Los eventos de E/S con tiempo sólo podrán ser especificados a partir del punto final del movimiento. Este tiempo no deberá exceder el tiempo de frenado utilizado por el robot, que es de aproximadamente 0,5 s (valores típicos a una velocidad de 500 mm/seg para el IRB2400: 150 ms y para el IRB6400: 250 ms). En el caso en que el tiempo especificado sea mayor que el tiempo de frenado utilizado, la interrupción seguirá siendo generada, aunque no lo será antes de que haya comenzado el tiempo de frenado (más tarde de lo especificado). No obstante, el tiempo total de movimiento del movimiento utilizado podrá ser usado en movimientos rápidos y pequeños.

Valores de precisión absoluta típicos para activación de salidas digitales +/- 5 ms.

Valores de precisión de repetición típicos para activación de salidas digitales +/- 2 ms.

Las señales de salida digitales utilizadas (*DOP* o *GOP*) no podrán ser enlazadas a otras señales.

Sintaxis

TriggIO

```
[ TriggData ':=' ] < variable (VAR) de triggdata > ','  
[ Distance ':=' ] < expresión (IN) de num >  
[ '\' Start ] | [ '\' Time ]  
[ '\' DOOp ':=' < variable (VAR) de signaldo > ]  
| [ '\' GOOp ':=' < variable (VAR) de signalgo > ]  
| [ '\' AOOp ':=' < variable (VAR) de signalao > ]  
| [ '\' ProcID ':=' < expresión (IN) de num > ] ','  
[ SetValue ':=' ] < expresión (IN) de num >  
[ '\' DODelay ':=' < expresión (IN) de num > ] ';
```

Información relacionada

Uso de los disparos

Descripción en:

Instrucciones - *TriggL, TriggC, TriggJ*

Definición de eventos de E/S relativos a
tiempo-posición

Instrucción - *TriggEquip*

Definición de interrupciones
relativas a la posición

Instrucción - *TriggInt*

Más ejemplos

Tipos de datos - *triggedata*

Activación de E/S

Instrucciones - *SetDO, SetGO, SetAO*

TriggJ**Movimientos de los ejes del robot con eventos**

TriggJ (*Trigg Joint*) sirve para activar señales de salida y/o ejecutar rutinas de interrupción en posiciones fijas, al mismo tiempo que el robot se mueve en una trayectoria circular.

Utilizando las instrucciones *TriggIO*, *TriggEquip* o *TriggInt* se podrá definir uno o más (4 como máximo) eventos y más adelante estas definiciones serán referidas en la instrucción *TriggJ*.

Ejemplos

```
VAR triggdata gunon;
TriggIO gunon, 0 \Start \DOp:=gun, on;
MoveL p1, v500, z50, gun1;
TriggJ p2, v500, gunon, fine, gun1;
```

La señal de salida digital *gun* se activa cuando el TCP del robot pasa por el punto central de la trayectoria esquina del punto *p1*.

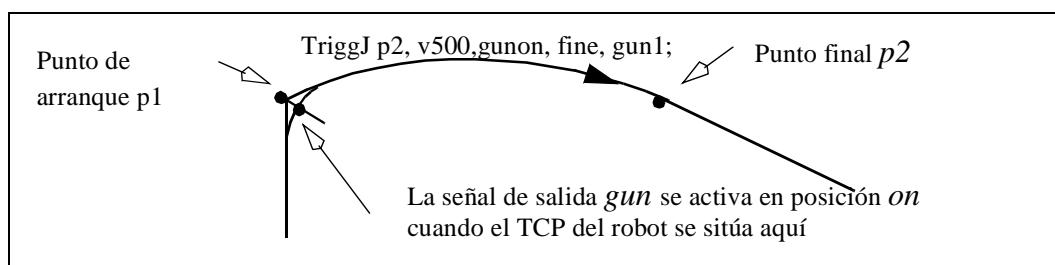


Figura 1 Ejemplo de un evento de E/S de posición fija.

Argumentos

TriggJ [\Conc] AlPunto Veloc [\T] Trigg1 [\T2] [\T3] [\T4] Zona Herram [\WObj]

[\Conc]

(Concurrente)

Tipo de dato: *switch*

Mientras el robot está en movimiento, diferentes instrucciones lógicas son ejecutadas. Este argumento sirve para acortar el tiempo de ciclo cuando, por ejemplo, se está comunicando con el equipo externo y no se requiere ninguna sincronización. También podrá utilizarse para ajustar la ejecución de la trayectoria del robot, para evitar el mensaje de aviso 50024 Fallo de trayectoria esquina o el error 40082 Límite de deceleración.

Cuando se usa el argumento \Conc, el número de instrucciones de movimiento sucesivas será limitado a 5. En una sección de programa que incluye una secuencia StorePath-RestoPath, las instrucciones de movimiento con el argumento \Conc no están permitidas.

Si se omite este argumento, la instrucción siguiente se ejecutará solamente después de que el robot haya alcanzado el punto de paro especificado o 100 ms antes de la zona especificada.

AlPuntoTipo de dato: *robtarget*

Es el punto de destino del robot y de los ejes externos. Está definido como una posición nombrada o es almacenado directamente en la instrucción (marcado con un asterisco * en la instrucción).

VelocTipo de dato: *speeddata*

Son los datos de velocidad que se aplican a los movimientos. Los datos de velocidad definen la velocidad del punto central de la herramienta, de los ejes externos y de la reorientación de la herramienta.

[\T]

(Tiempo)

Tipo de dato: *num*

Este argumento se utiliza para especificar el tiempo total en segundos durante el cual el robot se mueve. Posteriormente será sustituido por el dato de velocidad correspondiente.

Trigg1Tipo de dato: *triggdata*

Es una variable que se refiere a las condiciones y actividad de disparo, definidas previamente en el programa mediante las instrucciones *TriggIO*, *TriggEquip* o *TriggInt*.

[\T2]

(Trigg 2)

Tipo de dato: *triggdata*

Es una variable que se refiere a las condiciones y actividad de disparo, definidas previamente en el programa mediante las instrucciones *TriggIO*, *TriggEquip* o *TriggInt*.

[\T3]

(Trigg 3)

Tipo de dato: *triggdata*

Es una variable que se refiere a las condiciones y actividad de disparo, definidas previamente en el programa mediante las instrucciones *TriggIO*, *TriggEquip* o *TriggInt*.

[\T4]

(Trigg 4)

Tipo de dato: *triggdata*

Es una variable que se refiere a las condiciones y actividad de disparo, definidas previamente en el programa mediante las instrucciones *TriggIO*, *TriggEquip* o *TriggInt*.

ZonaTipo de dato: *zonedata*

Son los datos de zona utilizados para el movimiento. Los datos de zona describen

el tamaño de la trayectoria esquina generada.

Herram

Tipo de dato: *tooldata*

Es la herramienta que se utiliza cuando el robot se mueve. El punto central de la herramienta es el punto que es movido a la posición de destino especificada.

[\WObj]

(*Objeto de Trabajo*)

Tipo de dato: *wobjdata*

Es el objeto de trabajo (sistema de coordenadas) al que se refiere la posición del robot en la instrucción.

Este argumento puede ser omitido, y en el caso en que lo sea, la posición se referirá al sistema de coordenadas mundo. Por otra parte, si se utiliza un TCP estacionario o ejes externos coordinados, este argumento deberá ser especificado para el movimiento lineal a realizar relativo al objeto de trabajo.

Ejecución del programa

Véase la instrucción *MoveJ* para más información sobre los movimientos de los ejes.

Dado que las condiciones de disparo se cumplen cuando el robot se va acercando cada vez más cerca del punto final, las actividades de disparo definidas se llevarán a cabo. Las condiciones de disparo se cumplen a cierta distancia antes del punto final de la instrucción, o a cierta distancia después del punto de arranque de la instrucción, o también a cierto punto en el tiempo (limitado a un periodo de tiempo corto) antes del punto final de la instrucción.

Durante la ejecución paso a paso hacia adelante, las actividades de E/S se llevarán a cabo pero las rutinas de interrupción no serán ejecutadas. Durante la ejecución paso a paso hacia atrás, no se llevará a cabo ninguna actividad de disparo.

Ejemplos

```
VAR intnum intno1;
VAR triggdata trigg1;
...
CONNECT intno1 WITH trap1;
TriggInt trigg1, 0.1 \Time , intno1;
...
TriggJ p1, v500, trigg1, fine, gun1;
TriggJ p2, v500, trigg1, fine, gun1;
...
IDelete intno1;
```

La rutina de tratamiento de interrupción *trap1* se ejecuta cuando el punto de trabajo está en una posición que se encuentra a *0,1 s* antes del punto *p1* o *p2* respectivamente.

Limitaciones

En el caso en que el punto de arranque se haya desviado respecto a la posición habitual, de forma que la longitud de posicionamiento total de la instrucción *TriggJ* sea más pequeña que la habitual (por ejemplo, al inicio de *TriggJ* con la posición del robot en el punto final), puede ocurrir que varias o todas las condiciones de disparo se cumplan inmediatamente en una misma posición. En tales casos, la secuencia en la cual se llevarán a cabo las actividades de disparo no estarán definidas. La lógica del programa contenida en el programa del usuario no deberá estar basada en una secuencia normal de actividades de disparo, para un "movimiento incompleto".

Sintaxis

```
TriggJ
[ '\' Conc ',' ]
[ ToPoint ':=' ] < expresión (IN) de robtarget > ',' 
[ Speed ':=' ] < expresión (IN) de speeddata >
[ '\' T ':=' < expresión (IN) de num > ] ',' 
[ Trigg1 ':=' ] < variable (VAR) de triggdata >
[ '\' T2 ':=' < variable (VAR) de triggdata > ]
[ '\' T3 ':=' < variable (VAR) de triggdata > ]
[ '\' T4 ':=' < variable (VAR) de triggdata > ] ',' 
[ Zone ':=' ] < expresión (IN) de zonedata > ',' 
[ Tool ':=' ] < persistente (PERS) de tooldata >
[ '\' WObj ':=' < persistente (PERS) de wobjdata > ] ';'
```

Información relacionada

	<u>Descripción en:</u>
Movimiento lineal con disparos	Instrucciones - <i>TriggL</i>
Movimiento circular con disparos	Instrucciones - <i>TriggC</i>
Definición de los disparos	Instrucciones - <i>TriggIO</i> , <i>TriggEquip</i> o <i>TriggInt</i>
Movimiento de ejes	Principios de Movimiento y de E/S - <i>Posicionamiento durante la Ejecución del Programa</i>
Definición de la velocidad	Tipos de Datos - <i>speeddata</i>
Definición de las herramientas	Tipos de Datos- <i>tooldata</i>
Definición de los objetos de trabajo	Tipos de Datos - <i>wobjdata</i>
Movimiento en general	Principios de Movimiento y de E/S

TriggL**Movimientos lineales del robot con eventos**

TriggL (*Trigg Linear*) sirve para activar señales de salida y/o ejecutar rutinas de interrupción en posiciones fijas, al mismo tiempo que el robot está realizando un movimiento lineal.

Utilizando las instrucciones *TriggIO*, *TriggEquip* o *TriggInt* se podrá definir uno o más eventos (4 como máximo) y posteriormente estas definiciones serán referidas en la instrucción *TriggL*.

Ejemplos

```
VAR triggdata gunon;
```

```
TriggIO gunon, 0 \Start \DOp:=gun, on;
```

```
MoveJ p1, v500, z50, gun1;
TriggL p2, v500, gunon, fine, gun1;
```

La señal de salida digital *gun* está activada cuando el TCP del robot pasa por el punto central de la trayectoria esquina del punto *p1*.

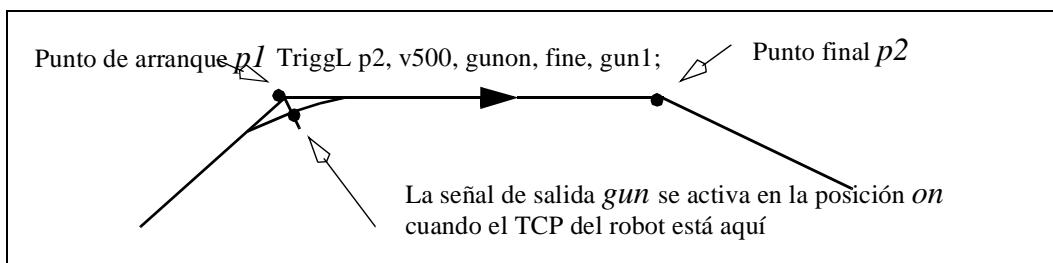


Figura 1 Ejemplo de un evento de E/S de posición fija

Argumentos

TriggL [\Conc] AlPunto Veloc [\T] Trigg1 [\T2] [\T3] [\T4]
Zona Herram [\WObj] [\Corr]

[\Conc]

(Concurrente)

Tipo de dato: *switch*

Las instrucciones lógicas siguientes son ejecutadas inmediatamente. Este argumento sirve para acortar el tiempo de ciclo cuando, por ejemplo, se está comunicando con el equipo externo y no se requiere ninguna sincronización. También podrá utilizarse para ajustar la ejecución de la trayectoria del robot, para evitar el mensaje de aviso 50024 Fallo de trayectoria esquina o el error 40082 Límite de deceleración.

Cuando se usa el argumento \Conc, el número de instrucciones de movimiento sucesivas será limitado a 5. En una sección de programa que incluye una secuencia StorePath-RestoPath, las instrucciones de movimiento con el argumento \Conc no están permitidas.

Si se omite este argumento y que AlPunto no es un punto de paro, la instrucción siguiente se ejecutará poco tiempo antes de que el robot haya alcanzado el punto de paro especificado o 100 ms antes de la zona especificada.

AlPuntoTipo de dato: *robtarget*

Es el punto de destino del robot y de los ejes externos. Está definido como una posición nombrada o es almacenado directamente en la instrucción (marcado con un asterisco * en la instrucción).

VelocTipo de dato: *speeddata*

Son los datos de velocidad que se aplican a los movimientos. Los datos de velocidad definen la velocidad del punto central de la herramienta, de los ejes externos y de la reorientación de la herramienta.

[\T]

(Tiempo)

Tipo de dato: *num*

Este argumento sirve para especificar el tiempo total en segundos durante el cual el robot se mueve. Posteriormente, será sustituido por los datos de velocidad correspondientes.

Trigg1Tipo de dato: *triggdata*

Es una variable que se refiere a las condiciones y a la actividad de disparo, definidas anteriormente en el programa mediante las instrucciones *TriggIO*, *TriggEquip* o *TriggInt*.

[\T2]

(Trigg 2)

Tipo de dato: *triggdata*

Es una variable que se refiere a las condiciones y a la actividad de disparo, definida anteriormente en el programa mediante las instrucciones *TriggIO*, *TriggEquip* o *TriggInt*.

[\T3]

(Trigg 3)

Tipo de dato: *triggdata*

Es una variable que se refiere a las condiciones y a la actividad de disparo, definida anteriormente en el programa mediante las instrucciones *TriggIO*, *TriggEquip* o *TriggInt*.

[\T4]

(Trigg 4)

Tipo de dato: *triggdata*

Es una variable que se refiere a las condiciones y a la actividad de disparo, definida anteriormente en el programa mediante las instrucciones *TriggIO*, *TriggEquip* o *TriggInt*.

ZonaTipo de dato: *zonedata*

Son los datos de zona para el movimiento. Los datos de zona describen el tamaño

de la trayectoria esquina generada.

Herram

Tipo de dato: *tooldata*

Es la herramienta que se está utilizando cuando el robot se está moviendo. El punto central de la herramienta es el punto que es movido al punto de destino especificado.

[\WObj]

(Objeto de Trabajo)

Tipo de dato: *wobjdata*

Es el objeto de trabajo (sistema de coordenadas) al que se refiere la posición del robot en la instrucción.

Este argumento puede ser omitido, y en el caso en que lo sea, la posición se referirá al sistema de coordenadas mundo. Si, por otra parte, se utiliza un TCP estacionario o ejes externos coordinados, este argumento deberá ser especificado para el movimiento lineal que se va a realizar, relativo al objeto de trabajo.

[\Corr]

(Corrección)

Tipo de dato: *switch*

Si este argumento está presente, los datos de corrección introducidos en una entrada de corrección mediante la instrucción *CorrWrite* serán añadidos a la trayectoria y a la posición de destino.

Ejecución del programa

Véase la instrucción *MoveL* para más información referente al movimiento lineal.

Dado que las condiciones de disparo se cumplen cuando el robot se va acercando cada vez más cerca del punto final, las actividades de disparo definidas se llevarán a cabo. Las condiciones de disparo se cumplen a cierta distancia antes del punto final de la instrucción, o a cierta distancia después del punto de arranque de la instrucción, o también a cierto punto en el tiempo (limitado a un periodo de tiempo corto) antes del punto final de la instrucción.

Durante la ejecución paso a paso hacia adelante, las actividades de E/S se llevarán a cabo pero las rutinas de interrupción no serán ejecutadas. Durante la ejecución paso a paso hacia atrás, no se llevará a cabo ninguna actividad de disparo.

Ejemplos

```
VAR intnum intno1;
VAR triggdata trigg1;
...
CONNECT intno1 WITH trap1;
TriggInt trigg1, 0.1 \Time, intno1;
...
TriggL p1, v500, trigg1, fine, gun1;
TriggL p2, v500, trigg1, fine, gun1;
```

...
IDelete intno1;

La rutina de tratamiento de interrupción *trap1* se ejecuta cuando el punto de trabajo está en una posición que se encuentra a 0,1 s antes del punto *p1* o *p2* respectivamente.

Limitaciones

En el caso en que el punto de arranque se haya desviado respecto a la posición habitual, de forma que la longitud de posicionamiento total de la instrucción *TriggL* sea más pequeña que la habitual (por ejemplo al principio de *TriggL* con la posición del robot en le punto final), puede ocurrir que varias o todas las condiciones de disparo se cumplan inmediatamente en una misma posición. En tales casos, la secuencia en la cual se llevarán a cabo las actividades de disparo no estarán definidas. La lógica del programa contenida en el programa del usuario no deberá estar basada en una secuencia normal de actividades de disparo, para un "movimiento incompleto".

Sintaxis

```
TriggL
[ '\' Conc ',' ]
[ ToPoint ':=' ] < expresión (IN) de robtarget > ',' 
[ Speed ':=' ] < expresión (IN) de speeddata >
[ '\' T ':=' < expresión (IN) de num > ] ',' 
[ Trigg1 ':=' ] < variable (VAR) de triggdata >
[ '\' T2 ':=' < variable (VAR) de triggdata > ]
[ '\' T3 ':=' < variable (VAR) de triggdata > ]
[ '\' T4 ':=' < variable (VAR) de triggdata > ] ',' 
[ Zone ':=' ] < expresión (IN) de zonedata > ',' 
[ Tool ':=' ] < persistente (PERS) de tooldata >
[ '\' WObj ':=' < persistente (PERS) de wobjdata > ]
[ '\' Corr ]','
```

Información relacionada

	<u>Descripción en:</u>
Movimiento circular con disparos	Instrucciones - <i>TriggC</i>
Movimiento de ejes con disparo	Instrucciones - <i>TriggJ</i>
Definición de los disparos	Instrucciones - <i>TriggIO, TriggEquip</i> <i>TriggInt</i>
Escritura en una entrada de corrección	Instrucciones - <i>CorrWrite</i>
Movimiento lineal	Principios de Movimiento y de E/S - <i>Posicionamiento durante la Ejecución del Programa</i>
Definición de la velocidad	Tipos de Datos - <i>speeddata</i>
Definición de las herramientas	Tipos de Datos - <i>tooldata</i>
Definición de los objetos de trabajo	Tipos de Datos - <i>wobjdata</i>
Movimiento en general	Principios de Movimiento y de E/S

TRYNEXT Salto de una instrucción que ha causado un error

TRYNEXT sirve para saltar una instrucción que ha causado un error. En vez de ésta, se ejecutará la siguiente instrucción.

Ejemplo

```
reg2 := reg3/reg4;  
  
ERROR  
  IF ERRNO = ERR_DIVZERO THEN  
    reg2:=0;  
    TRYNEXT;  
  ENDIF
```

Se ha realizado un intento de división de *reg3* por *reg4*. En el caso en que *reg4* sea igual a 0 (división por cero), se realizará un salto al gestor de error, en el que *reg2* está en 0. La instrucción *TRYNEXT* será entonces utilizada para continuar con la siguiente instrucción.

Ejecución del programa

La ejecución del programa continúa con la instrucción que sigue la instrucción que ha causado el error.

Limitaciones

La instrucción sólo podrá existir en un gestor de errores de una rutina.

Sintaxis

TRYNEXT';

Información relacionada

Descripción:

Gestores de error

Características Básicas -
Recuperación de Errores

TuneReset

**Reinicialización del ajuste
del servo**

TuneReset sirve para reinicializar el comportamiento dinámico de todos los ejes del robot y de las unidades mecánicas externas a sus valores normales.

Ejemplo

TuneReset;

Reinicialización de los valores de ajuste para todos los ejes al 100%.

Ejecución del programa

Los valores de ajuste de todos los ejes son reinicializados al 100%.

Los valores de ajuste por defecto del servo para todos los ejes son automáticamente activados cuando se ejecuta la instrucción *TuneReset*:

- a una puesta en marcha en frío
- cuando se carga un programa nuevo
- cuando se arranca la ejecución de un programa desde el principio

Sintaxis

TuneReset ';'

Información relacionada

Descrita en:

Ajuste de los servos

Instrucciones - *TuneServo*

TuneReset

Instrucciones

TuneServo

Ajuste de los servos

TuneServo sirve para ajustar el comportamiento dinámico de los diferentes ejes del robot. No es necesario utilizar la instrucción *TuneServo* en circunstancias normales, no obstante, en ciertas ocasiones se puede optimizar el ajuste según la configuración del robot y las características de carga. Para los ejes externos, la instrucción *TuneServo* puede usarse para la adaptación de la carga.



Tener en cuenta que una utilización incorrecta de esta instrucción *TuneServo* puede provocar movimientos de oscilación o pares que pueden dañar el robot. Siempre que se utilice *TuneServo* se deberá recordar esto.

Nota: Par obtener un ajuste óptimo, es imprescindible que se hayan usado los datos de carga correctos.

Comprobar que se hayan usado los valores correctos antes de utilizar *TuneServo*.

Descripción

Tune_df

Tune_df sirve para reducir los desbordamientos y las oscilaciones a lo largo de la trayectoria.

Siempre existe un valor de ajuste óptimo que podrá variar según la posición y la longitud del movimiento. Este valor óptimo podrá encontrarse cambiando el ajuste por pasos pequeños (1 - 2%) en los ejes implicados en este comportamiento no deseado. Habitualmente, el ajuste óptimo se alcanzará entre los límites 70% - 130%. Valores de ajuste demasiado altos o demasiado bajos comportan un efecto negativo y perjudicarán considerablemente los movimientos.

Cuando el valor de ajuste en el punto de arranque de un movimiento largo difiere considerablemente del valor de ajuste en el punto final, podrá resultar útil en ciertos casos utilizar un punto intermedio con una zona esquina para poder definir el lugar donde el valor de ajuste va a cambiar.

A continuación se muestran algunos ejemplos de utilización de la instrucción *Tune-Servo* para optimizar el ajuste:

IRB6400, en una aplicación de prensa (carga extendida y flexible), ejes 4 - 6: Reducir el valor de ajuste para el eje de la muñeca utilizado hasta que el movimiento sea aceptable. Un cambio en el movimiento será a penas perceptible hasta que se acerque del valor óptimo. Un valor bajo perjudicará considerablemente el movimiento. Valor de ajuste típico es del 25%.

IRB6400, partes superiores del área de trabajo. El eje 1 puede a menudo ser optimizado con un valor de ajuste del orden de 85% - 95%.

IRB6400, movimiento corto (< 80 mm). El eje 1 puede a menudo ser optimizado con un valor de ajuste del orden de 94% - 98%.

IRB2400, con sistema de desplazamiento lineal. En ciertos casos los ejes 2 - 3 podrán ser optimizados con un valor de ajuste del 110% - 130%. El movimiento a lo largo del sistema de desplazamiento lineal podrá requerir un valor de ajuste diferente comparado con el movimiento en los ángulos rectos del sistema de desplazamiento.

Los desbordamientos y oscilaciones podrán ser reducidas disminuyendo la aceleración o la rampa de aceleración (*AccSet*), aunque, ello aumente el tiempo del ciclo. Este es un método alternativo a la utilización de *TuneServo*.

Tune_kp, tune_kv, tune_ti, de los ejes externos

Estos tipos de ajuste afectan la ganancia de control de posición (kp) la ganancia de control de velocidad (kv) y el tiempo de integración del control de la velocidad (ti) de los ejes externos. Se utilizan para la adaptación de los ejes externos a diferentes inercias de carga. El ajuste básico de los ejes externos también puede ser simplificado utilizando estos tipos de ajuste.

Tune_kp, tune_kv, tune_ti, de los ejes del robot

Para los ejes del robot, estos tipos de ajuste tienen otro significado y podrán ser utilizados para reducir errores de trayectoria a velocidad baja (< 500 mm/s).

Los valores recomendados son: tune_kv 100 - 180%, tune_ti 50 - 100%. Tune_kp no deberá utilizarse para los ejes del robot. Valores demasiado elevados o demasiado bajos de tune_kv/tune_ti provocarán vibraciones u oscilaciones.

No utilizar nunca estos tipos de ajuste a velocidades elevadas o cuando se ha obtenido la precisión de trayectoria deseada.

Ejemplo

TuneServo IRB, 2, 90;

Activación del tipo de ajuste *TUNE_DF* con el valor de ajuste del 90% en el eje 2 de la unidad mecánica del *IRB*.

Argumentos

TuneServo UnidadMec Eje ValorAjuste [\Tipo]

UnidadMec *(Unidad Mecánica)* Tipo de dato: *mecunit*

Es el nombre de la unidad mecánica.

Eje Tipo de dato: *num*

Es el número del eje utilizado de la unidad mecánica (1 - 6).

ValorAjusteTipo de dato: *num*

Es el valor de ajuste expresado en un porcentaje (1 - 500). 100% es el valor normal.

[\\Tipo]Tipo de dato: *tunetype*

Tipo de ajuste servo. Los tipos disponibles son *TUNE_DF*, *TUNE_KP*, *TUNE_KV* y *TUNE_TI*. Estos tipos son predefinidos en el sistema con constantes.

Este argumento puede ser omitido cuando se utiliza el tipo de ajuste *TUNE_DF*.

Ejemplo

```
TuneServo MHA160R1, 1, 110 \Type:= TUNE_KP;
```

Activación del tipo de ajuste *TUNE_KP* con el valor de ajuste de 110% en el eje 1 en la unidad mecánica *MHA160R1*.

Ejecución del programa

El tipo de ajuste y el valor de ajuste especificados se activan para el eje especificado. Este valor es aplicable para todos los movimientos hasta que se haya programado un valor nuevo para el eje utilizado, o hasta que los tipos y valores de ajuste de todos los ejes hayan sido reinicializados mediante la instrucción *TuneReset*.

Los valores de ajuste servo por defecto para todos los ejes son automáticamente activados ejecutando la instrucción *TuneReset*

- a la puesta en marcha en frío
- cuando se carga un programa nuevo
- cuando se arranca la ejecución del programa desde el principio.

Sintaxis

```
TuneServo
[MecUnit ':=' ] < variable (VAR) de mecunit> ',' 
[Axis ':=' ] < expresión (IN) de num> ',' 
[TuneValue ':=' ] < expresión (IN) de num> ';' 
['\` Type ':=' <expresión (IN) de tunetype>];'
```

Información relacionada

Otras instrucciones de movimiento
Tipos de ajuste servo
Reinicialización de los ajustes servo
Ajuste de los ejes externos

Descripción:

Resumen RAPID - *Movimientos*
Tipos de Datos - *tunetype*
Instrucciones - *TuneReset*
Parámetros del Sistema - *Manipulador*

UnLoad**Descarga de un módulo de programa durante la ejecución**

UnLoad sirve para descargar un módulo de programa de la memoria de programa durante la ejecución.

El módulo del programa deberá haber sido previamente cargado en la memoria de programa mediante la instrucción *Load*.

Ejemplo

```
UnLoad ram1disk \File:="PART_A.MOD";
```

Descargar (Unload) el módulo de programa *PART_A.MOD* desde la memoria de programa, que, previamente había sido cargado en la memoria del programa con la instrucción *Load*. (Véase instrucciones *Load*). (*ram1disk* es una constante de cadena predefinida "ram1disk:").

Argumentos**Descargar Trayectoriadearchivo [\\Archivo]****Trayectoriadearchivo**Tipo de dato: *string*

Es la trayectoria del archivo y el nombre del archivo que serán descargados de la memoria de programa. La trayectoria de archivo y el nombre del archivo deben ser los mismos que en la instrucción *Load* ejecutada previamente. El nombre del archivo será excluido cuando el argumento *\\Archivo* haya sido utilizado.

[\\Archivo]Tipo de dato: *string*

Cuando el nombre del archivo esté excluido del argumento *Trayectoriadearchivo*, entonces deberá ser definido con este argumento. El nombre del archivo deberá ser el mismo que el que se ha ejecutado previamente en la instrucción *Load*.

Ejecución del programa

Para poder ejecutar una instrucción *Unload* en el programa, se deberá haber ejecutado previamente una instrucción *Load* con la misma trayectoria de archivo y el mismo nombre en el programa.

La ejecución del programa espera que el módulo de programa haya acabado de ser descargado antes de continuar con la instrucción siguiente.

Una vez que el módulo de programa ha sido descargado, el resto de los módulos de programa serán vinculados.

Para más información, véase la instrucción *Load*.

Ejemplos

UnLoad "ram1disk:DOORDIR/DOOR1.MOD";

Descargar (Unload) el módulo de programa *DOOR1.MOD* de la memoria de programa, que previamente había sido cargado dentro de la memoria de programa con la instrucción *Load*. (Véase las instrucciones *Load*).

UnLoad "ram1disk:DOORDIR" \File:="DOOR1.MOD";

Igual que el ejemplo anterior pero expresado con una sintaxis diferente.

Limitaciones

No se permitirá descargar un módulo de programa que se está ejecutando.

Las rutinas de tratamiento de interrupciones, los eventos de E/S del sistema y otras tareas del programa no podrán ser ejecutadas durante la secuencia de descarga.

Se deberán evitar los movimientos del robot durante la secuencia de descarga.

Un paro de programa generado durante la ejecución de una instrucción de *Descarga* originará un paro de protección de los motores y el mensaje de error siguiente: "20025 Stop order timeout" que se visualizará en la unidad de programación.

Gestión de errores

En el caso en que el archivo de la instrucción *Unload* no pueda ser descargado, debido a la ejecución que está en curso dentro del módulo o debido a una trayectoria incorrecta (módulo no ha sido cargado con la instrucción *Load*), entonces la variable del sistema ERRNO será activada en ERR_UNLOAD (véase “Tipos de Datos - errnum”). Este error será manipulado entonces por el gestor de errores.

Sintaxis

```
UnLoad
[FilePath':=']<expresión (IN) de string>
['\File':= '<expresión (IN) de string>'];
```

Información relacionada

Cargar un módulo de programa
Aceptar referencias no resueltas

Descripción:

Instrucciones - *Load*
Parámetros del Sistema - *Controller*
Parámetros del Sistema - *Tasks*
Parámetros del Sistema - *BindRef*

UnLoad

Instrucciones

VelSet

Cambio de la velocidad programada

VelSet sirve para aumentar o disminuir la velocidad programada de todas las instrucciones de posicionamiento siguientes. Esta instrucción puede utilizarse también para maximizar la velocidad.

Ejemplo

VelSet 50, 800;

Todas las velocidades programadas serán disminuidas del 50% del valor especificado en la instrucción. La velocidad del TCP no podrá, no obstante exceder los 800 mm/s.

Argumentos

VelSet Ajuste Max

Ajuste

Tipo de dato: *num*

Es la velocidad deseada expresada en un porcentaje de la velocidad programada.
El 100% corresponde a la velocidad programada.

Max

Tipo de dato: *num*

Es la velocidad máxima del TCP en mm/s.

Ejecución del programa

La velocidad programada de todas las instrucciones de posicionamiento siguientes, se verán afectadas hasta que se ejecute una instrucción *VelSet* nueva.

El argumento *Ajuste* afectará:

- A todos los componentes de velocidad (TCP, orientación, ejes externos lineales y rotativos) contenidos en *speeddata*.
- Al ajuste de la velocidad programada en la instrucción de posicionamiento (el argumento *\V*).
- A los movimientos temporizados.

El argumento *Ajuste* no afectará:

- A la velocidad de soldadura contenida en *welddata*.
- A la velocidad de calentamiento y de llenado contenida en *seamdata*.

El argumento *Max* sólo afectará a la velocidad del TCP.

Los valores por defecto de *Ajuste* y *Max* son del 100% y de 5000 mm/s respectivamente. Estos valores se activan automáticamente

- a la puesta en marcha en frío
- cuando se carga un programa nuevo
- cuando se ejecuta la primera instrucción en el programa.

Ejemplo

```
VelSet 50, 800;  
MoveL p1, v1000, z10, herramienta1;  
MoveL p2, v2000, z10, herramienta1;  
MoveL p3, v1000\T:=5, z10, herramienta1;
```

La velocidad es de 500 mm/s al punto *p1* y de 800 mm/s al punto *p2*. Tardará 10 segundos para ir de *p2* a *p3*.

Limitaciones

La velocidad máxima no será tomada en cuenta cuando el tiempo esté especificado en la instrucción de posicionamiento.

Sintaxis

```
VelSet  
[ Ajuste ':=' ] <expresión (IN) de num > ','  
[ Max ':=' ] <expresión (IN) de num > ';'
```

Información relacionada

Definición de la velocidad
Instrucciones de posicionamiento

Descripción:

Tipos de Datos - *speeddata*
Resumen RAPID - *Movimiento*

WaitDI Espera hasta la activación de una señal de entrada digital

WaitDI (Wait Digital Input) sirve para esperar hasta que se active una señal de entrada digital.

Ejemplos

WaitDI di4, 1;

La ejecución del programa continuará sólo después de que la entrada *di4* haya sido activada.

WaitDI grip_status, 0;

La ejecución del programa continuará sólo después de que la entrada *grip_status* haya sido reinicializada.

Argumentos

WaitDI Señal Valor [\MaxTime] [\TimeFlag]

Señal

Tipo de dato: *signaldi*

Es el nombre de la señal.

Valor

Tipo de dato: *dionum*

Es el valor deseado de la señal.

[\MaxTime]

(Tiempo Máximo)

Tipo de dato: *num*

Es el intervalo máximo de tiempo de espera permitido, expresado en segundos. En el caso en que este intervalo de tiempo haya transcurrido antes de que la condición se haya cumplido, el sistema llamará al gestor de errores, siempre y cuando disponga de uno, con el código de error *ERR_WAIT_MAXTIME*. Si el sistema no dispone de ningún gestor de error, la ejecución del robot se detendrá.

[\TimeFlag]

(Bandera Tiempo Excedido) Tipo de dato: *bool*

Es el parámetro de salida que contiene el valor TRUE cuando el tiempo máximo de espera permitido ha transcurrido antes de que la condición se haya cumplido. En el caso en que este parámetro esté incluido en la instrucción, no se considerará como un error en el caso en que el tiempo máximo de espera haya transcurrido. Este argumento será ignorado si el argumento *MaxTime* no está incluido en la instrucción.

Ejecución del programa

En el caso en que el valor de la señal es correcto cuando la instrucción es ejecutada, el programa sencillamente continuará con la instrucción siguiente.

En el caso en que el valor de la señal no es correcto, el robot entra en un estado de espera y cuando la señal adopta el valor correcto, el programa continuará. El cambio es detectado mediante una interrupción, que proporciona una respuesta rápida.

Cuando el robot está esperando, el tiempo será supervisado, y en el caso en que exceda el valor máximo de tiempo, el programa continuará siempre y cuando esté especificado *TimeFlag*, o suscitará un error en el caso en que no lo esté. Si se especifica *TimeFlag*, se activará en TRUE cuando el tiempo haya sido excedido, de lo contrario, se activará en FALSE.

Sintaxis

WaitDI

```
[ Señal ':=' ] < variable (VAR) de signaldi > ','  
[ Valor ':=' ] < expresión (IN) de dionum > ','  
['\MaxTime ':=' <expresión (IN) de num>]  
['\TimeFlag':=' <variable (VAR) de bool>] ','
```

Información relacionada

Descripción:

Espera hasta que se cumpla una condición

Instrucciones - *WaitUntil*

Espera durante un tiempo especificado

Instrucciones - *WaitTime*

WaitDO Espera hasta la activación de una señal de salida digital

WaitDO (Wait Digital Output) sirve para esperar hasta que se active una señal de salida digital.

Ejemplos

WaitDO do4, 1;

La ejecución del programa continuará sólo después de que la salida *do4* haya sido activada.

WaitDO grip_status, 0;

La ejecución del programa continuará sólo después de que la salida *grip_status* haya sido reinicializada.

Argumentos

WaitDO Señal Valor [\MaxTime] [\TimeFlag]

Señal

Tipo de dato: *signaldo*

Es el nombre de la señal.

Valor

Tipo de dato: *dionum*

Es el valor deseado de la señal.

[\MaxTime]

(Tiempo Máximo)

Tipo de dato: *num*

Es el intervalo máximo de tiempo de espera permitido, expresado en segundos. En el caso en que este intervalo de tiempo haya transcurrido antes de que la condición se haya cumplido, el sistema llamará al gestor de errores, siempre y cuando disponga de uno, con el código de error *ERR_WAIT_MAXTIME*. Si el sistema no dispone de ningún gestor de error, la ejecución del robot se detendrá.

[\TimeFlag]

(Bandera Tiempo Excedido) Tipo de dato: *bool*

Es el parámetro de salida que contiene el valor TRUE cuando el tiempo máximo de espera permitido ha transcurrido antes de que la condición se haya cumplido. En el caso en que este parámetro esté incluido en la instrucción, no se considerará como un error en el caso en que el tiempo máximo de espera haya transcurrido. Este argumento será ignorado si el argumento *MaxTime* no está incluido en la instrucción.

Ejecución del programa

En el caso en que el valor de la señal es correcto cuando la instrucción es ejecutada, el programa sencillamente continuará con la instrucción siguiente.

En el caso en que el valor de la señal no es correcto, el robot entra en un estado de espera y cuando la señal adopta el valor correcto, el programa continuará. El cambio es detectado mediante una interrupción, que proporciona una respuesta rápida.

Cuando el robot está esperando, el tiempo será supervisado, y en el caso en que exceda el valor máximo de tiempo, el programa continuará siempre y cuando esté especificado *TimeFlag*, o suscitará un error en el caso en que no lo esté. Si se especifica *TimeFlag*, se activará en TRUE cuando el tiempo haya sido excedido, de lo contrario, se activará en FALSE.

Sintaxis

```
WaitDO
[ Señal ':=' ] < variable (VAR) de signaldo > ',' ;
[ Valor ':=' ] < expresión (IN) de dionum > ',' ;
['\MaxTime ':=<expresión (IN) de num>]
['\TimeFlag':=<variable (VAR) de bool>] ',' ;
```

Información relacionada

Descripción:

Espera hasta que se cumpla una condición

Instrucciones - *WaitUntil*

Espera durante un tiempo especificado

Instrucciones - *WaitTime*

WaitTime Espera durante un tiempo especificado

WaitTime sirve para esperar durante un tiempo específico. Esta instrucción puede utilizarse también para esperar hasta que el robot y los ejes externos se hayan inmovilizado.

Ejemplo

`WaitTime 0.5;`

La ejecución del programa espera 0,5 segundos.

Argumentos

WaitTime [\\InPos] Tiempo

[\\InPos]

Tipo de dato: *switch*

En el caso en que se utilice este argumento, tanto el robot como los ejes externos deberán haberse detenido antes de que el tiempo de espera empiece a contar.

Tiempo

Tipo de dato: *num*

El tiempo, expresado en segundos, que la ejecución del programa debe esperar.

Ejecución del programa

La ejecución del programa se detiene temporalmente durante un tiempo especificado. Sin embargo, la manipulación de las interrupciones y otras funciones similares, siguen estando activas.

Ejemplo

`WaitTime \\InPos,0;`

La ejecución del programa espera hasta que el robot y los ejes externos se hayan parado.

Limitaciones

En el caso en que se use el argumento *\\Inpos*, la instrucción de movimiento que precede a esta instrucción deberá terminar con un punto de paro para que en esta instrucción sea posible realizar un rearranque después de un corte de potencia.

El argumento *\Inpos* no podrá utilizarse junto con SoftServo.

Sintaxis

```
WaitTime  
[\'\InPos\',']  
[Tiempo ':='] <expresión (IN) de num>;'
```

Información relacionada

Descripción:

Esperar hasta que se cumpla una condición

Instrucciones - *WaitUntil*

Esperar hasta que se haya activado/
reinicializado una E/S

Instrucciones - *WaitDI*

WaitUntil**Esperar hasta el cumplimiento
de una condición**

WaitUntil sirve para esperar hasta que se cumpla una condición lógica; por ejemplo, el sistema podrá esperar hasta que se haya activado una o varias entradas.

Ejemplo

WaitUntil di4 = 1;

La ejecución del programa continuará sólo después de que la entrada *di4* haya sido activada.

Argumentos

WaitUntil [\\InPos] Cond [\\MaxTime] [\\TimeFlag]

[\\InPos]

Tipo de dato: *switch*

En el caso en que se utilice este argumento, tanto el robot como los ejes externos deberán haberse detenido antes de que la condición empiece a ser evaluada.

Cond

Tipo de dato: *bool*

Es la expresión lógica que se debe esperar.

[\\MaxTime]

Tipo de dato: *num*

Es el intervalo de tiempo de espera máximo permitido y expresado en segundos. En el caso en que este tiempo haya transcurrido antes de que se haya cumplido la condición, el sistema llamará el gestor de errores, siempre y cuando haya uno, con el código de error *ERR_WAIT_MAXTIME*. En el caso en que el sistema no disponga de gestor de errores la ejecución del programa se detendrá.

[\\TimeFlag]

Tipo de dato: *bool*

Es el parámetro de salida que contiene el valor TRUE en el caso en que el tiempo de espera máximo permitido haya transcurrido antes de que se haya cumplido la condición. Si este parámetro está incluido en la instrucción, el hecho de que el tiempo máximo haya transcurrido, no será considerado como un error. Este argumento será ignorado siempre que el argumento *MaxTime* no esté incluido en la instrucción.

Ejecución del programa

Si la condición programada no se cumple con la ejecución de la instrucción *WaitUntil*,

la condición será comprobada de nuevo cada 100 ms.

Cuando el robot está esperando, el tiempo será supervisado, y en el caso en que exceda el valor máximo de tiempo, el programa continuará siempre y cuando esté especificado *TimeFlag*, o suscitará un error en el caso en que no lo esté. Si se especifica *TimeFlag*, se activará en TRUE si el tiempo ha sido excedido, de lo contrario, se activará en FALSE.

Ejemplos

```
VAR bool tempmax;
WaitUntil DInput(ent_inicio) = 1 AND DInput (estado_pinza) = 1\MaxTime := 60
    \TimeFlag := tempmax;
IF tempmax THEN
    TPWrite "No se ha recibido orden de arranque en el tiempo especificado";
ELSE
    ciclo_sig;
ENDIF
```

Si las dos condiciones de entrada no se han cumplido en 60 segundos, aparecerá un mensaje de error en el visualizador de la unidad de programación.

WaitUntil \Inpos, di4= 1;

La ejecución del programa esperará hasta que el robot se haya inmovilizado y que la entrada *di4* se haya activado.

Limitaciones

En el caso en que se use el argumento *\Inpos*, la instrucción de movimiento que precede esta instrucción deberá terminar con un punto de paro para que en esta instrucción sea posible realizar un rearranque después de un corte de potencia.

Sintaxis

```
WaitUntil
[ '\'InPos', ]
[Cond ':='] <expresión (IN) de boolIN) de num>]
[ '\'TimeFlag':=><variable (VAR) de bool>] ','
```

Información relacionada

Esperar hasta que una entrada se haya activado/reinicializado

Esperar durante un tiempo específico

Expresiones

Descripción:

Instrucciones - *WaitDI*

Instrucciones - *WaitTime*

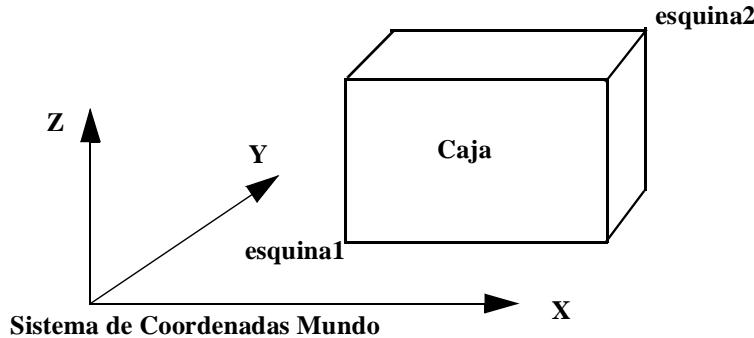
Características Básicas - *Expresiones*

WaitUntil

Instrucciones

WZBoxDef**Definición de una zona mundo en forma de caja rectangular**

WZBoxDef (World Zone Box Definition) sirve para definir una zona mundo que tiene la forma de una caja rectangular con todos sus laterales paralelos al Sistema de Coordenadas Mundo.

Ejemplo

```
VAR shapedata volume;
CONST pos corner1:=[200,100,100];
CONST pos corner2:=[600,400,400];
...
WZBoxDef \Inside, volume, corner1, corner2;
```

Define una caja rectangular con coordenadas paralelas a los ejes del sistema de coordenadas mundo, definida con las esquinas opuestas *esquina1* y *esquina2*.

Argumentos

WZBoxDef (\Dentro) | (\Fuera) Forma PuntoBajo PuntoAlto

\Dentro

Tipo de dato: *switch*

Definición del volumen dentro de la caja.

\Fuera

Tipo de dato: *switch*

Definición del volumen fuera de la caja. (volumen inverso).

Uno de los argumentos *\Dentro* o *\Fuera* deberá ser especificado.

Forma	Tipo de dato: <i>shapedata</i>
--------------	--------------------------------

Variable para el almacenamiento del volumen definido (datos particulares del sistema).

PuntoBajo	Tipo de dato: <i>pos</i>
------------------	--------------------------

Posición (x,y,z) en mm que define una esquina de la caja.

PuntoAlto	Tipo de dato: <i>pos</i>
------------------	--------------------------

Posición (x,y,z) en mm que define la esquina diagonalmente opuesta respecto a la anterior.

Ejecución del programa

La definición de la caja es almacenada en la variable del tipo *shapedata* (argumento *Forma*), para un uso ulterior en las instrucciones *WZLimSup* o *WZDOSet*.

Limitaciones

Las posiciones *PuntoBajo* y *PuntoAlto* deben ser esquinas opuestas válidas (con diferentes valores de coordenadas x, y, z).

Si se utiliza el robot para determinar el *PuntoBajo* o el *PuntoAlto*, el objeto de trabajo *wobj0* deberá estar activado (utilización del componente *trans* en *robtarget*, por ejemplo, *p1.trans* como argumento).

Sintaxis

```
WZBoxDef
('\'Dentro) | ('\'Fuera) ,
[Forma':=']<variable (VAR) de shapedata>,
[PuntoBajo':=']<expresión (IN) de pos>,
[PuntoAlto':=']<expresión (IN) de pos>;
```

Información relacionada

Forma de la zona mundo

Descripción en:

Tipos de datos - *shapedata*

Definición de la zona mundo en forma de esfera

Instrucciones - *WZSphDef*

Definición de la zona mundo en forma de cilindro

Instrucciones - *WZCylDef*

Activación del área de trabajo restringida

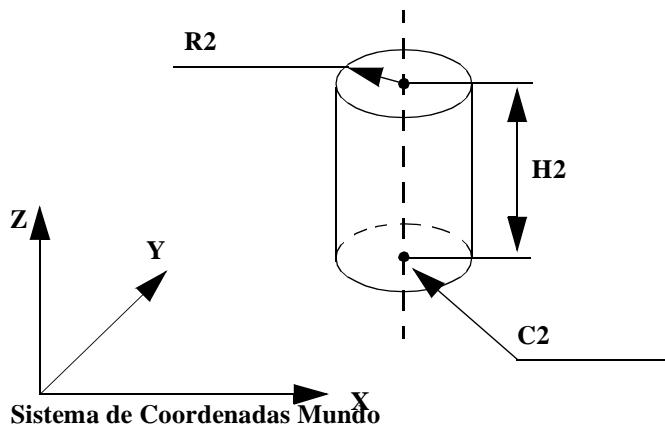
Instrucciones - *WZLimSup*

Activación de una salida referida a una zona

Instrucciones - *WZDOSet*

WZCylDef**Definición de una zona mundo
en forma de cilindro**

WZCylDef (World Zone Cylinder Definition) sirve para definir una zona mundo que tiene la forma de un cilindro con el eje del cilindro paralelo al eje-z del Sistema de Coordenadas Mundo.

Ejemplo

```
VAR shapedata volume;
CONST pos C2:=[300,200,200];
CONST num R2:=100;
CONST num H2:=200;
...
WZCylDef \Inside, volume, C2, R2, H2;
```

Define un cilindro, con el centro del círculo inferior en *C2*, con un radio *R2* y una altura *H2*.

Argumentos

WZCylDef (\Dentro) | (\Fuera) Forma PuntoCentral Radio Altura

\Dentro

Tipo de dato: *switch*

Definición del volumen dentro del cilindro.

\Fuera

Tipo de dato: *switch*

Definición del volumen fuera del cilindro (volumen inverso).

Uno de los argumentos *\Dentro* o *\Fuera* deberá ser especificado.

Forma	Tipo de dato: <i>shapedata</i>
--------------	--------------------------------

Variable para el almacenamiento del volumen definido (datos particulares del sistema).

PuntoCentral	Tipo de dato: <i>pos</i>
---------------------	--------------------------

Posición (x,y,z) en mm que define el centro de uno de los círculos del cilindro.

Radio	Tipo de dato: <i>num</i>
--------------	--------------------------

El radio del cilindro en mm.

Altura	Tipo de dato: <i>num</i>
---------------	--------------------------

La altura en mm del cilindro.

Si es positiva (dirección +z), el argumento *PuntoCentral* será el centro del círculo inferior del cilindro (como se indica en el ejemplo anterior).

Si es negativa (dirección -z) el argumento *PuntoCentral* será el centro del círculo superior del cilindro.

Ejecución del programa

La definición del cilindro es almacenada en la variable del tipo *shapedata* (argumento *Forma*), para un uso ulterior en las instrucciones *WZLimSup* o *WZDOSet*.

Limitaciones

Si se utiliza el robot para determinar el *PuntoCentral*, el objeto de trabajo *wobj0* deberá estar activado (utilización del componente *trans* en *robtarget*, por ejemplo, *p1.trans* como argumento).

Sintaxis

```
WZCylDef
('\'Dentro) | ('\'Fuera) ',''
[Forma':=']<variable (VAR) de shapedata>',''
[PuntoCentral':=']<expresión (IN) de pos>',''
[Radio':=']<expresión (IN) de num>',''
[Altura':=']<expresión (IN) de num>;'
```

Información relacionada

Definición de la zona mundo en forma de caja rectangular
Definición de la zona mundo en forma de esfera
Activación de la supervisión límite de la zona mundo
Activación de una zona mundo para activar una salida digital

Descripción:

Instrucciones - *WZSphDef*
Instrucciones - *WZCylDef*
Instrucciones - *WZLimSup*
Instrucciones - *WZDOSet*

WZDisable

Desactivación de la supervisión de la zona mundo temporal

WZDisable (*World Zone Disable*) sirve para desactivar la supervisión de una zona mundo temporal, previamente definida bien sea para parar el movimiento o bien para activar una salida.

Ejemplo

```
VAR wztemporary wzone;  
...  
PROC ...  
    WZLimSup \Temp, wzone, volume;  
    MoveL p_pick, v500, z40, tool1;  
    WZDisable wzone;  
    MoveL p_place, v200, z30, tool1;  
ENDPROC
```

En su movimiento hacia *p_pick*, la posición del TCP del robot es comprobada a fin de que no entre dentro del volumen especificado *wzone*. Esta supervisión no se lleva a cabo cuando el robot se mueve hacia la posición *p_place*.

Argumentos

WZDisable ZonaMundo

ZonaMundo

Tipo de dato: *wztemporary*

Variable o persistente del tipo *wztemporary*, que contiene la identidad de la zona mundo que debe ser desactivada.

Ejecución del programa

La zona mundo temporal es desactivada. Esto significa que la supervisión del TCP del robot relativa al volumen correspondiente, se encuentra temporalmente parada. La supervisión podrá ser reactivada mediante la instrucción *WZEnable*.

Limitaciones

Sólo se podrá desactivar una zona mundo temporal. Una zona mundo estacionaria está siempre activada.

Sintaxis

```
WZDisable  
[ZonaMundo':=']<variable o persistente (INOUT) de wztemporary>;'
```

Información relacionada

	<u>Descripción:</u>
Datos de zona mundo temporal	Tipos de datos - <i>wztemporary</i>
Activación de la supervisión límite de una zona mundo	Instrucciones - <i>WZLimSup</i>
Activación de una zona mundo para activar una salida digital	Instrucciones - <i>WZDOSet</i>
Activación de una zona mundo	Instrucciones - <i>WZEnable</i>
Borrado de una zona mundo	Instrucciones - <i>WZFree</i>

WZDOSet**Activación de la zona mundo para activar una salida digital**

WZDOSet (*World Zone Digital Output Set*) sirve para definir la acción y para activar una zona mundo para la supervisión de los movimientos del robot.

Una vez que esta instrucción ha sido ejecutada y que el TCP del robot está dentro de la zona mundo definida o se está acercando a ella, la señal de salida digital se activará en el valor específico.

Ejemplo

```
VAR wztemporary service;

PROC zone_output()
    VAR shapedata volume;
    CONST pos p_service:=[500,500,700];
    ...
    WZSphDef \Inside, volume, p_service, 50;
    WZDOSet \Temp, service \Inside, volume, do_service, 1;
ENDPROC
```

Una zona mundo temporal conocida con el nombre de *service* en la aplicación, ha sido definida. Cuando el TCP del robot está dentro de la esfera definida durante la ejecución del programa o durante el movimiento manual, la señal *do_service* será automáticamente activada.

Argumentos

WZDOSet (\Temp) | (\Stat) ZonaMundo (\Dentro) | (\Antes) Forma Señal ActValor

\Temp	<i>(Temporal)</i>	Tipo de dato: <i>switch</i>
--------------	-------------------	-----------------------------

La zona mundo a definir es una zona mundo temporal.

\Stat	<i>(Estacionaria)</i>	Tipo de dato: <i>switch</i>
--------------	-----------------------	-----------------------------

La zona mundo a definir es una zona mundo fija.

Uno de los argumentos *\Temp* o *\Stat* debe ser especificado.

ZonaMundo	Tipo de dato: <i>wztemporary</i>
------------------	----------------------------------

Variable o persistente que será actualizada con la identidad (valor numérico) de la zona mundo.

Al utilizar la posición *\Temp*, el tipo de dato debe ser *wztemporary*.

Al utilizar la posición `\Stat`, el tipo de dato debe ser `wzstationary`.

\Dentro

Tipo de dato: *switch*

La señal de salida digital se activará cuando el TCP del robot esté dentro del volumen definido.

\Antes

Tipo de dato: *switch*

La señal de salida digital se activará justo antes de que el TCP del robot alcance el volumen definido (tan pronto como sea posible antes del volumen).

Uno de los argumentos `\Dentro` o `\Antes` deberá ser especificado.

Forma

Tipo de dato: *shapedata*

La variable utilizada para definir el volumen de la zona mundo.

Señal

Tipo de dato: *signaldo*

El nombre de la señal de salida digital que será cambiada.

ActValor

Tipo de dato: *dionum*

El valor deseado de la señal (0 o 1) cuando el TCP del robot está dentro del volumen o justo antes de entrar dentro del volumen.

Cuando está fuera o justo fuera del volumen, la señal se activa en el valor opuesto.

Ejecución del programa

La zona mundo definida es activada. A partir de este momento, la posición del TCP del robot es supervisada y la salida será activada cuando la posición del TCP del robot se encuentre dentro del volumen (`\Dentro`) o se acerque al límite del volumen (`\Antes`).

Ejemplo

```
VAR wztemporary home;
VAR wztemporary service;
PERS wztemporary equip1:=[0];

PROC main()
  ...
  ! Definition of all temporary world zones
  zone_output;
  ...
  ! equip1 in robot work area
  WZEnable equip1;
  ...

```

```

! equip1 out of robot work area
WZDisable equip1;
...
! No use for equip1 any more
WZFree equip1;
...
ENDPROC

PROC zone_output()
  VAR shapedata volume;
  CONST pos p_home:=[800,0,800];
  CONST pos p_service:=[800,800,800];
  CONST pos p_equip1:=[-800,-800,0];
  ...
  WZSphDef \Inside, volume, p_home, 50;
  WZDOSet \Temp, home \Inside, volume, do_home, 1;
  WZSphDef \Inside, volume, p_service, 50;
  WZDOSet \Temp, service \Inside, volume, do_service, 1;
  WZCylDef \Inside, volume, p_equip1, 300, 1000;
  WZLimSup \Temp, equip1, volume;
  ! equip1 not in robot work area
  WZDisable equip1;
ENDPROC

```

Definición de las zonas mundo temporales *home* y *service* en la aplicación, que activan las señales *do_home* y *do_service*, cuando el robot está dentro de la esfera *home* o *service* respectivamente durante la ejecución del programa o durante el movimiento manual.

También se realiza la definición de una zona mundo temporal *equip1*, que está activa solamente en la parte del programa robot cuando *equip1* está dentro del área de trabajo del robot. En este momento el robot se detiene antes de entrar en el volumen *equip1*, tanto durante la ejecución del programa como durante la ejecución del programa y el movimiento manual. *equip1* podrá ser inhabilitada o habilitada desde otras tareas del programa mediante la utilización del valor de la variable persistente *equip1*.

Limitaciones

Una zona mundo no puede ser definida de nuevo utilizando la misma variable en el argumento *WorldZone*.

Una zona mundo estacionaria no podrá ser desactivada, activada de nuevo o borrada en el programa RAPID.

Una zona mundo temporal podrá ser desactivada (*WZDisable*), activada de nuevo (*WZEnable*) o borrada (*WZFree*) en el programa RAPID.

Sintaxis

```
WZDOSet
  ('\'Temp) | ('\'Stat) ','  
  [ZonaMundo':=']<variable o persistente (INOUT) de wztemporary>  
  ('\'Dentro) | ('\'Antes) ','  
  [Forma':=']<variable (VAR) de shapedata>','  
  [Señal':=']<variable (VAR) de signaldo>','  
  [ActValor':=']<expresión (IN) de dionum>;'
```

Información relacionada

	<u>Descripción en:</u>
Zona mundo temporal	Tipos de datos - <i>wztemporary</i>
Zona mundo estacionaria	Tipos de datos - <i>wzstationary</i>
Forma de la zona mundo	Tipos de datos - <i>shapedata</i>
Definición de la zona mundo en forma de caja rectangular	Instrucciones - <i>WZBoxDef</i>
Definición de la zona mundo en forma de esfera	Instrucciones - <i>WZSphDef</i>
Definición de la zona mundo en forma de cilindro	Instrucciones - <i>WZCylDef</i>
Activación de la supervisión límite de la zona mundo	Instrucciones - <i>WZLimSup</i>

WZEnable

Activación de la supervisión de una zona mundo temporal

WZEnable (*World Zone Enable*) sirve para volver a activar la supervisión de una zona mundo temporal, previamente definida bien sea para parar el movimiento o bien para activar una salida.

Ejemplo

```
VAR wztemporary wzone;  
...  
PROC ...  
    WZLimSup \Temp, wzone, volume;  
    MoveL p_pick, v500, z40, tool1;  
    WZDisable wzone;  
    MoveL p_place, v200, z30, tool1;  
    WZEnable wzone;  
    MoveL p_home, v200, z30, tool1;  
ENDPROC
```

En su movimiento hacia *p_pick*, la posición del TCP del robot es comprobada a fin de que no entre dentro del volumen especificado *wzone*. Esta supervisión no se lleva a cabo cuando el robot se mueve hacia la posición *p_place*, pero es reactivada antes de ir a la posición *p_home*.

Argumentos

WZEnable ZonaMundo

ZonaMundo

Tipo de dato: *wztemporary*

Variable o persistente del tipo *wztemporary*, que contiene la identidad de la zona mundo que debe ser desactivada.

Ejecución del programa

La zona mundo temporal es activada de nuevo.

Obsérvese que una zona mundo es automáticamente activada cuando es creada. Sólo deberá ser activada de nuevo cuando haya sido previamente desactivada mediante la instrucción *WZDisable*.

Limitaciones

Sólo una zona mundo temporal puede ser desactivada y activada de nuevo. Una zona mundo estacionaria está siempre activada.

Sintaxis

WZEnable

[ZonaMundo':=']<variable o persistente (**INOUT**) de *wztemporary*>';'

Información relacionada

Descripción:

Datos de una zona mundo temporal

Tipos de datos - *wztemporary*

Activación de la supervisión límite
de la zona mundo

Instrucciones - *WZLimSup*

Activación de una zona mundo para
activar una salida digital

Instrucciones - *WZDOSet*

Desactivación de una zona mundo

Instrucciones - *WZDisable*

Borrado de una zona mundo

Instrucciones - *WZFree*

WZFree

Borrado de la supervisión de la zona mundo temporal

WZFree (*World Zone Free*) sirve para borrar la definición de una zona mundo temporal, previamente definida bien sea para parar el movimiento o bien para activar una salida.

Ejemplo

```
VAR wztemporary wzone;  
...  
PROC ...  
    WZLimSup \Temp, wzone, volume;  
    MoveL p_pick, v500, z40, tool1;  
    WZDisable wzone;  
    MoveL p_place, v200, z30, tool1;  
    WZEnable wzone;  
    MoveL p_home, v200, z30, tool1;  
    WZFree wzone;  
ENDPROC
```

En su movimiento hacia la posición *p_pick*, la posición del TCP del robot es comprobada a fin de no entrar dentro de un volumen especificado como *wzone*. Esta supervisión se lleva a cabo cuando el robot se mueve a la posición *p_place*, pero es activada de nuevo antes de ir a la posición *p_home*. Cuando esta posición es alcanzada, la definición de la zona mundo es borrada.

Argumentos

WZFree ZonaMundo

ZonaMundo

Tipo de dato: *wztemporary*

Variable o persistente del tipo *wztemporary*, que contiene la identidad de la zona mundo que debe ser desactivada.

Ejecución del programa

La zona mundo temporal será en primer lugar desactivada y luego será borrada su definición.

Una vez borrada, una zona mundo temporal no podrá ser activada de nuevo ni desactivada.

Limitaciones

Sólo una zona mundo temporal puede ser desactivada, activada de nuevo o borrada.
Una zona mundo estacionaria está siempre activada.

Sintaxis

WZFree
[ZonaMundo':=']<variable o persistente (**INOUT**) de *wztemporary*>' ;'

Información relacionada

	<u>Descripción en:</u>
Datos de zona mundo temporal	Tipos de datos - <i>wztemporary</i>
Activación de la supervisión límite de una zona mundo	Instrucciones - <i>WZLimSup</i>
Activación de una zona mundo para activar una salida digital	Instrucciones - <i>WZDOSet</i>
Desactivación de una zona mundo	Instrucciones - <i>WZDisable</i>
Activación de una zona mundo	Instrucciones - <i>WZEnable</i>

WZLimSup

Activación de la supervisión del límite de la zona mundo

WZLimSup (*World Zone Limit Supervision*) sirve para definir la acción y para activar una zona mundo para la supervisión del área de trabajo del robot.

Una vez que esta instrucción ha sido ejecutada y cuando el TCP del robot ha alcanzado la zona mundo definida, el movimiento es detenido tanto durante la ejecución del programa como durante el movimiento manual.

Ejemplo

```
VAR wzstationary max_workarea;
...
PROC POWER_ON()
    VAR shapedata volume;
    ...
    WZBoxDef \Outside, volume, corner1, corner2;
    WZLimSup \Stat, max_workarea, volume;
ENDPROC
```

Definición y activación de la zona mundo estacionaria *max_workarea*, con la forma del área situada fuera de la caja rectangular (temporalmente almacenada en *volume*) y la acción de supervisión del área de trabajo activada. El robot se detiene produciendo un mensaje de error antes de entrar dentro del área fuera de la caja rectangular.

Argumentos

WZLimSup (\Temp) | (\Stat) ZonaMundo Forma

\Temp	<i>(Temporal)</i>	Tipo de dato: <i>switch</i>
--------------	-------------------	-----------------------------

La zona mundo a definir es una zona mundo temporal.

\Stat	<i>(Estacionaria)</i>	Tipo de dato: <i>switch</i>
--------------	-----------------------	-----------------------------

La zona mundo a definir es una zona mundo estacionaria.

Uno de los argumentos *\Temp* o *\Stat* debe estar especificado.

ZonaMundo	Tipo de dato: <i>wztemporary</i>
------------------	----------------------------------

Variable o persistente que será actualizada con la identidad (valor numérico) de la zona mundo.

Al utilizar el argumento *\Temp*, el tipo de dato debe ser *wztemporary*.

Al utilizar el argumento *\Stat*, el tipo de dato debe ser *wzstationary*.

FormaTipo de dato: *shapedata*

La variable que define el volumen de la zona mundo.

Ejecución del programa

La zona mundo definida es activada. A partir de este momento la posición del TCP del robot es supervisada. Si alcanza el área definida, el movimiento es detenido.

Ejemplo

```

VAR wzstationary box1_invers;
VAR wzstationary box2;

PROC wzone_power_on()
    VAR shapedata volume;
    CONST pos box1_c1:=[500,-500,0];
    CONST pos box1_c2:=[-500,500,500];
    CONST pos box2_c1:=[500,-500,0];
    CONST pos box2_c2:=[200,-200,300];
    ...
    WZBoxDef \Outside, volume, box1_c1, box1_c2;
    WZLimSup \Stat, box1_invers, volume;
    WZBoxDef \Inside, volume, box2_c1, box2_c2;
    WZLimSup \Stat, box2, volume;
ENDPROC

```

Limitación del área de trabajo del robot con las siguientes zonas mundo estacionarias:

- Fuera del área de trabajo si está fuera de box1_invers
- Fuera del área de trabajo si está dentro de box2

Si esta rutina está conectada al evento de sistema POWER ON, estas zonas mundo siempre estarán activas en el sistema tanto para los movimientos del programa como para el movimiento manual.

Limitaciones

Una zona mundo no puede ser definida de nuevo utilizando la misma variable en el argumento *WorldZone*.

Una zona mundo estacionaria no podrá ser desactivada, activada de nuevo o borrada en el programa RAPID.

Una zona mundo temporal podrá ser desactivada (*WZDisable*), activada de nuevo (*WZEnable*) o borrada (*WZFree*) en el programa RAPID.

Sintaxis

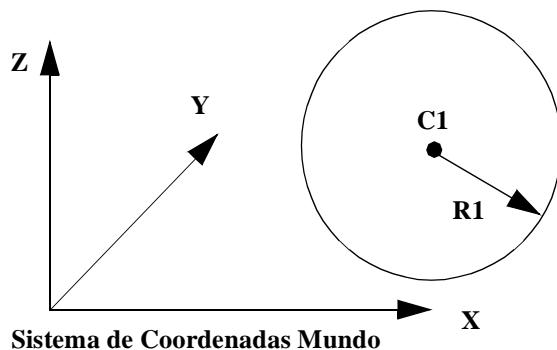
```
WZLimSup
  ('Temp) | ('\Stat)','
  [ZonaMundo':=']<variable o persistente (INOUT) de wztemporary>','
  [Forma':='] <variable (VAR) de shapedata>;'
```

Información relacionada

	<u>Descripción:</u>
Zona mundo temporal	Tipos de datos - <i>wztemporary</i>
Zona mundo estacionaria	Tipos de datos - <i>wzstationary</i>
Forma de la zona mundo	Tipos de datos - <i>shapedata</i>
Definición de la zona mundo en forma de caja rectangular	Instrucciones - <i>WZBoxDef</i>
Definición de la zona mundo en forma de esfera	Instrucciones - <i>WZSphDef</i>
Definición de la zona mundo en forma de cilindro	Instrucciones - <i>WZCylDef</i>
Activación de una zona mundo para activar una salida digital	Instrucciones - <i>WZDOSet</i>

WZSphDef**Definición de una zona mundo en forma de esfera**

WZSphDef (World Zone Sphere Definition) sirve para definir una zona mundo que tiene la forma de una esfera.

Ejemplo

```
VAR shapedata volume;
CONST pos C1:=[300,300,200];
CONST num R1:=200;
...
WZSphDef \Inside, volume, C1, R1;
```

Define una esfera que tiene el nombre de *volume* por su centro *C1* y su radio *R1*.

Argumentos

WZSphDef (\Dentro) | (\Fuera) Forma PuntoCentral Radio

\Dentro

Tipo de dato: *switch*

Definición del volumen dentro de la esfera.

\Fuera

Tipo de dato: *switch*

Definición del volumen fuera de la esfera (volumen inverso).

Uno de los argumentos *\Dentro* o *\Fuera* debe ser especificado.

Forma

Tipo de dato: *shapedata*

Variable para el almacenamiento del volumen definido (datos particulares del sistema).

PuntoCentralTipo de dato: *pos*

Posición (x,y,z) en mm que define el centro de la esfera.

RadioTipo de dato: *num*

El radio de la esfera en mm.

Ejecución del programa

La definición de la esfera es almacenada en la variable del tipo *shapedata* (argumento *Shape*), para un uso ulterior en las instrucciones *WZLimSup* o *WZDOSet*.

Limitaciones

Si el robot es utilizado para determinar el *PuntoCentral*, el objeto de trabajo *wobj0* deberá estar activado (utilización del componente *trans* en *robtarget* por ejemplo, *p1.trans* como argumento).

Sintaxis

```
WZSphDef
  ('\'Dentro) | ('\'Fuera)', '
  [Forma':=']<variable (VAR) de shapedata>', '
  [PuntoCentral':=']<expresión (IN) de pos>', '
  [Radio':=']<expresión (IN) de num>;'
```

Información relacionada

Forma de la zona mundo

Descripción:Tipos de datos - *shapedata*

Definición de la zona mundo en forma de caja rectangular

Instrucciones - *WZBoxDef*

Definición de la zona mundo en forma de cilindro

Instrucciones - *WZCylDef*

Activación de la supervisión límite de la zona mundo

Instrucciones - *WZLimSup*

Activación de una zona mundo para activar una salida digital

Instrucciones - *WZDOSet*

WHILE Repetición de una instrucción mientras...

WHILE se utiliza cuando una serie de instrucciones deben ser repetidas mientras una condición específica se vaya cumpliendo.

En el caso en que sea posible determinar por adelantado el número de repeticiones, se podrá utilizar la instrucción **FOR**.

Ejemplo

```
WHILE reg1 < reg2 DO
  ...
  reg1 := reg1 +1;
ENDWHILE
```

Repite las instrucciones del bucle WHILE mientras *reg1 < reg2*.

Argumentos

WHILE Condición DO ... ENDWHILE

Condición	Tipo de dato: <i>bool</i>
------------------	---------------------------

La condición que debe ser cumplida para que las instrucciones del bucle WHILE puedan ejecutarse.

Ejecución del programa

1. Se calcula la condición. Si la condición no ha sido cumplida, el bucle WHILE finaliza y la ejecución del programa continúa con la instrucción que sigue **ENDWHILE**.
2. Las instrucciones del bucle WHILE son ejecutadas.
3. Se repite el bucle WHILE empezando por el punto 1.

Sintaxis

(EBNF)

```
WHILE <expresión condicional> DO
  <lista instrucciones>
ENDWHILE
```

Información Relacionada

Expresiones

Descripción:

Características Básicas - *Expresiones*

Write Escritura en un archivo basado en caracteres o en un canal serie

Write sirve para escribir en un archivo basado en caracteres o en un canal serie. El valor de algunos datos podrán ser introducidos como textos.

Ejemplos

Write archivo_event, "Ejecución iniciada";

El texto *Ejecución iniciada* será introducido en el archivo con el nombre de referencia *archivo_event*.

Write archivo_event, "Nº de piezas producidas="\Num:=reg1;

El texto *Nº de piezas producidas=5*, por ejemplo, será introducido en el archivo con el nombre de referencia *archivo_event* (presuponiendo que el contenido de *reg1* es 5).

Argumentos

Write Dispositivo E/S Texto [\Num] | [\Bool] | [\Pos] | [\Orient] [\NoNewLine]

Dispositivo E/S

Tipo de dato: *iodev*

El nombre (referencia) del archivo o del canal serie utilizado.

Texto

Tipo de dato: *string*

El texto que se desea introducir.

[\Num]

(*Numérico*)

Tipo de dato: *num*

El dato cuyos valores numéricos deberán ser introducidos después de la cadena de texto.

[\Bool]

(*Boolean*)

Tipo de dato: *bool*

El dato cuyos valores lógicos deberán ser introducidos después de la cadena de texto.

[\Pos]

(*Posición*)

Tipo de dato: *pos*

El dato cuya posición deberá ser introducida después de la cadena de texto.

[|Orient] *(Orientación)* Tipo de dato: *orient*

El dato cuya orientación deberá ser introducida después de la cadena de texto.

[|NoNewLine] Tipo de dato: *switch*

Omite el símbolo de retorno de carro que normalmente indica el final del texto.

Ejecución del programa

La cadena de texto será introducida en un archivo o en un canal serie específico. En el caso en que no se use el argumento *|NoNewLine*, se introducirá un símbolo de retorno de carro (line-feed: LF).

En el caso en que se utilice uno de los argumentos *|Num*, *|Bool*, *|Pos* o *|Orient*, su valor será primeramente convertido en una cadena de texto antes de ser añadido a la primera cadena. La conversión del valor en cadena de texto ocurre de la siguiente forma:

<u>Argumentos</u>	<u>Valor</u>	<u>Cadena de texto</u>
\Num	23	"23"
\Num	1.141367	"1.141367"
\Bool	VERDADERO	"VERDADERO"
\Pos	[1817.3,905.17,879.11]	"[1817.3,905.17,879.11]"
\Orient	[0.96593,0,0.25882,0]	"[0.96593,0,0.25882,0]"

El valor es convertido en una cadena con el formato estándar en RAPID. Esto significa en principio 6 cifras significantes. Si la parte decimal es menor que 0,000005 o mayor que 0,999995, el número será redondeado a un número entero.

Ejemplo

```
VAR iodev impresora;
.
Open "sio1:", impresora\Write;
WHILE DInput(stopprod)=0 DO
    produce_part;
    Write impresora, "Piezas producidas="\Num:=reg1\NoNewLine;
    Write impresora, "      "\NoNewLine;
    Write impresora, CTime();
ENDWHILE
Close impresora;
```

Una línea que incluye el número de piezas producidas así como la hora saldrá en la impresora al final de cada ciclo. La impresora deberá estar conectada al canal serie *sio1:*. El mensaje impreso tendrá un aspecto parecido al siguiente:

Piezas producidas=473 09:47:15

Limitaciones

Los argumentos *\Num*, *\Bool*, *\Pos* y *\Orient* son mutuamente exclusivos y por lo tanto no podrán ser utilizados simultáneamente en la misma instrucción.

Esta instrucción sólo podrá utilizarse para archivos o canales serie que hayan sido abiertos para la escritura.

Gestión de errores

Si ocurre un error durante la escritura, la variable del sistema ERRNO pasará a ERR_FILEACC. Este error podrá ser procesado posteriormente por el gestor de errores.

Sintaxis

Write

```
[Dispositivo de E/S':='] <variable (VAR) de iodev' ,'  
[Texto':='] <expresión (IN) de string>  
[\'Num':='] <expresión (IN) de num> ]  
| [\'Bool':='] <expresión (IN) de bool> ]  
| [\'Pos':='] <expresión (IN) de pos> ]  
| [\'Orient':='] <expresión (IN) de orient> ]  
[\'NoNewLine'];'
```

Información relacionada

Descripción:

Apertura de un archivo o canal serie

Resumen RAPID - Comunicación

Write

Instrucciones

WriteBin Escritura en un canal serie binario

`WriteBin` sirve para escribir un número de bytes en un canal serie binario.

Ejemplo

```
WriteBin canal2, buffer_text, 10;
```

10 caracteres de la lista *buffer_text* serán introducidos en el canal referido como *canal2*.

Argumentos

WriteBin Dispositivo E/S Buffer NCar

El nombre (referencia) del canal serie utilizado.

La lista (matriz) que contiene los números (caracteres) que se desean introducir.

NCar (*Número de Caracteres*) Tipo de dato: num

El número de caracteres que se desean introducir a partir del *Buffer*.

Ejecución del programa

La cantidad especificada de números (caracteres) de la lista será escrita en el canal serie.

Limitaciones

Esta instrucción sólo podrá utilizarse para los canales serie que hayan sido abiertos para la lectura y escritura binaria.

Gestión de errores

En el caso en que ocurra un error durante la escritura, la variable del sistema ERRNO pasará a ERR_FILEACC. Este error podrá ser procesado posteriormente en el gestor de errores.

Ejemplo

```

VAR iodev canal;
VAR num buffer_sal {20};
VAR num entrada;
VAR num ncar;
Open "sio1:", canal\Bin;

buffer_sal {1} := 5;                                ( enq )
WriteBin canal, buffer_sal, 1;
entrada:= ReadBin (canal \Time:= 0.1);

IF entrada = 6 THEN                                ( ack )
    buffer_sal{1} := 2;                            ( stx )
    buffer_sal{2} := 72;                           ( 'H' )
    buffer_sal{3} := 101;                          ( 'e' )
    buffer_sal{4} := 108;                          ( 'l' )
    buffer_sal{5} := 108;                          ( 'l' )
    buffer_sal{6} := 111;                           ( 'o' )
    buffer_sal{7} := 32;                            ( ' ' )
    buffer_sal{8} := StrToByte("w"\Char);          ( 'w' )
    buffer_sal{9} := StrToByte("o"\Char);          ( 'o' )
    buffer_sal{10} := StrToByte("r"\Char);          ( 'r' )
    buffer_sal{11} := StrToByte("l"\Char);          ( 'l' )
    buffer_sal{12} := StrToByte("d"\Char);          ( 'd' )
    buffer_sal{13} := 3;                            ( etx )
    WriteBin canal, buffer_sal, 13;
ENDIF

```

La cadena de texto *Hello world* (con sus caracteres de control asociados) será escrita en un canal serie. La función *StrToByte* se utiliza en los mismos casos para convertir una cadena en un dato *byte* (*num*).

Sintaxis

```

WriteBin
[Dispositivo E/S:=] <variable (VAR) de iodev>;'
[Buffer:=] <matriz {*} (IN) de num>;'
[NCar:=] <expresión (IN) de num>;'

```

Información relacionada

Descripción:

Apertura (etc.) de canales serie

Resumen RAPID - *Comunicación*

Conversión de una cadena en un dato byte

Funciones - *StrToByte*

Datos byte

Tipos de datos - *byte*

WriteStrBin

Escritura de una cadena en un canal serie binario

WriteStrBin (*Write String Binary*) sirve para escribir una cadena en un canal serie binario o en un archivo binario.

Ejemplo

```
WriteStrBin channel2, "Hello World\0A";
```

La cadena "*Hello World\0A*" está escrita en el canal que se conoce con el nombre de *channel2*. En este caso, la cadena se termina con una línea nueva \0A. Todos los caracteres y valores hexadecimales escritos con *WriteStrBin* permanecerán incambiados por el sistema.

Argumentos

WriteStrBin	DispositivoE/S	Str
--------------------	-----------------------	------------

IODevice		Tipo de dato: <i>iodev</i>
-----------------	--	----------------------------

El nombre (referencia) del canal serie actual.

Str	(<i>Cadena</i>)	Tipo de dato: <i>string</i>
------------	-------------------	-----------------------------

El texto que se va a escribir.

Ejecución del programa

La cadena de texto ha sido escrita en el canal serie o en el archivo especificado.

Limitaciones

Esta instrucción sólo podrá utilizarse para canales serie o archivos que han sido abiertos para la lectura o escritura binaria.

Gestión de errores

Si ocurre un error durante la escritura, la variable del sistema ERRNO se activa en ERR_FILEACC. Este error podrá entonces ser manipulado en el gestor de errores.

Ejemplo

```

VAR iodev channel;
VAR num input;
Open "sio1:", channel\Bin;

! Send the control character enq
WriteStrBin channel, "\05";
! Wait for the control character ack
input := ReadBin (channel \Time:= 0.1);
IF input = 6 THEN
    ! Send a text starting with control character stx and ending with etx
    WriteStrBin channel, "\02Hello world\03";
ENDIF

Close channel;

```

La cadena de texto *Hello world* (con los caracteres de control asociados en hexadecimales) es escrita en un canal serie binario.

Sintaxis

```

WriteStrBin
[DispositivoE/S':='] <variable (VAR) of iodev>,
[Str':='] <expression (IN) of string>;

```

Información relacionada

Apertura (etc.) de canales serie

Descripción:

Resumen RAPID - *Comunicación*

INDICE

Abs	Obtención del valor absoluto
ACos	Cálculo del valor del arco coseno
AOutput	Lectura del valor de una señal de salida analógica
ArgName	Coger el nombre de un argumento
ASin	Cálculo del valor del arco seno
ATan	Cálculo del valor del arco tangente
ATan2	Cálculo del valor del arco tangente2
ByteToStr	Conversión de un byte en un dato de cadena
CDate	Lectura de la fecha actual como una cadena
CJointT	Lectura de los ángulos actuales de los ejes
ClkRead	Lectura de un reloj utilizado para el cronometraje
CorrRead	Lectura de todos los offsets utilizados
Cos	Cálculo del valor del coseno
CPos	Lectura de los datos de posición actuales (pos)
CRobT	Lectura de los datos de la posición actuales (robtarget)
CTime	Lectura de la hora actual como una cadena
CTool	Lectura de los datos de herramienta actuales
CWObj	Lectura de los datos del objeto de trabajo actuales
DefDFrame	Definición de una base de desplazamiento
DefFrame	Definición de una base de coordenadas
Dim	Obtención del tamaño de una matriz
DOutput	Lectura del valor de una señal de salida digital
EulerZYX	Obtención de ángulos Euler a partir de una variable de orientación
Exp	Cálculo del valor exponencial
GOutput	Lectura del valor de un grupo de señales de salida digital
GetTime	Lectura de la hora actual como un valor numérico
IndInpos	Estado de la posición independiente
IndSpeed	Estado de la velocidad independiente
IsPers	Es Persistente
IsVar	Es Variable
MirPos	Creación de la imagen espejo de una posición
NumToStr	Conversión de un valor numérico en cadena
Offs	Desplazamiento de una posición del robot
OpMode	Lectura del modo de operación
OrientZYX	Cálculo de una variable de orientación a partir de ángulos Euler
ORobT	Eliminación de un desplazamiento de programa de una posición

Funciones

PoseInv	Inversión de la posición
PoseMult	Multiplicación de los datos de posición
PoseVect	Aplicación de una transformación a un vector
Pow	Cálculo de la potencia de un valor
Present	Comprobación de la utilización de un parámetro opcional
ReadBin	Lectura a partir de un canal serie binario o archivo
ReadMotor	Lectura de los ángulos del motor actuales
ReadNum	Lectura de un número a partir de un archivo o de un canal serie
ReadStr	Lectura de una cadena a partir de un archivo o de un canal serie
RelTool	Ejecución de un desplazamiento relativo a la herramienta
Round	Round es un valor numérico
RunMode	Lectura del modo de funcionamiento
Sin	Cálculo del valor del seno
Sqrt	Cálculo del valor de la raíz cuadrada
StrFind	Búsqueda de un carácter en una cadena
StrLen	Obtención de la longitud de la cadena
StrMap	Mapa de una cadena
StrMatch	Búsqueda de una estructuraen una cadena
StrMemb	Comprobar si un carácter pertenece a un conjunto
StrOrder	Comprobar si las cadenas están ordenadas
StrPart	Obtención de una parte de una cadena
StrToByte	Conversión de una cadena en un dato byte
StrToVal	Conversión de una cadena en un valor numérico
Tan	Cálculo del valor de la tangente
TestDI	Comprobación de la activación e una entrada digital
Trunc	Truncar un valor numérico
ValToStr	Conversión de un valor en una cadena

Abs**Obtención del valor absoluto**

Abs sirve para obtener el valor absoluto, es decir, el valor positivo de un dato numérico.

Ejemplo

```
reg1 := Abs(reg2);
```

Reg1 tiene asignado el valor absoluto de *reg2*.

Valor de Retorno

Tipo de dato: *num*

El valor absoluto, es decir, un valor numérico positivo.

ejemplo:	<u>Valor de entrada</u>	<u>Valor de retorno</u>
	3	3
	-3	3
	-2.53	2.53

Argumentos**Abs (Entrada)****Entrada**

Tipo de dato: *num*

El valor de entrada.

Ejemplo

```
TPReadNum num_piezas, "¿Cuantas piezas se deben producir? ";
num_piezas := Abs(num_piezas);
```

El sistema pide al usuario que introduzca el número de piezas que deben ser procesadas. Para garantizar que el valor sea mayor que cero, el valor proporcionado por el usuario será convertido en un valor positivo.

Sintaxis

```
Abs '('  
[ Entrada ':=' ] <expression (IN) of num > ')'
```

Una función con un valor de retorno del tipo de dato *num*.

Información relacionada

Describo en:

Instrucciones y funciones matemáticas

Resumen RAPID - *Matemáticas*

ACos**Cálculo del valor del arco coseno**

ACos (Arc Cosine) sirve para calcular el valor del arco coseno.

Ejemplo

```
VAR num angulo;  
VAR num valor;
```

```
.
```

```
.
```

```
angulo:= ACos(valor);
```

Valor de retorno

Tipo de dato: *num*

El valor del arco coseno, expresado en grados y comprendido entre [0, 180].

Argumentos**ACos (Valor)****Valor**

Tipo de dato: *num*

El valor del argumento, comprendido entre [-1, 1].

Sintaxis

```
Acos'(''  
[Valor ':='] <expresión (IN) de num>  
'')
```

Una función con un valor de retorno del tipo de dato *num*.

Información relacionada

Descripción:

Instrucciones y funciones matemáticas

Resumen RAPID - *Matemáticas*

AOutput**Lectura del valor de una señal de salida analógica**

AOutput sirve para leer el valor actual de una señal de salida analógica.

Ejemplo

IF AOutput(ao4) > 5 THEN ...

Si el valor actual de la señal *ao4* es mayor que 5, entonces ...

Valor de retorno

Tipo de dato: *num*

El valor actual de la señal.

El valor actual es escalado (de acuerdo con los parámetros del sistema) antes de ser leído por el programa RAPID. Véase la Figura 1.

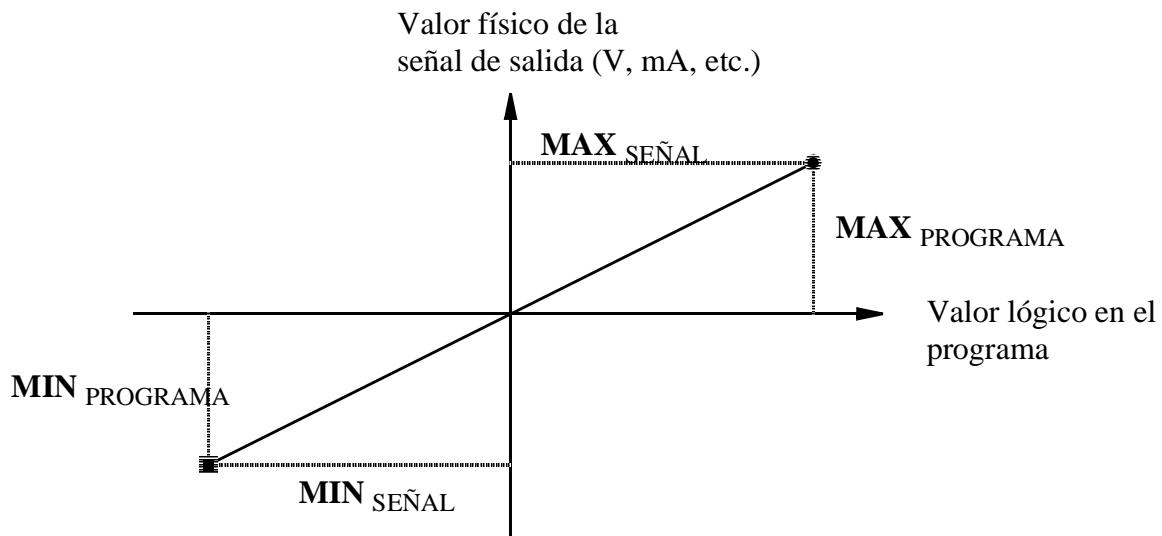


Figura 1 Diagrama indicando el escalado de los valores de señales analógicas.

Argumentos

AOutput (Señal)

SeñalTipo de dato: *signalao*

El nombre de la salida analógica que debe ser leída.

Sintaxis

AOutput ' ([Señal ':='] < variable (**VAR**) de *signalao* >)'

Una función con un valor de retorno del tipo de dato *num*.

Información relacionada

Descripción:

Instrucciones de Entrada/Salida

Resumen RAPID -
Señales de Entrada y Salida

Funcionalidad Entrada/Salida en general

Principios de Movimiento y de E/S -
Principios de E/S

Configuración de E/S

Guía del Usuario - *Parámetros del Sistema*

ArgName**Coger el nombre de un argumento**

ArgName (Argument Name) sirve para coger el nombre del dato original del argumento actual o de los datos actuales.

Ejemplo

```

VAR num abc123 :=5;
...
proc1 abc123;

PROC proc1 (num par1)
    VAR string parstring;
    ...
    parstring:=ArgName(par1);
    TPWrite "Argument name "+parstring+" with value "\Num:=par1;
ENDPROC

```

Se ha asignado a la variable *parstring* el valor de cadena "*abc123*" y aparecerá visualizado la cadena siguiente en la unidad de programación: "Nombre de argumento abc123 con valor 5" ("Argument name chales with value 5").

Valor de retornoTipo de dato: *string*

El nombre del dato original.

Argumentos**ArgName (Parámetro)****Parámetro**Tipo de dato: *anytype*

El identificador del parámetro (para la rutina en la que se encuentra *ArgName*) o la identidad del dato.

Ejecución del programa

La función retorna el nombre original del dato de un objeto entero del tipo constante, variable o persistente. El dato original puede ser global, local en el módulo del programa o local en la rutina (normas de alcance RAPID normales).

Si es una parte de un dato, el nombre del dato objeto completo será retornado.

Ejemplo

Conversión a partir de un identificador a una cadena

Esta función puede usarse también para realizar la conversión a partir de un identificador a una cadena, determinando el identificador en el argumento *Parámetro* para cualquier dato objeto de alcance global, local en el módulo o local en la rutina:

```
VAR num chales :=5;  
...  
proc1;  
  
PROC proc1 ()  
    VAR string name;  
    ...  
    name:=ArgName(chales);  
    TPWrite "Global data object "+name+" has value "\Num:=chales;  
ENDPROC
```

El valor de cadena "*chales*" es asignado a la variable *name* y aparece escrita en la unidad de programación la siguiente cadena: "El dato objeto global chales tiene el valor 5" ("Argument name chales with value 5").

Llamada de una rutina en varios pasos

Observar que la función retorna el nombre original del dato objeto:

```
VAR num chales :=5;  
...  
proc1 chales;  
...  
PROC proc1 (num parameter1)  
    ...  
    proc2 parameter1;  
    ...  
ENDPROC  
  
PROC proc2 (num par1)  
    VAR string name;  
    ...  
    name:=ArgName(par1);  
    TPWrite "Original data object name "+name+" with value "\Num:=par1;  
ENDPROC
```

El valor de cadena "*chales*" es asignado a la variable *name* y aparece escrita en la unidad de programación la siguiente cadena: "El dato objeto original chales tiene el valor 5" ("Argument name chales with value 5").

Gestión de errores

Si ocurre uno de los siguientes errores, la variable del sistema ERRNO se activa en ERR_ARGNAME:

- El argumento es un valor de expresión
- El argumento no está presente
- El argumento es del tipo switch

Este error puede ser manipulado en el gestor de errores.

Sintaxis

ArgName '('
[Parámetro':='] <referencia (**REF**) de cualquier tipo >)'

Una función con un valor de retorno del tipo de dato *string*.

Información relacionada

Descrita en:

Funciones de cadena

Resumen RAPID - *Funciones de Cadena*

Definición de una cadena

Tipos de datos - *string*

Valores de cadena

Características Básicas -
Elementos Básicos

ASin

Cálculo del valor del arco seno

ASin (*Arc Sine*) sirve para calcular el valor del arco seno.

Ejemplo

```
VAR num angulo;  
VAR num valor;  
. .  
angulo := ASin(valor);
```

Valor de retorno

Tipo de dato: *num*

El valor del arco seno, expresado en grados, y comprendido entre [-90, 90].

Argumentos

ASin (Valor)

Valor

Tipo de dato: *num*

El valor del argumento, comprendido entre [-1, 1].

Sintaxis

ASin'(
[Valor ':=] <expresión (IN) de *num*>
)'

Una función con un valor de retorno del tipo de dato *num*.

Información relacionada

Descripción:

Instrucciones y funciones matemáticas

Resumen RAPID - *Matemáticas*

ATan

Cálculo del valor del arco tangente

ATan (Arc Tangent) sirve para calcular el valor del arco tangente.

Ejemplo

```
VAR num angulo;  
VAR num valor;  
. .  
angulo := ATan(valor);
```

Valor de retorno

Tipo de dato: *num*

El valor del arco tangente, expresado en grados y comprendido entre [-90, 90].

Argumentos

ATan (Valor)

Valor

Tipo de dato: *num*

El valor del argumento.

Sintaxis

ATan'(
[Valor ':=] <expresión (**IN**) de *num*>
)'

Una función con un valor de retorno del tipo de dato *num*.

Información relacionada

Descripción:

Instrucciones y funciones matemáticas

Resumen RAPID - *Matemáticas*

Arco tangente con un valor de retorno
comprendido entre [-180, 180]

Funciones - *ATan2*

ATan2**Cálculo del valor del arco tangente2**

ATan2 (Arc Tangent2) sirve para calcular el valor del arco tangente2.

Ejemplo

```
VAR num angulo;
VAR num x_valor;
VAR num y_valor;

angulo:= ATan2(y_valor, x_valor);
```

Valor de retorno

Tipo de dato: *num*

El valor del arco de tangente, expresado en grados y comprendido entre [-180, 180].

El valor será igual a $\text{ATan}(y/x)$, y estará comprendido entre [-180, 180], puesto que la función utiliza el signo de ambos argumentos para determinar el cuadrante del valor de retorno.

Argumentos**ATan2 (Y X)**

Y

Tipo de dato: *num*

El valor del argumento del numerador.

X

Tipo de dato: *num*

El valor del argumento del denominador.

Sintaxis

```
ATan2'(
    [Y ':='] <expresión (IN) de num> ',' 
    [X ':='] <expresión (IN) de num>
    ')'
```

Una función con un valor de retorno del tipo de dato *num*.

Información relacionada

Instrucciones y funciones matemáticas
Arco de tangente con sólo un argumento

Descripción en:

Resumen RAPID - *Matemáticas*
Funciones - *ATan*

ByteToStr**Conversión de un byte
en un dato de cadena**

ByteToStr (Byte To String) sirve para convertir un *byte* en un dato de *cadena* con un formato de dato de byte definido.

Ejemplo

```
VAR string con_data_buffer{5};
VAR byte data1 := 122;
```

```
con_data_buffer{1} := ByteToStr(data1);
```

El contenido de un componente de matriz *con_data_buffer{1}* será "122" después de la función *ByteToStr* ...

```
con_data_buffer{2} := ByteToStr(data1\Hex);
```

El contenido de un componente de matriz *con_data_buffer{2}* será "7A" después de la función *ByteToStr* ...

```
con_data_buffer{3} := ByteToStr(data1\Okt);
```

El contenido de un componente de matriz *con_data_buffer{3}* será "172" después de la función *ByteToStr* ...

```
con_data_buffer{4} := ByteToStr(data1\Bin);
```

El contenido de un componente de matriz *con_data_buffer{4}* será "01111010" después de la función *ByteToStr* ...

```
con_data_buffer{5} := ByteToStr(data1\Char);
```

El contenido de un componente de matriz *con_data_buffer{5}* será "z" después de la función *ByteToStr* ...

Valor de retorno

Tipo de dato: *string*

El resultado de la operación de conversión con el siguiente formato:

Formato:	Caracteres:	Long.cadena:	Alcance:
Dec	'0' - '9'	1-3	"0" - "255"
Hex	'0' - '9', 'A' - 'F'	2	"00" - "FF"
Okt	'0' - '7'	3	"000" - "377"
Bin	'0' - '1'	8	"00000000" - "11111111"
Char:	Car. ASCII de escritura1		Tabla ASCII (*)

(*) Si se trata de un carácter ASCII que no se puede escribir, el formato de retorno será un código de carácter RAPID (por ejemplo “\07” para carácter de control BEL).

Argumentos

ByteToStr (ByteData [**\Hex**] | [**\Okt**] | [**\Bin**] | [**\Char**])

El dato byte que se desea convertir.

Si se omite el argumento opcional switch, el dato será convertido en un formato decimal (Dec).

[Hex] *(Hexadecimal)* Tipo de dato: *switch*

El dato será convertido en un formato *hexadecimal*.

[\\Okt] *(Octal)* Tipo de dato: *switch*

El dato será convertido en un formato *octal*.

[Bin] *(Binario)* Tipo de dato: *switch*

El dato será convertido en un formato *binario*.

[Char] *(Character)* Tipo de dato: *switch*

El dato será convertido en un formato de *caracter ASCII*.

Limitaciones

Un tipo de dato *byte* podrá variar entre 0 y 255 decimal.

Sintaxis

```
ByteToStr'(
    [ByteData ':='] <expresión (IN) de byte>
    ['\' Hex ] | ['\' Okt] | ['\' Bin] | ['\' Char]
    ')';'
```

Una función con un valor de retorno del tipo de dato *string*.

Información relacionada

Descripción:

Conversión de una cadena a un dato byte
Otras funciones bit (byte)
Otras funciones de cadena

Instrucciones - *StrToByte*
Resumen RAPID - *Funciones*
Resumen RAPID - *Funciones cadena*

CDate**Lectura de la fecha actual como una cadena**

CDate (Current Date) sirve para leer la fecha actual del sistema.

Esta función sirve para hacer aparecer la fecha actual en el visualizador de la unidad de programación o para introducir la fecha actual en un archivo de texto.

Ejemplo

```
VAR string fecha;  
fecha := CDate();
```

La fecha actual está almacenada en la variable *fecha*.

Valor de Retorno

Tipo de dato: *string*

La fecha actual en una cadena.

El formato estándar de la fecha es "año-mes-día", por ejemplo: "93-05-16".

Ejemplo

```
fecha := CDate();  
TPWrite "La fecha actual es: "+fecha;  
Write archivo_event , fecha;
```

La fecha actual será visualizada en la pantalla de la unidad de programación y en un archivo de texto.

Sintaxis

CDate '(' ')

Una función con un valor de retorno del tipo *string*.

Información relacionada

Descripción:

Instrucciones de hora
Activación del reloj del sistema

Resumen RAPID - *Sistema y Hora*
Parámetros del Sistema

CJointT Lectura de los ángulos actuales de los ejes

CJointT (*Current Joint Target*) sirve para leer los ángulos actuales de los ejes del robot y de los ejes externos.

Ejemplo

```
VAR jointtarget joints;
```

```
joints := CJointT();
```

Los ángulos actuales de los ejes del robot y de los ejes externos están almacenados en *joints*.

Valor de retorno

Tipo de dato: *jointtarget*

Los ángulos actuales de los ejes del robot, del lado del brazo, expresados en grados.

Los valores actuales de los ejes externos, expresados en mm para los ejes lineales, y en grados para los ejes de rotación.

Los valores devueltos se refieren a la posición de calibración.

Sintaxis

`CJointT('')`

Una función con un valor de retorno del tipo de dato *jointtarget*.

Información relacionada

Descripción:

Definición del eje

Tipos de Datos - *jointtarget*

Lectura del ángulo actual del motor

Funciones - *ReadMotor*

ClkRead**Lectura de un reloj utilizado para el cronometraje**

ClkRead sirve para la lectura de un reloj que funciona como un cronómetro.

Ejemplo

```
reg1:=ClkRead(reloj1);
```

El reloj *reloj1* será leído y la hora quedará almacenada en segundos en la variable *reg1*.

Valor de Retorno

Tipo de dato: *num*

La hora, en segundos, almacenada en el reloj.

Argumento**ClkRead (Reloj)****Reloj**

Tipo de dato: *clock*

El nombre del reloj que se desea leer.

Ejecución del programa

Un reloj podrá ser leído cuando está parado o cuando está funcionando.

Una vez que se ha leído el reloj, se podrá volver a leer, volver a arrancar, parar o poner a cero.

Si el reloj se desborda, la ejecución del programa se detiene y aparece un mensaje de error.

Sintaxis

```
ClkRead '('  
[ Reloj ':=' ] <variable (VAR) of clock > ')'
```

Una función con un valor de retorno del tipo *num*.

Información relacionada

Descripción:

Instrucciones de reloj
Saturación del reloj
Más ejemplos

Resumen RAPID - *Sistema y Hora*
Tipos de dato - *clock*
Instrucciones - *ClkStart*

CorrRead Lectura de todos los offsets utilizados

CorrRead sirve para leer todas las correcciones proporcionadas por todos los generadores de corrección conectados.

CorrRead puede utilizarse para:

- determinar cuánto difiere la trayectoria actual respecto a la trayectoria original.
- tomar medidas para reducir la diferencia.

Ejemplo

```
VAR pos offset;  
...  
offset := CorrRead();
```

Los offsets actuales proporcionados por todos los generadores de corrección conectados están disponibles en la variable *offset*.

Valor de retorno

Tipo de dato: *pos*

Los offsets absolutos totales proporcionados hasta el momento por todos los generadores de corrección conectados.

Ejemplo

Véase Instrucciones - *CorrCon*

Sintaxis

CorrRead '()''

Una función con un valor de retorno del tipo de dato *pos*.

Información relacionada

Descripción:

Conexión a un generador de correcciones	Instrucciones - <i>CorrCon</i>
Desconexión de un generador de correcciones	Instrucciones - <i>CorrDiscon</i>
Escritura en un generador de correcciones	Instrucciones - <i>CorrWrite</i>
Eliminación de todos los generadores de correcciones	Instrucciones - <i>CorrClear</i>
Descriptor de correcciones	Tipos de datos - <i>corrdescr</i>

Cos**Cálculo del valor del coseno**

Cos (Cosine) sirve para calcular el valor del coseno a partir del valor de un ángulo.

Ejemplo

```
VAR num ángulo;  
VAR num valor;
```

```
.
```

```
.
```

```
valor := Cos(ángulo);
```

Valor de retornoTipo de dato: *num*

El valor del coseno, comprendido entre [-1, 1].

Argumentos**Cos (Angulo)****Angulo**Tipo de dato: *num*

El valor del ángulo, expresado en grados.

Sintaxis

```
Cos'('  
[Angulo ':='] <expresión (IN) de num>  
'')
```

Una función con un valor de retorno del tipo de dato *num*.

Información relacionadaDescripción:

Instrucciones y funciones matemáticas

Resumen RAPID - *Matemáticas*

Cos

Funciones

CPos Lectura de los datos de posición actuales (pos)

CPos (Current Position) sirve para leer la posición actual del robot.

Esta función devuelve los valores x, y, z del TCP del robot como datos del tipo *pos*. Si se desea leer la posición completa del robot (*robtarget*) se deberá, en vez de ello, utilizar la función *CRobT*.

Ejemplo

```
VAR pos pos1;
pos1 := CPos(Tool:=tool1 \WObj:=wobj0);
```

La posición actual del TCP del robot será almacenada en la variable *pos1*. La herramienta *tool1* y el objeto de trabajo *wobj0* sirven para calcular la posición.

Valor de retorno

Tipo de dato: *pos*

La posición actual (*pos*) del robot con x, y, z en el sistema de coordenadas más externo, teniendo en cuenta el sistema de coordenadas ProgDisp activo, la herramienta y el objeto de trabajo especificados.

Argumentos

CPos ([\Tool] [\WObj])

[\Tool]

Tipo de dato: *tooldata*

La herramienta utilizada para el cálculo de la posición actual del robot.

En el caso en que se omita este argumento, el sistema utilizará la herramienta activa actual.

[\WObj]

(*Objeto de trabajo*)

Tipo de dato: *wobjdata*

El objeto de trabajo (sistema de coordenadas) al que se refiere la posición actual del robot retornada por la función.

En el caso en que se omita este argumento, el sistema utilizará el objeto de trabajo actual activado.

En una buena programación se deberá especificar siempre el argumento \Tool y \WObj. Entonces, la función siempre retorna la posición deseada, incluso si se ha activado manualmente alguna otra herramienta u objeto de trabajo.

Ejecución del programa

Las coordenadas devueltas indican la posición del TCP en el sistema de coordenadas ProgDisp.

Ejemplo

```

VAR pos pos2;
VAR pos pos3;
VAR pos pos4;

pos2 := CPos(Tool:=grip3 \WObj:=fixture);
.
.
.
pos3 := CPos(Tool:=grip3 \WObj:=fixture);
pos4 := pos3-pos2;

```

La posición x, y, z del robot está tomada en dos lugares en el programa utilizando la función *CPos*. La herramienta *grip3* y el objeto de trabajo *fixture* sirven para calcular la posición. Las distancias x, y, z recorridas entre estas posiciones serán entonces calculadas y almacenadas en la variable *pos p4*.

Sintaxis

```

CPos '()
[\'Tool ':= <persistente (PERS) de tooldata>]
[\'WObj ':= <persistente (PERS) de wobjdata>] ')'

```

Una función con un valor de retorno del tipo de dato *pos*.

Información relacionada

	<u>Descripción:</u>
Definición de la posición	Tipos de datos - <i>pos</i>
Definición de las herramientas	Tipos de datos - <i>tooldata</i>
Definición de los objetos de trabajo	Tipos de datos - <i>wobjdata</i>
Sistemas de coordenadas	Principios de Movimiento y de E/S - <i>Sistemas de Coordenadas</i>
Lectura de <i>robtarget</i> utilizado	Funciones - <i>CRobT</i>

CRobT**Lectura de los datos de la posición actuales (robtarget)**

CRobT (*Current Robot Target*) sirve para leer la posición actual del robot y de los ejes externos.

Esta función devuelve un valor *robtarget* con la posición (x, y, z), la orientación (q1 ... q4), la configuración de los ejes del robot y la posición de los ejes externos. En el caso en que sólo se deba leer los valores x, y, z del TCP del robot (*pos*), se deberá en vez de ello, utilizar la función *CPos*.

Ejemplo

```
VAR robtarget p1;
p1 := CRobT(\Tool:=tool1 \WObj:=wobj0);
```

La posición actual del robot y de los ejes externos será almacenada en *p1*. La herramienta *tool1* y el objeto de trabajo *wobj0* sirven para calcular la posición.

Valor de retorno

Tipo de dato: *robtarget*

La posición actual del robot y de los ejes externos en el sistema de coordenadas más externo, teniendo en cuenta el sistema de coordenadas ProgDisp/ExtOffs, la herramienta y el objeto de trabajo especificados.

Argumentos**CRobT ([\Tool] [\WObj])****[\Tool]**

Tipo de dato: *tooldata*

La herramienta utilizada para el cálculo de la posición actual del robot.

En el caso en que se omita este argumento, se utilizará la herramienta actual activada.

[\WObj]

(*Objeto de Trabajo*)

Tipo de dato: *wobjdata*

El objeto de trabajo (sistema de coordenadas) al que se refiere la posición actual del robot retornada por la función.

En el caso en que se omita este argumento, se utilizará el objeto de trabajo actual activado.

En una buena programación se deberá siempre especificar el argumento \Tool y

\WObj. Entonces, la función siempre retornará la posición deseada, incluso si se ha activado manualmente alguna otra herramienta u objeto de trabajo.

Ejecución del programa

Las coordenadas devueltas indican la posición del TCP en el sistema de coordenadas ProgDisp. Los ejes externos están representados en el sistema de coordenadas ExtOffs.

Ejemplo

```
VAR robtarget p2;
p2 := ORobT( RobT(\Tool:=grip3 \WObj:=fixture) );
```

La posición actual en el sistema de coordenadas del objeto (sin ningún ProgDisp ni ExtOffs) del robot y de los ejes externos será almacenada en *p2*. La herramienta *grip3* y el objeto de trabajo *fixture* sirven para calcular la posición.

Sintaxis

```
CRobT '(
  [\Tool :=] <persistente (PERS) de tooldata>
  [\WObj :=] <persistente (PERS) de wobjdata>
)'
```

Una función con un valor de retorno del tipo de dato *robtarget*.

Información relacionada

	<u>Descripción:</u>
Definición de la posición	Tipos de datos - <i>robtarget</i>
Definición de las herramientas	Tipos de datos - <i>tooldata</i>
Definición de los objetos de trabajo	Tipos de datos - <i>wobjdata</i>
Sistemas de coordenadas	Principios de Movimiento y de E/S - <i>Sistemas de Coordenadas</i>
Sistema de coordenadas ExtOffs	Instrucciones - <i>EOffsOn</i>
Lectura de la <i>pos</i> actual (sólo x, y, z)	Funciones - <i>CPos</i>

CTime**Lectura de la hora actual
como una cadena**

CTime sirve para leer la hora actual del sistema.

Esta función sirve para hacer aparecer la hora actual en el visualizador de la unidad de programación o para introducir la hora actual en un archivo de texto.

Ejemplo

```
VAR string tiempo;
```

```
tiempo := CTime();
```

La hora actual será almacenada en la variable *tiempo*.

Valor de Retorno

Tipo de dato: *string*

La hora actual en una cadena.

El formato estándar de la hora es "horas:minutos:segundos", por ejemplo: "18:20:46".

Ejemplo

```
tiempo := CTime();
TPWrite “La hora actual es: “+tiempo;
Write archivo_event, tiempo;
```

La hora actual será visualizada en la pantalla de la unidad de programación y quedará almacenada en un archivo de texto.

Sintaxis

CTime '()''

Una función con un valor de retorno del tipo *string*.

Información relacionada

Instrucciones de fecha y hora
Activación del reloj del sistema

Descripción en:

Resumen RAPID - *Sistema y Hora*
Parámetros del Sistema

CTool

Lectura de los datos de herramienta actuales

CTool (*Current Tool*) sirve para leer los datos de la herramienta utilizada.

Ejemplo

```
PERS tooldata temp_tool;  
temp_tool := CTool();
```

El valor de la herramienta actual es almacenado en la variable *temp_tool*.

Valor de retorno

Tipo de dato: *tooldata*

Esta función retorna un valor *tooldata* manteniendo el valor de la herramienta actual, es decir, de la herramienta que se utilizó por última vez en una instrucción de movimiento.

El valor returnedo representa la posición y la orientación del TCP en el sistema de coordenadas central de la muñeca, véase *tooldata*.

Sintaxis

`CTool'()'`

Una función con un valor de retorno del tipo de dato *tooldata*.

Información relacionada

Descripción:

Definición de las herramientas

Tipos de datos - *tooldata*

Sistemas de coordenadas

Principios de Movimiento y de E/S -
Sistemas de Coordenadas

CWObj Lectura de los datos del objeto de trabajo actuales

CWObj (Current Work Object) sirve para leer los datos del objeto de trabajo actual.

Ejemplo

```
PERS wobjdata temp_wobj;
```

```
temp_wobj := CWObj();
```

El valor del objeto de trabajo utilizado es almacenado en la variable *temp_wobj*.

Valor de retorno

Tipo de dato: *wobjdata*

Esta función retorna un valor *wobjdata* manteniendo el valor del objeto de trabajo actual, es decir el objeto de trabajo utilizado en último lugar en una instrucción de movimiento.

El valor devuelto representa la posición del objeto de trabajo y la orientación en el sistema de coordenadas mundo, véase *wobjdata*.

Sintaxis

```
CWObj('')
```

Una función con un valor de retorno del tipo de dato *wobjdata*.

Información relacionada

Descripción:

Definición de los objetos de trabajo

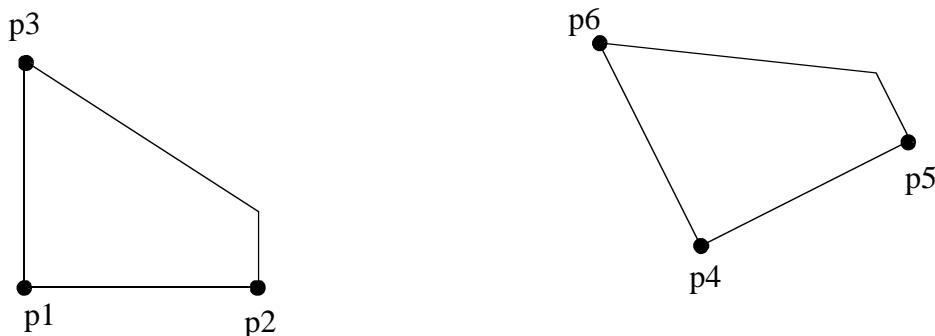
Tipos de datos - *wobjdata*

Sistemas de coordenadas

Principios de movimiento y de E/S -
Sistemas de Coordenadas

DefDFrame**Definición de una base
de desplazamiento**

DefDFrame (Define Displacement Frame) sirve para calcular una base de desplazamiento a partir de tres posiciones de origen y de tres posiciones desplazadas.

Ejemplo

Se han almacenado tres posiciones, $p1$ a $p3$, relativos a un objeto en una posición de origen. Después de haber realizado el desplazamiento del objeto, las mismas posiciones son buscadas y almacenadas como $p4$ a $p6$. A partir de estas seis posiciones, se calculará la base de desplazamiento. A continuación, la base calculada servirá para desplazar todas las posiciones almacenadas en el programa.

```

CONST robtarget p1 := [...];
CONST robtarget p2 := [...];
CONST robtarget p3 := [...];
VAR robtarget p4;
VAR robtarget p5;
VAR robtarget p6;
VAR pose base1;

!Buscar las posiciones nuevas
SearchL sen1, p4, *, v50, herram1;
.
SearchL sen1, p5, *, v50, herram1;
.
SearchL sen1, p6, *, v50, herram1;
frame1 := DefDframe (p1, p2, p3, p4, p5, p6);

!activación del desplazamiento definido por base1
PDispSet base1;

```

Valor de retorno

Tipo de dato: *pose*

La base de desplazamiento.

Argumentos

**DefDFrame (AntiguoP1 AntiguoP2 AntiguoP3 NuevoP1
NuevoP2 NuevoP3)**

AntiguoP1

Tipo de dato: *robtarget*

La primera posición de origen.

AntiguoP2

Tipo de dato: *robtarget*

La segunda posición de origen.

AntiguoP3

Tipo de dato: *robtarget*

La tercera posición de origen.

NuevoP1

Tipo de dato: *robtarget*

La primera posición desplazada. Esta posición deberá ser medida y determinada con gran precisión.

NuevoP2

Tipo de dato: *robtarget*

La segunda posición desplazada. Deberá observarse que esta posición podrá ser medida y determinada con menor precisión en una dirección, por ejemplo, esta posición deberá colocarse en una línea que describe una nueva dirección de *p1* a *p2*.

NuevoP3

Tipo de dato: *robtarget*

La tercera posición desplazada. Esta posición podrá ser medida y determinada con menor precisión en dos direcciones, por ejemplo, deberá ser colocada en un plano que describe el plano nuevo para *p1*, *p2* y *p3*.

Sintaxis

```
DefDFrame'(
    [AntiguoP1 ':=] <expresión (IN) de robtarget> ',' 
    [AntiguoP2 ':=] <expresión (IN) de robtarget> ',' 
    [AntiguoP3 ':=] <expresión (IN) de robtarget> ',' 
    [NuevoP1 ':=] <expresión (IN) de robtarget> ',' 
    [NuevoP2 ':=] <expresión (IN) de robtarget> ',' 
    [NuevoP3 ':=] <expresión (IN) de robtarget> ')'
```

Una función con un valor de retorno del tipo de dato *pose*.

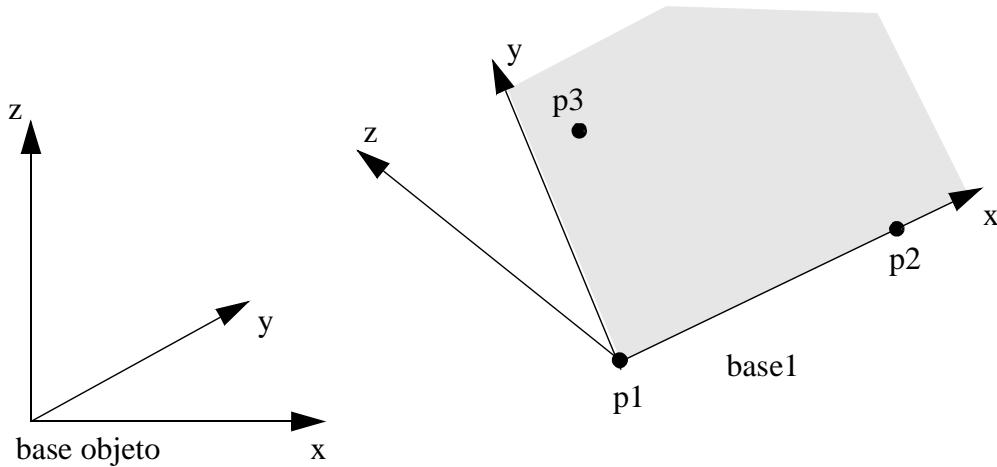
Información relacionada

	<u>Descripción:</u>
Activación de la base de desplazamiento	Instrucciones - <i>PDispSet</i>
Definición manual de la base de desplazamiento	Programación y Pruebas

DefFrame Definición de una base de coordenadas

DefFrame (Define Frame) sirve para calcular una base de coordenadas, a partir de tres posiciones que definen la base.

Ejemplo



Las tres posiciones, $p1$ a $p3$, relativas al sistema de coordenadas del objeto, son utilizadas para definir el sistema de coordenadas nuevo, $base1$. La primera posición, $p1$, define el origen de $base1$, la segunda posición, $p2$, define la dirección del eje-x y la tercera posición, $p3$, define la situación del plano xy-. La base $base1$ definida puede ser utilizada como base de desplazamiento, según se indica en el ejemplo siguiente:

```

CONST robtarget p1 := [...];
CONST robtarget p2 := [...];
CONST robtarget p3 := [...];
VAR pose base1;
.
.
.
base1:= DefFrame (p1, p2, p3);
.
.
.
!activación del desplazamiento definido por base1
PDispSet base1;

```

Valor de retorno

Tipo de dato: *pose*

La base calculada.

El cálculo se refiere al sistema de coordenadas del objeto activado.

Argumentos

DefFrame (NuevoP1 NuevoP2 NuevoP3 [Origen])

NuevoP1

Tipo de dato: *robtarget*

La primera posición, que definirá el origen de la nueva base.

NuevoP2

Tipo de dato: *robtarget*

La segunda posición, que definirá la dirección del eje-x de la nueva base.

NuevoP3

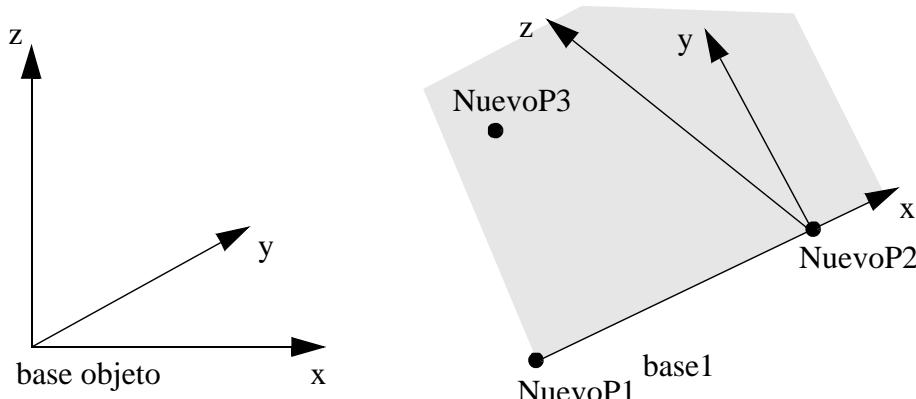
Tipo de dato: *robtarget*

La tercera posición, que definirá el plano xy- de la nueva base. La posición del punto 3 se encontrará en el lado positivo y, según se indica en la figura de la página anterior.

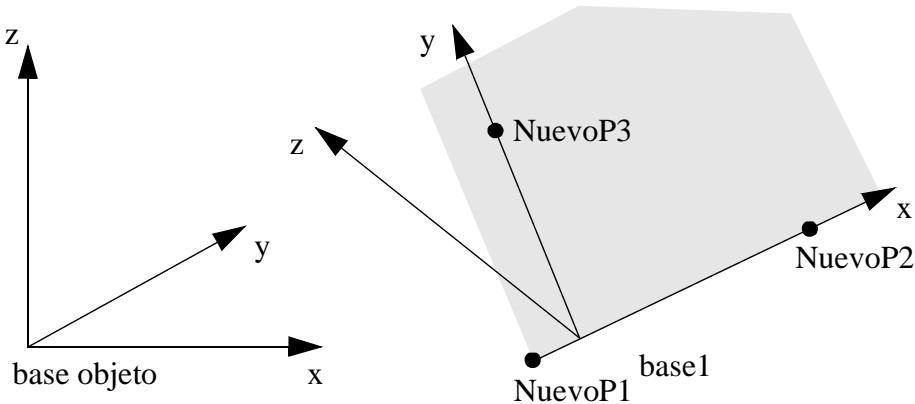
[Origen]

Tipo de dato: *num*

Es el argumento opcional, que definirá como se colocará el origen de la base de coordenadas. Origen = 1, significa que el origen está colocado en NuevoP1, es decir, el mismo que cuando este argumento es omitido. Origen = 2, significa que el origen está colocado en NuevoP2, según se indica en la siguiente figura.



Origen = 3, significa que el origen está colocado en la línea que pasa por NuevoP1 y NuevoP2, y por consiguiente NuevoP3 estará colocado en el eje y-, según se indica en la siguiente figura.



Otros valores, o si se omite el Origen, pondrán el punto de origen en NuevoP1.

Sintaxis

```
DefFrame'(
    [NuevoP1 ':='] <expresión (IN) de robtarget> ',' 
    [NuevoP2 ':='] <expresión (IN) de robtarget> ',' 
    [NuevoP3 ':='] <expresión (IN) de robtarget> ')'
    ['\''Origin ':='] <expresión (IN) de num> ]')
```

Una función con un valor de retorno del tipo de dato *pose*.

Información relacionada

Instrucciones y funciones matemáticas
Activación de la base de desplazamiento

Descripción:

Resumen RAPID - *Matemáticas*
Instrucciones - *PDispSet*

Dim**Obtención del tamaño de una matriz**

Dim (Dimension) sirve para obtener el número de elementos de una matriz.

Ejemplo

```
PROC mulmat(VAR num array{ * }, num factor)
  FOR index FROM 1 TO Dim(array, 1) DO
    array{index} := array{index} * factor;
  ENDFOR
ENDPROC
```

Todos los elementos de una matriz num son multiplicados por un factor.
Este procedimiento puede tomar cualquier matriz unidimensional del tipo de dato *num* como entrada.

Valor de Retorno

Tipo de dato: *num*

El número de elementos de matriz de la dimensión especificada.

Argumentos**Dim (ParMat NumDim)****Parmat**

(*Parámetro matriz*)

Tipo de dato: Cualquier tipo

El nombre de la matriz.

NumDim

(*Número Dimensión*)

Tipo de dato: *num*

La dimensión de matriz deseada:

1 = unidimensional

2 = bidimensional

3 = tridimensional

Ejemplo

```

PROC sum_mat(VAR num array1{*,*,*}, num array2{*,*,*})

IF Dim(array1,1) <> Dim(array2,1) OR Dim(array1,2) <> Dim(array2,2) OR
    Dim(array1,3) <> Dim(array2,3) THEN
    TPWrite "El tamaño de las matrices no es el mismo";
    Stop;
ELSE
    FOR i1 FROM 1 TO Dim(array1, 1) DO
        FOR i2 FROM 1 TO Dim(array1, 2) DO
            FOR i3 FROM 1 TO Dim(array1, 3) DO
                array1{i1,i2,i3} := array1{i1,i2,i3} + array2{i1,i2,i3};
            ENDFOR
        ENDFOR
    ENDFOR
ENDIF
RETURN;

ENDPROC

```

Dos matrices han sido añadidas. Si el tamaño de las matrices difiere, el programa se detendrá y aparecerá un mensaje de error.

Este procedimiento podrá utilizar cualquier matriz tridimensional del tipo de dato *num* como una entrada.

Sintaxis

```

Dim '('
[ParMat':='] <referencia (REF) de cualquier tipo> ',' 
[NumDim':='] <expresión (IN) de num> ')'

```

Un parámetro REF requiere que el argumento correspondiente sea una constante, una variable o un dato entero persistente. El argumento podría ser también un parámetro IN, un parámetro VAR o un parámetro entero PERS.

Una función con un valor de retorno del tipo de dato *num*.

Información relacionada

Parámetros de matriz

Declaración de matriz

Descrito en:

Características Básicas - *Rutinas*

Características Básicas - *Datos*

DOutput**Lectura del valor de una señal
de salida digital**

DOutput sirve para leer el valor actual de una señal de salida digital.

Ejemplo

IF DOutput(do2) = 1 THEN . . .

Si el valor actual de la señal *do2* es igual a *1*, entonces. . .

Valor de Retorno

Tipo de dato: *dionum*

El valor actual de la señal (0 o 1).

Argumentos**DOutput (Señal)****Señal**

Tipo de dato: *signaldo*

El nombre de la señal que se desea leer.

Ejecución del programa

El valor leído depende de la configuración de la señal. Si la señal ha sido invertida en los parámetros del sistema, el valor returnedo por esta función será el opuesto del valor verdadero del canal físico.

Ejemplo

IF DOutput(modo_auto) <> activa THEN . . .

Si el valor actual de la señal *modo_auto* es *no activa*, entonces ..., Observar que la señal deberá primero haber sido definida como una salida del sistema en los parámetros del sistema.

Sintaxis

```
DOutput '('  
[ Señal ':=' ] <variable (VAR) of signaldo > ')'
```

Una función con un valor de retorno del tipo de dato *dionum*.

Información relacionada

Describo en:

Instrucciones de Entrada/Salida

Resumen RAPID -
Señales de Entrada y Salida

Funciones de Entrada/Salida
en general

Principios de Movimiento y de E/S -
Principios de E/S

Configuración de las E/S

Parámetros del Sistema

EulerZYX**Obtención de ángulos Euler a partir de una variable de orientación**

EulerZYX (rotaciones Euler ZYX) sirve para obtener una componente de ángulo Euler a partir de una variable del tipo orientación.

Ejemplo

```

VAR num angulox;
VAR num anguloy;
VAR num anguloz;
VAR pose object;

.

angulox := GetEuler(\X, object.rot);
anguloy := GetEuler(\Y, object.rot);
anguloz := GetEuler(\Z, object.rot);

```

Valor de retornoTipo de dato: *num*

El ángulo Euler correspondiente, expresado en grados y comprendido entre [-180, 180].

Argumentos**EulerZYX ([\X] | [\Y] | [\Z] Rotación)**

Los argumentos \X, \Y y \Z son mutuamente exclusivos. En el caso en que ninguno de ellos haya sido especificado, se generará un error de funcionamiento.

[\X]

Tipo de dato: *switch*

Obtiene la rotación en torno al eje X.

[\Y]

Tipo de dato: *switch*

Obtiene la rotación en torno al eje Y.

[\Z]

Tipo de dato: *switch*

Obtiene la rotación en torno al eje Z.

RotaciónTipo de dato: *orient*

La rotación en su representación de quaterniones.

Sintaxis

```
EulerZYX'('
  [\'X \','] | [\'Y \','] | [\'Z \',']
  Rotación ':='] <expresión (IN) de orient>
  ')'
```

Una función con un valor de retorno del tipo de dato *num.*

Información relacionada

Descripción en:

Instrucciones y funciones matemáticas

Resumen RAPID - *Matemáticas*

Exp**Cálculo del valor exponencial**

Exp (Exponential) sirve para calcular el valor exponencial, e^x .

Ejemplo

```
VAR num x;  
VAR num valor;  
. .  
valor:= Exp( x);
```

Valor de retorno

Tipo de dato: *num*

El valor exponencial e^x .

Argumentos**Exp (Exponente)**

Exponente

Tipo de dato: *num*

El valor del argumento exponente.

Sintaxis

Exp'(' [Exponente ':='] <expresión (**IN**) de *num*> ')'

Una función con un valor de retorno del tipo de dato *num*.

Información relacionada

Descripción en:

Instrucciones y funciones matemáticas

Resumen RAPID - *Matemáticas*

Exp

Funciones

GetTime**Lectura de la hora actual como un valor numérico**

GetTime sirve para leer un componente específico de la hora actual del sistema como un valor numérico.

GetTime podrá utilizarse para:

- hacer que el programa realice una operación a una hora específica,
- realizar ciertas actividades en un día de trabajo,
- impedir la realización de ciertas actividades durante el fin de semana,
- responder a errores de forma diferente según la hora del día.

Ejemplo

```
hora := GetTime(Hour);
```

La hora actual está almacenada en la variable *hora*.

Valor de Retorno

Tipo de dato: *num*

Uno de los cuatro componentes de hora especificados a continuación.

Argumentos

GetTime ([WDay] | [Hour] | [Min] | [Sec])

[WDay]

Tipo de dato: *switch*

Retorna el día de la semana actual.
Gama: de 1 a 7 (de lunes a domingo).

[Hour]

Tipo de dato: *switch*

Retorna la hora actual.
Gama: de 0 a 23.

[Min]

Tipo de dato: *switch*

Retorna el minuto actual.
Gama: de 0 a 59.

[Sec]

Tipo de dato: *switch*

Retorna el segundo actual.
Gama: de 0 a 59.

Se deberá especificar uno de los argumentos, de lo contrario la ejecución del programa se detendrá y el sistema producirá un mensaje de error.

Ejemplo

```
dia_sem:= GetTime(\WDay);
hora := GetTime(\Hour);
IF dia_sem< 6 AND hora >6 AND hora < 16 THEN
    producción;
ELSE
    mantenimiento;
ENDIF
```

Si es un día de trabajo y que la hora está comprendida entre 7:00 y 15:59 el robot llevará a cabo el proceso de producción. A cualquier otra hora, el robot permanecerá en el modo de mantenimiento.

Sintaxis

```
GetTime '('
[ '\' WDay ]
| [ '\' Hour ]
| [ '\' Min ]
| [ '\' Sec ] ')'
```

Una función con un valor de retorno del tipo *num*.

Información relacionada

Descrito en:

Instrucciones de fecha y hora
Activación del reloj del sistema

Resumen RAPID - *Sistema y Hora*
Parámetros del Sistema

GOutput**Lectura del valor de un grupo
de señales de salida digital**

GOutput sirve para leer el valor actual de un grupo de señales de salida digital.

Ejemplo

IF GOutput(go2) = 5 THEN ...

Si el valor actual de la señal *go2* es igual a 5, entonces ...

Valor de Retorno

Tipo de dato: *num*

El valor actual de la señal (un número entero positivo).

Los valores de cada señal del grupo son leídos e interpretados como un número binario sin signo. Este número binario será luego convertido en un número entero.

El valor devuelto se encuentra dentro de unos límites que dependen del número de señales del grupo.

Nº de señales	Valor de retorno	Nº de señales	Valor de retorno
1	0 - 1	9	0 - 511
2	0 - 3	10	0 - 1023
3	0 - 7	11	0 - 2047
4	0 - 15	12	0 - 4095
5	0 - 31	13	0 - 8191
6	0 - 63	14	0 - 16383
7	0 - 127	15	0 - 32767
8	0 - 255	16	0 - 65535

Argumentos**GOutput (Señal)**

Señal

Tipo de dato: *signalgo*

El nombre del grupo de señales que se desea leer.

Sintaxis

```
GOutput '(
    [ Señal ':=' ] <variable (VAR) of signalgo > )'
```

Una función con un valor de retorno del tipo de dato *num*.

Información relacionada

Descripción en:

Instrucciones de Entrada/Salida

Resumen RAPID -
Señales de Entrada y Salida

Funciones de Entrada/Salida
en general

Principios de Movimiento y de E/S -
Principios de E/S

Configuración de las E/S

Parámetros del Sistema

IndInpos**Estado de la posición independiente**

IndInpos sirve para comprobar si un eje independiente ha alcanzado la posición seleccionada.

Ejemplo

```
IndAMove Station_A,1\ToAbsNum:=90,20;
WaitUntil IndInpos(Station_A,2) = TRUE;
WaitTime 0.2;
```

Esperar hasta que el eje 1 de *Station_A* esté en la posición de 90 grados.

Valor de retornoTipo de dato: *bool*

Los valores de retorno desde *IndInpos* son:

<u>Valor de retorno</u>	<u>Estado del eje</u>
TRUE	En posición y tiene velocidad cero.
FALSE	No está en posición y/o no tiene velocidad cero.

Argumentos**IndInpos MecUnit Axis**

MecUnit	(<i>Unidad Mecánica</i>)	Tipo de dato: <i>mecunit</i>
----------------	----------------------------	------------------------------

El nombre de la unidad mecánica.

Axis	Tipo de dato: <i>num</i>
-------------	--------------------------

El número del eje utilizado por la unidad mecánica (1-6).

Limitaciones

Un eje independiente ejecutado con la instrucción *IndCMove* siempre retornará el valor FALSE, incluso cuando la velocidad es cero.

Un periodo de espera de 0,2 segundos deberá ser añadido después de la instrucción, para garantizar que el estado correcto ha sido conseguido. Este periodo de tiempo deberá ser mayor para los ejes externos de poca capacidad.

Gestión de errores

Si el eje no está en el modo independiente, la variable del sistema ERRNO será activada en ERR_AXIS_IND.

Sintaxis

```
IndInpos '('  
[ MecUnit':=' ] <variable (VAR) de mecunit> ','  
[ Axis':=' ] <expresión (IN) de num> ')'
```

Una función con un valor de retorno del tipo de dato *bool*.

Información relacionada

Descripción:

Ejes independientes en general

Principios de Movimiento y de E/S -
Ejecución del programa

Comprobar el estado de velocidad de los
ejes independientes

Funciones - *IndSpeed*

IndSpeed Estado de la velocidad independiente

IndSpeed sirve para comprobar si un eje independiente ha alcanzado la velocidad seleccionada.

Ejemplo

```
IndCMove Station_A, 2, 3.4;
WaitUntil IndSpeed(Station_A,2 \InSpeed) = TRUE;
WaitTime 0.2;
```

Esperar hasta que el eje 2 de *Station_A* haya alcanzado la velocidad de 3,4 grados/s.

Valor de retorno

Tipo de dato: *bool*

Los valores de retorno desde *IndSpeed* son:

<u>Valor de retorno</u>	<u>Valor de estado</u>
-------------------------	------------------------

opción *\InSpeed*

TRUE	Ha alcanzado la velocidad seleccionada.
FALSE	No ha alcanzado la velocidad seleccionada.

opción *\ZeroSpeed*

TRUE	Velocidad cero.
FALSE	Sin velocidad cero.

Argumentos

IndSpeed MecUnit Axis [\InSpeed] | [\ZeroSpeed]

MecUnit	(<i>Unidad Mecánica</i>)	Tipo de dato: <i>mecunit</i>
----------------	----------------------------	------------------------------

El nombre de la unidad mecánica.

Axis	Tipo de dato: <i>num</i>
-------------	--------------------------

El número del eje utilizado por la unidad mecánica (1-6).

[\InSpeed]	Tipo de dato: <i>switch</i>
---------------------	-----------------------------

IndSpeed retorna el valor TRUE si el eje ha alcanzado la velocidad seleccionada, de lo contrario, retorna el valor FALSE.

[\ZeroSpeed]Tipo de dato: *switch*

IndSpeed retorna el valor TRUE si el eje tiene una velocidad cero, de lo contrario, retorna el valor FALSE.

Si se omiten ambos argumentos *\InSpeed* y *\ZeroSpeed*, aparecerá visualizado un mensaje de error.

Limitación

La función *IndSpeed\InSpeed* retornará siempre el valor FALSE en las siguientes situaciones:

- El robot está en modo manual con velocidad reducida.
- La velocidad es reducida utilizando la instrucción *VelSet*.
- La velocidad es reducida desde la ventana de producción.

Un periodo de espera de 0,2 segundos deberá ser añadido después de la instrucción, para garantizar que el estado correcto ha sido conseguido. Este periodo de tiempo deberá ser mayor para los ejes externos de poca capacidad.

Gestión de errores

Si el eje no está en el modo independiente, la variable del sistema ERRNO será activada en **ERR_AXIS_IND**.

Sintaxis

```
IndSpeed '('
    [ MecUnit'=' ] < variable (VAR) de mecunit > ',' 
    [ Axis'=' ] < expresión (IN) de num >
    [ '\' InSpeed ] | [ '\' ZeroSpeed ] ')'
```

Una función con un valor de retorno del tipo de dato *bool*.

Información relacionada

Ejes independientes en general

Descripción:

Principios de Movimiento y de E/S-
Ejecución del programa

Más ejemplos

Instrucciones - *IndCMove*

Comprobar el estado de posición de los
ejes independientes

Funciones - *IndInpos*

IsPers**Es Persistente**

IsPers sirve para comprobar si un dato de objeto es una variable persistente o no.

Ejemplo

```
PROC procedure1 (INOUT num parameter1)
  IF IsVar(parameter1) THEN
    ! referencia de llamada a una variable
    ...
  ELSEIF IsPers(parameter1) THEN
    ! referencia de llamada a una variable persistente
    ...
  ELSE
    ! No debe ocurrir
    EXIT;
  ENDIF
ENDPROC
```

El procedimiento *procedure1* ejecutará diferentes acciones dependiendo de si el parámetro utilizado *parameter1* es una variable o una variable persistente.

Valor de retorno

Tipo de dato: *bool*

TRUE en el caso en que el parámetro INOUT utilizado que se ha comprobado es una variable persistente.

FALSE en el caso en que el parámetro INOUT utilizado que se ha comprobado no es una variable persistente.

Argumentos**IsPers (DatObj)**

DatObj

(*Dato Objeto*)

Tipo de dato: cualquiera

Es el nombre del parámetro INOUT formal.

Sintaxis

IsPers'(' [DatObj ':='] < var o pers (**INOUT** de cualquier tipo) > ')'

Una función con un valor de retorno del tipo de dato *bool*.

Información relacionada

Comprobar si es una variable

Tipos de parámetros (modos de acceso)

Descripción:

Función - *IsVar*

Características RAPID - *Rutinas*

IsVar**Es Variable**

IsVar sirve para comprobar si un dato de objeto es una variable o no.

Ejemplo

```
PROC procedure1 (INOUT num parameter1)
  IF IsVAR(parameter1) THEN
    ! referencia de llamada a una variable
    ...
  ELSEIF IsPers(parameter1) THEN
    ! referencia de llamada a una variable persistente
    ...
  ELSE
    ! No debe ocurrir
    EXIT;
  ENDIF
ENDPROC
```

El procedimiento *procedure1* ejecutará diferentes acciones dependiendo de si el parámetro utilizado *parameter1* es una variable o una variable persistente.

Valor de retorno

Tipo de dato: *bool*

TRUE en el caso en que el parámetro INOUT utilizado que se ha comprobado es una variable.

FALSE en el caso en que el parámetro INOUT utilizado que se ha comprobado no es una variable.

Argumentos**IsVar (DatObj)**

DatObj

(*Dato Objeto*)

Tipo de dato: cualquiera

Es el nombre del parámetro INOUT formal.

Sintaxis

IsVar'(' [DatObj ':='] < var o pers (**INOUT** de cualquier tipo) > ')'

Una función con un valor de retorno del tipo de dato *bool*.

Información relacionada

Descripción:

Comprobar si es persistente

Función - *IsPers*

Tipos de parámetros (modos de acceso)

Características RAPID - *Rutinas*

MirPos**Creación de la imagen espejo
de una posición**

MirPos (Mirror Position) sirve para crear una imagen espejo de la rotación y translación de una posición.

Ejemplo

```
CONST robtarget p1;
VAR robtarget p2;
PERS wobjdata mirror;
.
.
p2 := MirPos(p1, mirror);
```

p1 es un dato robtarget que almacena una posición del robot y la orientación de la herramienta. Se creará una imagen espejo de esta posición en el plano xy- de la base definida por *mirror*, relativa al sistema de coordenadas mundo. El resultado es un dato robtarget nuevo, que está almacenado en *p2*.

Valor de retorno

Tipo de dato: *robtarget*

La posición nueva que es la posición espejo creada de la posición de entrada.

Argumentos

MirPos (Punto PlanoEspej [WObj] [\MirY])

Punto

Tipo de dato: *robtarget*

Es la posición de entrada del robot. La parte de orientación de esta posición define la orientación utilizada del sistema de coordenadas de la herramienta.

PlanoEspej

(*Plano Espejo*)

Tipo de dato: *wobjdata*

Son los datos del objeto de trabajo que definen el plano espejo. El plano espejo es el plano xy- de la base del objeto definida en *PlanoEspej*. La situación de la base del objeto será definida respecto a la base del usuario, también definida en *PlanoEspej*, que, a su vez, será definida respecto a la base de coordenadas mundo.

[WObj]

(*Objeto de Trabajo*)

Tipo de dato: *wobjdata*

Son los datos del objeto de trabajo que definen la base del objeto, y la base del usuario, respecto a las cuales está definida la posición de entrada, *Point*. Si se omite este argumento, la posición será definida respecto al sistema de coordenadas mundo.

Nota Si la posición ha sido creada con un objeto de trabajo activado, se deberá referir a este objeto de trabajo en el argumento.

[**MirY**]

(*Espejo Y*)

Tipo de dato: *switch*

Si se omite esto, que es la regla por defecto, se creará una imagen espejo de la base de la herramienta con respecto a los ejes x- y z-. Si se especifica este argumento, se creará una imagen espejo de la base de la herramienta con respecto a los ejes y- y z-.

Limitaciones

No se volverá a realizar un cálculo de la parte de configuración del robot relativa al dato de entrada *robtarget*.

Sintaxis

```
MirPos'('
  [ Punto':=' ] <expresión (IN) de robtarget> ',' 
  [PlanoEspej ':='] <expresión (IN) de wobjdata> ',' 
  ['\'WObj ':=' <expresión (IN) de wobjdata> ]
  ['\'MirY ']')
```

Una función con un valor de retorno del tipo de dato *robtarget*.

Información relacionada

Descripción:

Instrucciones y funciones matemáticas

Resumen RAPID - *Matemáticas*

NumToStr Conversión de un valor numérico en cadena

NumToStr (Numeric To String) sirve para convertir un valor numérico en un cadena.

Ejemplo

```
VAR string str;
```

```
str := NumToStr(0.38521,3);
```

La variable *str* toma el valor "0.385".

```
reg1 := 0.38521
```

```
str := NumToStr(reg1, 2\Exp);
```

La variable *str* toma el valor "3.85E-01".

Valor de retorno

Tipo de dato: *string*

El valor numérico convertido en una cadena con el número especificado de decimales, indicando también el exponente si se requiere. El valor numérico será redondeado si es necesario. La coma del decimal será suprimida si no se incluye ningún decimal.

Argumentos

NumToStr (Val Dec [\Exp])

Val	(<i>Valor</i>)	Tipo de dato: <i>num</i>
------------	------------------	--------------------------

Es el valor numérico que se desea convertir.

Dec	(<i>Decimales</i>)	Tipo de dato: <i>num</i>
------------	----------------------	--------------------------

Es el número de decimales. El número de decimales no deberá ser negativo ni mayor que la precisión disponible para los valores numéricos.

[\Exp]	(<i>Exponente</i>)	Tipo de dato: <i>switch</i>
---------------	----------------------	-----------------------------

En el caso en que se utilice un exponente.

Sintaxis

```
NumToStr(''  
[ Val ':=' ] <expresión (IN) de num> ','  
[ Dec ':=' ] <expresión (IN) de num>  
[ \Exp ]  
'')
```

Una función con un valor de retorno del tipo de dato *string*.

Información relacionada

Descripción:

Funciones de las cadenas

Resumen RAPID - *Funciones de cadena*

Definición de una cadena

Tipos de Datos - *string*

Valores de las cadenas

Características Básicas -
Elementos Básicos

Offs**Desplazamiento de una posición del robot**

Offs sirve para añadir un offset a una posición del robot.

Ejemplos

MoveL Offs(p2, 0, 0, 10), v1000, z50, herramienta1;

El robot se moverá a un punto que se encuentra a 10 mm de la posición *p2* (en la dirección z-).

p1 := Offs (p1, 5, 10, 15);

La posición del robot *p1* será desplazada de 5 mm en la dirección x-, de 10 mm en la dirección y- y de 15 mm en la dirección z-.

Valor de Retorno

Tipo de dato: *robtarget*

El dato de la posición desplazada.

Argumentos**Offs (Punto OffsetX OffsetY OffsetZ)****Punto**

Tipo de dato: *robtarget*

El dato de la posición que se desea desplazar.

OffsetX

Tipo de dato: *num*

El desplazamiento en la dirección x.

OffsetY

Tipo de dato: *num*

El desplazamiento en la dirección y.

OffsetZ

Tipo de dato: *num*

El desplazamiento en la dirección z.

Ejemplo

```
PROC pallet (num fila, num columna, num distancia, PERS tooldata herra,
            PERS wobjdata wobj)
```

```
VAR robtarget pospallet:=[[0, 0, 0], [1, 0, 0, 0], [0, 0, 0, 0],
                           [9E9, 9E9, 9E9, 9E9, 9E9, 9E9]];
```

```
pospalett := Offs (pospalett, (fila-1)*distancia, (columna-1)*distancia, 0);
MoveL pospalett, v100, fine, herra\WObj:=wobj;
```

```
ENDPROC
```

Se ha concebido una rutina para coger objetos de un pallet. Cada pallet ha sido definido como un objeto de trabajo (véase la Figura 1). La pieza que debe ser cogida (línea y columna) y la distancia entre los objetos deberán proporcionarse como parámetros de entrada.

Para aumentar el índice de líneas y columnas, se deberá salir de la rutina.

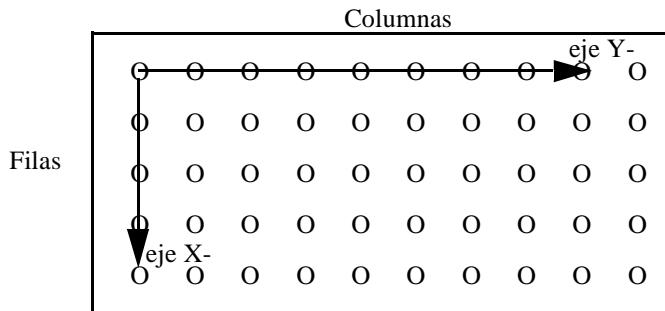


Figura 1 La posición y la orientación de un pallet serán especificadas definiendo un objeto de trabajo.

Sintaxis

```
Offs '('
[Punto ':='] <expresión (IN) de robtarget> ',' 
[OffsetX ':='] <expresión (IN) de num> ',' 
[OffsetY ':='] <expresión (IN) de num> ',' 
[OffsetZ ':='] <expresión (IN) de num> ')'
```

Una función con un valor de retorno del tipo de dato *robtarget*.

Información relacionada

Descrito en:

Dato de posición

Tipos de dato - *robtarget*

OpMode**Lectura del modo de operación**

OpMode (Operating Mode) sirve para leer el modo de operación utilizado del sistema.

Ejemplo

```
TEST OpMode()
CASE OP_AUTO:
    ...
CASE OP_MAN_PROG:
    ...
CASE OP_MAN_TEST:
    ...
DEFAULT:
    ...
ENDTEST
```

Se ejecutarán diferentes secciones del programa, dependiendo del modo de operación utilizado.

Valor de retorno

Tipo de dato: *symnum*

El modo de operación utilizado aparece indicado de la siguiente forma según se observa en la tabla, a continuación.

Valor de retorno	Constante simbólica	Comentario
0	OP_UNDEF	Modo de operación no definido
1	OP_AUTO	Modo de operación automático
2	OP_MAN_PROG	Modo de operación manual max. 250 mm/s
3	OP_MAN_TEST	Modo de operación manual Velocidad total, 100 %

Sintaxis

OpMode'(' ')

Una función con un valor de retorno del tipo de dato *symnum*.

Información relacionada

Descripción:

Diferentes modos de operación

Puesta en Marcha - *Panel de control*

Lectura del modo de funcionamiento

Funciones - *RunMode*

OrientZYX

Cálculo de una variable de orientación a partir de ángulos Euler

OrientZYX (Orient from Euler ZYX angles) sirve para calcular una variable de tipo orientación a partir de ángulos Euler.

Ejemplo

```
VAR num angulox;
VAR num anguloy;
VAR num anguloz;
VAR pose object;
.
object.rot := OrientZYX(anguloz, anguloy, angulox)
```

Valor de retorno

Tipo de dato: *orient*

La orientación obtenida a partir de los ángulos Euler.

Las rotaciones se realizarán siguiendo el orden siguiente:

- rotación en torno al eje z,
- rotación en torno al eje y nuevo
- rotación en torno al eje x nuevo.

Argumentos

OrientZYX (ZAngulo YAngulo XAngulo)

ZAngulo

Tipo de dato: *num*

La rotación, en grados, en torno al eje Z.

YAngulo

Tipo de dato: *num*

La rotación, en grados, en torno al eje Y.

XAngulo

Tipo de dato: *num*

La rotación, en grados, en torno al eje X.

Las rotaciones se realizarán siguiendo el orden siguiente:

- rotación en torno al eje z,
- rotación en torno al eje y nuevo
- rotación en torno al eje x nuevo.

Sintaxis

```
OrientZYX'('
  [ZAngulo ':='] <expresión (IN) de num> '',
  [YAngulo ':='] <expresión (IN) de num> '',
  [XAngulo ':='] <expresión (IN) de num>
  ')'
```

Una función con un valor de retorno del tipo de dato *orient*.

Información relacionada

[Descripción](#):

Instrucciones y funciones matemáticas

Resumen RAPID - *Matemáticas*

ORobT

Eliminación de un desplazamiento de programa de una posición

ORobT (*Object Robot Target*) sirve para transformar una posición robot del sistema de coordenadas del desplazamiento de programa al sistema de coordenadas del objeto y/o eliminar un offset de los ejes externos.

Ejemplo

```
VAR robttarget p10;
VAR robttarget p11;

p10 := CRobT();
p11 := ORobT(p10);
```

Las posiciones actuales del robot y de los ejes externos están almacenadas en *p10* y *p11*. Los valores almacenados en *p10* se refieren al sistema de coordenadas ProgDisp/ExtOffs. Los valores almacenados en *p11* se refieren al sistema de coordenadas del objeto y no tienen ningún offset de los ejes externos.

Valor de retorno

Tipo de dato: *robttarget*

El dato de posición transformado.

Argumentos**ORobT (PuntOri [*\InPDisp*] | [*\InEOffs*])**

PuntOri(*Punto Original*)

Tipo de dato: *robttarget*

El punto original que deberá ser transformado.

[*\InPDisp*] (*En Desplazamiento de Programa*)

Tipo de dato: *switch*

Devuelve la posición del TCP en el sistema de coordenadas ProgDisp, es decir, que elimina únicamente el offset de los ejes externos.

[*\InEOffs*] (*En Offset Externo*)

Tipo de dato: *switch*

Devuelve los ejes externos en el sistema de coordenadas offset, es decir, que elimina únicamente el desplazamiento de programa del robot.

Ejemplos

p10 := ORobT(p10 \InEOffs);

La función ORobT eliminará todos los desplazamiento de programa activos, dejando la posición del TCP referido al sistema de coordenadas del objeto. Los ejes externos permanecerán en el sistema de coordenadas offset.

```
p10 := ORobT(p10 \InPDisp );
```

La función ORobT eliminará todos los offset de los ejes externos. La posición del TCP permanecerá en el sistema de coordenadas ProgDisp.

Sintaxis

ORobT '('
[PuntOri':='] <expresión (IN) de robtarget>
['\InPDisp] | ['\InEOffs])'

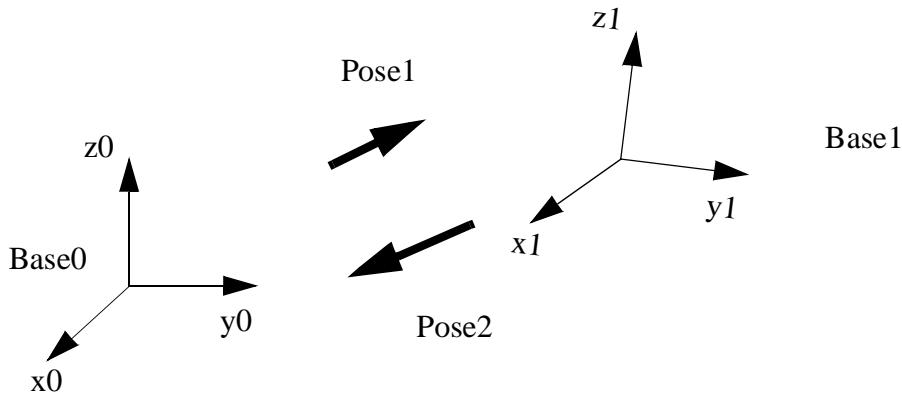
Una función con un valor de retorno del tipo de dato *robtarget*.

Información relacionada

	<u>Descripción:</u>
Definición del desplazamiento del programa del robot	Instrucciones - <i>PDispOn, PDispSet</i>
Definición de los offset de los ejes externos	Instrucciones - <i>EOffsOn, EOffsSet</i>
Sistemas de Coordenadas	Principios de Movimiento - <i>Sistemas de Coordenadas</i>

PoseInv**Inversión de la posición**

PoseInv (Pose Invert) calcula la transformación inversa de una posición.

Ejemplo

Pose1 representa las coordenadas de la Base1 respecto a la Base0.

La transformación que da como resultado las coordenadas de la Base0 respecto a la Base1 se obtiene realizando la transformación inversa:

```

VAR pose pose1;
VAR pose pose2;
.
.
pose2 := PoseInv(pose1);

```

Valor de retorno

Tipo de dato: *pose*

El valor de la posición inversa.

Argumentos**PoseInv (Pose)****Pose**

Tipo de dato: *pose*

La posición que se desea invertir.

Sintaxis

```
PoseInv'('
  [Pose ':='] <expresión (IN) de pose>
  ')'
```

Una función con un valor de retorno del tipo de dato *pose*.

Información relacionada

[Descripción](#):

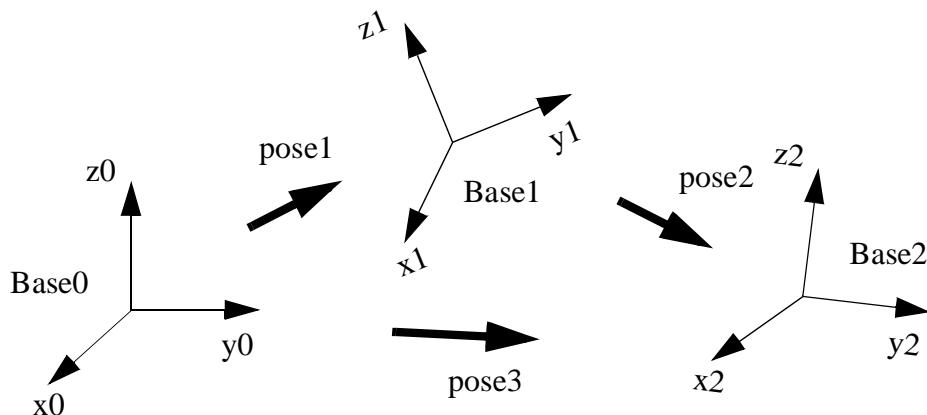
Instrucciones y funciones matemáticas

Resumen RAPID - *Matemáticas*

PoseMult Multiplicación de los datos de posición

PoseMult (Pose Multiply) sirve para calcular el producto de dos transformaciones de base. La forma más corriente consiste en calcular una estructura nueva como el resultado de un desplazamiento aplicado a una estructura de base.

Ejemplo



pose1 representa las coordenadas de la Base1 respecto a la Base0.

pose2 representa las coordenadas de la Base2 respecto a la Base1.

La transformación que produce la pose3, las coordenadas de la Base2 respecto a la Base0, se obtiene por el producto de las dos transformaciones:

```

VAR pose pose1;
VAR pose pose2;
VAR pose pose3;
.
.
.
pose3 := PoseMult(pose1, pose2);
  
```

Valor de retorno

Tipo de dato: *pose*

El valor del producto de las dos posiciones.

Argumentos

PoseMult (Pose1 Pose2)

Pose1Tipo de dato: *pose*

La primera posición.

Pose2Tipo de dato: *pose*

La segunda posición.

Sintaxis

```
PoseMult'(  
    [Pose1 ':='] <expresión (IN) de pose> ','  
    [Pose2 ':='] <expresión (IN) de pose>  
)'
```

Una función con un valor de retorno del tipo de dato *pose*.

Información relacionada

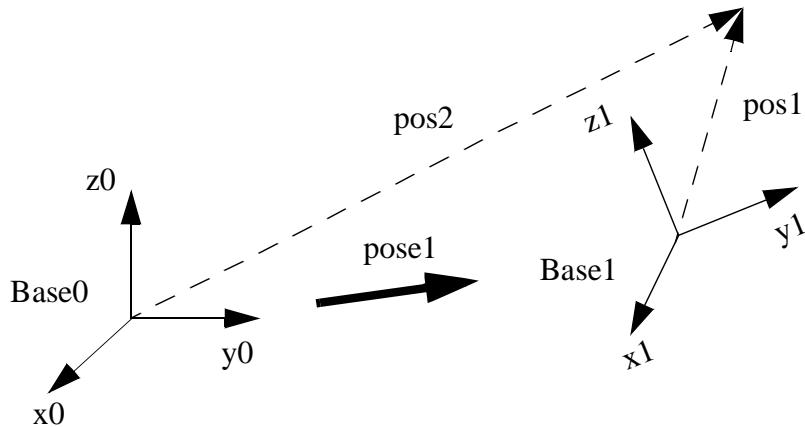
Descripción:

Instrucciones y funciones matemáticas

Resumen RAPID - *Matemáticas*

PoseVect**Aplicación de una transformación
a un vector**

PoseVect (Pose Vector) sirve para calcular el producto de una posición y de un vector. Se utiliza generalmente para calcular un vector como resultado del efecto de un desplazamiento en un vector original.

Ejemplo

pose1 representa las coordenadas de la Base1 respecto a la Base0.
pos1 es un vector respecto a la Base1.

El vector correspondiente respecto a la Base0 se obtendrá por el producto:

```

VAR pose pose1;
VAR pos pos1;
VAR pos pos2;
.
.
.
pos2:= PoseVect(pose1, pos1);

```

Valor de retorno

Tipo de dato: *pos*

El valor del producto de la pose y de la pos original.

Argumentos

PoseVect (Pose Pos)

Pose	Tipo de dato: <i>pose</i>
La transformación que se desea aplicar.	
Pos	Tipo de dato: <i>pos</i>
La pos que se desea transformar.	

Sintaxis

```
PoseVect'(  
  [Pose ':='] <expresión (IN) de pose> ','  
  [Pos ':='] <expresión (IN) de pos>  
)'
```

Una función con un valor de retorno del tipo de dato *pos*.

Información relacionada

Instrucciones y funciones matemáticas

Descripción en:

Resumen RAPID - *Matemáticas*

Pow**Cálculo de la potencia de un valor**

Pow (Power) sirve para calcular el valor exponencial en cualquier base.

Ejemplo

```
VAR num x;
VAR num y
VAR num reg1;
.
reg1:= Pow(x, y);
```

reg1 tiene asignado el valor x^y .

Valor de retorno

Tipo de dato: *num*

El valor de la base x elevado a la potencia del exponente y (x^y).

Argumentos**Pow (Base Exponente)****Base**

Tipo de dato: *num*

El valor del argumento de la base.

Exponente

Tipo de dato: *num*

El valor del argumento del exponente.

Limitaciones

La ejecución de la función x^y dará como resultado un error si:

- . $x < 0$ y además "y" no es un número entero;
- . $x = 0$ y además "y" ≤ 0 .

Sintaxis

```
Pow('
    [Base ':='] <expresión (IN) de num> ',' 
    [Exponente ':='] <expresión (IN) de num>
    ')'
```

Una función con un valor de retorno del tipo de dato *num*.

Información relacionada

InSTRUCCIONES Y FUNCIONES MATEMÁTICAS

DESCRITO EN:

RESUMEN RAPID - MATEMÁTICAS

Present

Comprobación de la utilización de un parámetro opcional

Present sirve para comprobar si se ha utilizado un argumento opcional al llamar una rutina.

Un parámetro opcional puede no ser utilizado si no ha sido especificado al llamar la rutina. Esta función puede usarse para comprobar si un parámetro ha sido especificado, a fin de evitar que se produzcan errores.

Ejemplo

```
PROC aliment (\switch on | \switch off)
```

```
    IF Present (on) Set do1;  
    IF Present (off) Reset do1;
```

```
ENDPROC
```

La salida *do1*, que controla una alimentación, será activada o desactivada dependiendo del argumento utilizado al llamar la rutina.

Valor de Retorno

Tipo de dato: *bool*

TRUE = El valor del parámetro ha sido definido al llamar la rutina.

FALSE = No ha sido definido ningún valor de parámetro.

Argumentos**Present (ParOpc)**

ParOpc (*Parámetro opcional*)

Tipo de dato: Cualquier tipo

El nombre del parámetro opcional que se desea comprobar.

Ejemplo

```

PROC cola (\switch on, num flujocola, robtarget alpunto, speeddata velocidad,
           zonedata zona, PERS tooldata herram, \PERS wobjdata wobj)

  IF Present (on) PulseDO activ_cola;
  SetAO señal_cola, flujocola;
  IF Present (wobj) THEN
    MoveL alpunto, velocidad, zona, herram\WObj=wobj;
  ELSE
    MoveL alpunto, velocidad, zona, herram;
  ENDIF

ENDPROC

```

Se ha concebido una rutina de aplicación de adhesivo. Si el argumento *on* ha sido especificado al llamar la rutina, se generará un pulso en la señal *activ_cola*. El robot activará entonces una salida analógica *señal_cola*, que controla la pistola de aplicación de adhesivo, y se moverá a la posición final. Dado que el parámetro *wobj* es opcional, se utilizarán diferentes instrucciones MoveL dependiendo de si se usa este argumento o no.

Sintaxis

Present '('
[ParOpc':='] <referencia (**REF**) de cualquier tipo> ')

Un parámetro REF requerirá en este caso, el nombre del parámetro opcional.

Una función con un valor de retorno del tipo de dato *bool*.

Información relacionada

Parámetros de rutina

Descripción en:

Características Básicas - *Rutinas*

ReadBin**Lectura a partir de un canal serie binario o archivo**

ReadBin (Read Binary) sirve para leer un byte (8 bits) a partir de un canal serie binario o archivo.

Ejemplo

```
VAR iodev canalEnt;
.
Open "sio1:", canalEnt\Bin;
carácter := ReadBin(canalEnt);
```

Un byte será leído a partir de un canal binario *canalEnt*.

Valor de retorno

Tipo de dato: *num*

Un byte (8 bits) es leído a partir de un canal serie especificado. Este byte será convertido en el valor numérico positivo correspondiente. En el caso en que el archivo esté vacío (final del archivo), el sistema devolverá el número -1.

Argumentos**ReadBin (DispositivoE/S [\\Time])**

DispositivoE/S	Tipo de dato: <i>iodev</i>
-----------------------	----------------------------

Es el nombre (referencia) del canal serie o archivo utilizado.

[\\Time]	Tipo de dato: <i>num</i>
-----------------	--------------------------

Es el tiempo máximo permitido para la operación de lectura (excedido) en segundos. En el caso en que este argumento no haya sido especificado, el tiempo máximo estará fijado a 60 segundos.

En el caso en que este tiempo haya transcurrido antes de que se haya terminado la operación de lectura, el sistema llamará al gestor de errores con el código de error **ERR_DEV_MAXTIME**. En el caso en que no haya ningún gestor de errores, la ejecución del programa será detenida.

La función de tiempo excedido (timeout) es operativa también durante el paro del programa y será observable en un programa RAPID al arranque del programa.

Ejecución del programa

La ejecución del programa espera hasta que un byte (8 bits) pueda ser leído a partir de un canal serie binario.

Ejemplo

```
Open "flp1:myfile.bin", file\Bin;  
.Rewind file;  
bindata := ReadBin(file);  
WHILE bindata <> EOF_BIN DO  
    TPWrite ByteToStr(bindata\Char);  
    bindata := ReadBin(file);  
ENDWHILE
```

Se ha realizado la lectura del contenido de un archivo binario *myfile.bin* desde el principio hasta el final del archivo y han aparecido visualizados los datos binarios en la unidad de programación, convertidos en caracteres ASCII (un carácter en cada línea).

Limitaciones

La función sólo podrá ser utilizada para los canales y archivos que hayan sido abiertos para la lectura y la escritura binarias.

Gestión de errores

En caso de ocurrir un error durante la lectura, la variable del sistema ERRNO se activará en ERR_FILEACC. Este error podrá luego ser procesado en el gestor de errores.

Datos predefinidos

La constante *EOF_BIN* podrá utilizarse para detener la lectura al final del archivo.

```
CONST num EOF_BIN := -1;
```

Sintaxis

```
ReadBin'(  
    [Dispositivo E/S ':=] <variable (VAR) de iodev>  
    ['\Time':= <expresión (IN) de num>]'
```

Un función con un valor de retorno del tipo *num*.

Información relacionada

Descripción:

Apertura (etc.) de canales serie

Resumen RAPID - *Comunicación*

Conversión de un byte en un dato de cadena

Funciones - *ByteToStr*

Datos byte

Tipos de datos - *byte*

ReadBin

Funciones

ReadMotor Lectura de los ángulos del motor actuales

ReadMotor sirve para leer los ángulos utilizados de los diferentes motores del robot y de los ejes externos. Esta función se usa fundamentalmente en el procedimiento de calibración del robot.

Ejemplo

```
VAR num motor_angle2;
motor_angle2 := ReadMotor(2);
```

El ángulo del motor utilizado del segundo eje del robot será almacenado en *motor_angle2*.

Valor de retorno

Tipo de dato: *num*

El ángulo actual del motor expresado en radianes para un eje especificado del robot o de los ejes externos.

Argumentos

ReadMotor [|MecUnit] Axis

UnitMec	(<i>Unidad Mecánica</i>)	Tipo de dato: <i>mecunit</i>
----------------	----------------------------	------------------------------

Es el nombre de la unidad mecánica para la que se debe leer un eje. En el caso en que se omita este argumento, se leerá el eje del robot.

Eje	Tipo de dato: <i>num</i>
------------	--------------------------

Es el número del eje que se debe leer (1 - 6).

Ejecución del programa

El ángulo del motor devuelto indica la posición actual, en radianes, del motor e independientemente de cualquier offset de calibración. El valor no se refiere a una posición fija del robot sino únicamente a la posición cero interna del resolver, es decir, normalmente la posición cero del resolver más próxima de la posición de calibración (la diferencia entre la posición cero del resolver y la posición de calibración es el valor offset de la calibración). El valor representa el movimiento completo de cada eje, incluso si se trata de varias vueltas.

Ejemplo

```
VAR num motor_angle3;  
motor_angle3 := ReadMotor(\MecUnit:=robot, 3);
```

El ángulo actual del motor para el tercer eje del robot será almacenado en *motor_angle3*.

Sintaxis

```
ReadMotor'(  
  [ '\'MecUnit ':= ] <variable (VAR) de mecunit> ',' ]  
  [ Axis ':= ] <expresión (IN) de num>  
)'
```

Una función con un valor de retorno del tipo de dato *num*.

Información relacionada

Lectura del ángulo actual del eje

Descripción:

Funciones - *CJointT*

ReadNum**Lectura de un número a partir de un archivo o de un canal serie**

ReadNum (Read Numeric) sirve para leer un número a partir de un archivo de caracteres o de un canal serie.

Ejemplo

```
VAR iodev enarchiv;
.
Open "flp1:archiv.doc", enarchiv\Read;
reg1 := ReadNum(enarchiv);
```

Reg1 tiene asignado un número leído en el archivo *archiv.doc* del disquete.

Valor de retorno

Tipo de dato: *num*

Es el valor numérico leído de un archivo específico. Si el archivo está vacío (final de archivo), el número 9.999E36 será devuelto.

Argumentos**ReadNum (DispositivoE/S [\Time])**

DispositivoE/S Tipo de dato: *iodev*

Es el nombre (referencia) del archivo que se desea leer.

[\Time] Tipo de dato: *num*

Es el tiempo máximo permitido para la operación de lectura (excedido) en segundos. En el caso en que no se haya especificado este argumento, el tiempo máximo estará fijado a 60 segundos.

En el caso en que este tiempo haya transcurrido antes de que la operación de lectura se haya terminado, el sistema llamará al gestor de errores con el código de error **ERR_DEV_MAXTIME**. En el caso en que no haya ningún gestor de errores, la ejecución será detenida.

La función (timeout) de tiempo excedido es operativa también durante el paro del programa y podrá observarse en el programa RAPID al arranque del programa.

Ejecución del programa

La función lee una línea de un archivo, es decir, lee todo hasta el retorno del carro (salto de línea, LF) siguiente inclusive, pero no excederá los 80 caracteres. En el caso en que la línea exceda los 80 caracteres, los caracteres restantes serán leídos a la siguiente lectura.

La cadena que ha sido leída será convertida posteriormente en un valor numérico; por ejemplo, "234,4" será convertida en el valor numérico 234,4. En el caso en que todos los caracteres no sean cifras, el sistema devolverá cero.

Ejemplo

```
reg1 := ReadNum(enarchiv);
IF reg1 > EOF_NUM THEN
    TPWrite "El archivo está vacío"
    ..

```

Antes de utilizar el número leído de un archivo, se realizará una comprobación para asegurarse de que el archivo no está vacío.

Limitaciones

La función sólo podrá utilizarse para los archivos que hayan sido abiertos para la lectura.

Gestión de errores

Si ocurre un error de acceso durante la lectura, la variable del sistema ERRNO se activará en ERR_FILEACC. Si se realiza un intento de lectura de datos no numéricos, la variable del sistema ERRNO pasará a ser ERR_RCVDATA. Estos errores podrán ser procesados posteriormente en el gestor de errores.

Datos predefinidos

La constante *EOF_NUM* podrá utilizarse para detener la lectura al final del archivo.

```
CONST num EOF_NUM := 9.998E36;
```

Sintaxis

```
ReadNum '('
    [DispositivoE/S ':='] <variable (VAR) de iodev> ')'
    ['\Time ':= <expresión (IN) de num>] )'
```

Una función con un valor de retorno del tipo *num*.

Información relacionada

Descripción:

Apertura (etc.) de los canales serie

Resumen RAPID - *Comunicación*

ReadNum

Funciones

ReadStr**Lectura de una cadena a partir de un archivo o de un canal serie**

ReadStr (Read String) sirve para leer texto a partir de un archivo de caracteres o de un canal serie.

Ejemplo

```
VAR iodev enarchiv;
.
Open "flp1:archiv.doc", enarchiv\Read;
text := ReadStr(enarchiv);
```

Text tiene asignado una cadena de texto leída del archivo *archiv.doc* en el disquete.

Valor de Retorno

Tipo de dato: *string*

La cadena de texto leída a partir del archivo especificado. Si el archivo está vacío (final del archivo), la cadena "EOF" será devuelta.

Argumentos**ReadStr (DispositivoE/S [\\Time])**

DispositivoE/S	Tipo de dato: <i>iodev</i>
-----------------------	----------------------------

Es el nombre (referencia) del archivo que se desea leer.

[\\Time]	Tipo de dato: <i>num</i>
-----------------	--------------------------

Es el tiempo máximo permitido para la operación de lectura (excedido) en segundos. En el caso en que no se haya especificado este argumento, el tiempo máximo estará fijado a 60 segundos.

En el caso en que este tiempo haya transcurrido antes de que la operación de lectura se haya terminado, el sistema llamará al gestor de errores con el código de error **ERR_DEV_MAXTIME**. En el caso en que no haya ningún gestor de errores, la ejecución será detenida.

Ejecución del programa

La función lee una línea de un archivo, es decir, lee todo hasta el retorno de carro (salto de línea, LF) siguiente inclusive, pero no excederá los 80 caracteres. En el caso en que la línea exceda los 80 caracteres, los caracteres restantes serán leídos a la siguiente lectura.

Ejemplo

```
text := ReadStr(enarchiv);
IF text = EOF THEN
    TPWrite "El archivo está vacío";
.
```

Antes de utilizar la cadena leída del archivo, se realizará una comprobación para asegurarse de que el archivo no está vacío.

Limitaciones

La función sólo podrá utilizarse para archivos que hayan sido abiertos para la lectura.

Gestión de errores

Si ocurre un error durante la lectura, la variable del sistema ERRNO se activará en ERR_FILEACC. Este error podrá ser procesado posteriormente en el gestor de errores.

Datos predefinidos

La constante *EOF* podrá utilizarse para comprobar si el archivo estaba vacío en el momento de la lectura del archivo o para detener la lectura al final del archivo.

```
CONST string EOF := "EOF";
```

Sintaxis

```
ReadStr '('
    [DispositivoE/S ':='] <variable (VAR) of iodev>')
    ['\Time ':= <expresión (IN) de num>]'
```

Una función con un valor de retorno del tipo *num*.

Información relacionada

Descripción:

Apertura (etc.) de los canales serie

Resumen RAPID - Comunicación

RelTool Ejecución de un desplazamiento relativo a la herramienta

RelTool (Relative Tool) sirve para añadir a una posición del robot un desplazamiento y/o una rotación, expresados en el sistema de coordenadas de la herramienta.

Ejemplo

MoveL RelTool (p1, 0, 0, 100), v100, fine, herra1;

El robot se mueve a una posición que se encuentra a 100 mm del punto p1 en la dirección de la herramienta.

MoveL RelTool (p1, 0, 0, 0 \Rz:= 25), v100, fine, herra1;

La herramienta será girada 25º en torno al eje z-.

Valor de retorno

Tipo de dato: *robtarget*

La nueva posición con la adición de un desplazamiento y/o de una posible rotación, relativos a la herramienta utilizada.

Argumentos

RelTool (Punto Dx Dy Dz [\Rx] [\Ry] [\Rz])

Punto Tipo de dato: *robtarget*

Es la posición de entrada del robot. La parte de orientación de esta posición define la orientación utilizada del sistema de coordenadas de la herramienta.

Dx Tipo de dato: *num*

Es el desplazamiento expresado en mm en la dirección x del sistema de coordenadas de la herramienta.

Dy Tipo de dato: *num*

Es el desplazamiento expresado en mm en la dirección y del sistema de coordenadas de la herramienta.

Dz Tipo de dato: *num*

Es el desplazamiento expresado en mm en la dirección z del sistema de coordenadas de la herramienta.

[\Rx]

Tipo de dato: *num*

Es la rotación expresada en grados en torno al eje x del sistema de coordenadas de la herramienta.

[\Ry]

Tipo de dato: *num*

Es la rotación expresada en grados en torno al eje y del sistema de coordenadas de la herramienta.

[\Rz]

Tipo de dato: *num*

Es la rotación expresada en grados en torno al eje z del sistema de coordenadas de la herramienta.

En el caso en que se hayan especificado dos o tres rotaciones al mismo tiempo, ello se llevará a cabo en primer lugar en torno al eje x, luego en torno al eje y nuevo, y luego en torno al eje z nuevo.

Sintaxis

```
RelTool'(
    [ Punto' := ] <expresión (IN) de robtarget>,
    [Dx ' := ] <expresión (IN) de num> ,
    [Dy ' := ] <expresión (IN) de num> ,
    [Dz ' := ] <expresión (IN) de num>
    [ '\Rx ' := <expresión (IN) de num> ]
    [ '\Ry ' := <expresión (IN) de num> ]
    [ '\Rz ' := <expresión (IN) de num> ]')
```

Una función con un valor de retorno del tipo de dato *robtarget*.

Información relacionadaDescripción:

[Instrucciones y funciones matemáticas](#)

[Resumen RAPID - Matemáticas](#)

[Instrucciones de posicionamiento](#)

[Resumen RAPID - Movimiento](#)

Round**Round es un valor numérico**

Round sirve para redondear un valor numérico a un número especificado de decimales o a un valor entero.

Ejemplo

VAR num val;

val := Round(0,38521\Dec:=3);

La variable *val* toma el valor de 0,385.

val := Round(0,38521\Dec:=-1);

La variable *val* toma el valor de 0,4.

val := Round(0,38521);

La variable *val* toma el valor de 0.

Valor de retorno

Tipo de dato: *num*

Es el valor numérico redondeado al número especificado de decimales.

Argumentos**Round (Val [\Dec])**

Val

(*Valor*)

Tipo de dato: *num*

Es el valor numérico que se desea redondear.

[\Dec]

(*Decimales*)

Tipo de dato: *num*

Es el número de decimales.

En el caso en que el número especificado de decimales sea 0 o si se omite el argumento, el valor será redondeado a un número entero.

El número de decimales no deberá ser negativo o mayor que la precisión disponible para el valor numérico.

Sintaxis

```
Round'('
  [ Val ':=' ] <expresión (IN) de num>
  [ \Dec ':=' <expresión (IN) de num> ]
  ')'
```

Una función con un valor de retorno del tipo de dato *num*.

Información relacionada

Instrucciones y funciones matemáticas
Forma de truncar un valor

Descripción:

Resumen RAPID - *Matemáticas*
Funciones - *Trunc*

RunMode Lectura del modo de funcionamiento

RunMode (Running Mode) sirve para leer el modo de funcionamiento utilizado del sistema.

Ejemplo

```
IF RunMode() = RUN_CONT_CYCLE THEN
    .
    .
ENDIF
```

La sección del programa será ejecutada únicamente para una ejecución continua o cíclica.

Valor de retorno

Tipo de dato: *symnum*

El modo de funcionamiento utilizado se indica según está definido en la tabla que se muestra a continuación.

Valor de retorno	Constante simbólica	Comentario
0	RUN_UNDEF	Modo de funcionamiento no definido
1	RUN_CONT_CYCLE	Modo de funcionamiento cíclico o continuo
2	RUN_INSTR_FWD	Modo de funcionamiento instrucción hacia adelante
3	RUN_INSTR_BWD	Modo de funcionamiento instrucción hacia atrás
4	RUN_SIM	Modo de funcionamiento simulado

Sintaxis

RunMode'(' ')

Una función con un valor de retorno del tipo de dato *symnum*.

Información relacionada

Descripción:

Lectura del modo de operación Funciones - *OpMode*

Sin**Cálculo del valor del seno**

Sin (Sine) sirve para calcular el valor del seno a partir de un valor de un ángulo.

Ejemplo

```
VAR num angulo;  
VAR num valor;
```

```
.
```

```
.
```

```
valor:= Sin(angulo);
```

Valor de retorno

Tipo de dato: *num*

El valor del seno, comprendido entre [-1, 1].

Argumentos**Sin (Angulo)****Angulo**

Tipo de dato: *num*

El valor del ángulo, expresado en grados.

Sintaxis

```
Sin'('  
[Angulo':='] <expresión (IN) de num>  
)'
```

Una función con un valor de retorno del tipo de dato *num*.

Información relacionada

Descripción:

Instrucciones y funciones matemáticas

Resumen RAPID - *Matemáticas*

Sin

Funciones

Sqrt

Cálculo del valor de la raíz cuadrada

Sqrt (Square root) sirve para calcular el valor de la raíz cuadrada.

Ejemplo

```
VAR num x_valor;  
VAR num y_valor;  
  
y_valor := Sqrt( x_valor);
```

Valor de retorno

Tipo de dato: *num*

El valor de la raíz cuadrada.

Argumentos

Sqrt (Valor)

Valor

Tipo de dato: *num*

El valor del argumento para la raíz cuadrada ($\sqrt{}$); tiene que ser ≥ 0 .

Sintaxis

Sqrt'('
 [Valor':='] <expresión (IN) de num>
 ')

Una función con un valor de retorno del tipo de dato *num*.

Información relacionada

Descrito en:

Instrucciones y funciones matemáticas

Resumen RAPID - *Matemáticas*

StrFind Búsqueda de un carácter en una cadena

StrFind (String Find) sirve para realizar la búsqueda en una cadena de texto, empezando en una posición específica, de un carácter que constituye un elemento de un grupo específico de caracteres.

Ejemplo

VAR num found;

found := StrFind("Robotics",1,"aeiou");

La variable *found* toma el valor 2.

found := StrFind("Robotics",1,"aeiou"\NotInSet);

La variable *found* toma el valor 1.

found := StrFind("IRB 6400",1,STR_DIGIT);

La variable *found* toma el valor 5.

found := StrFind("IRB 6400",1,STR_WHITE);

La variable *found* toma el valor 4.

Valor de retorno

Tipo de dato: *num*

La posición de carácter del primer carácter en o después de la posición especificada, que sea un elemento del grupo especificado. En el caso en que no hubiera dicho carácter, se devolverá la longitud de la cadena +1.

Argumentos

StrFind (Str ChPos Set [\NotInSet])

Str	<i>(Cadena)</i>	Tipo de dato: <i>string</i>
------------	-----------------	-----------------------------

La cadena en la que hay que realizar la búsqueda.

ChPos	<i>(Posición del Carácter)</i>	Tipo de dato: <i>num</i>
--------------	--------------------------------	--------------------------

Es el inicio de la búsqueda de la posición del carácter. Se generará un error de ejecución si la posición se encuentra fuera de la cadena.

SetTipo de dato: *string*

Es el grupo de caracteres que tiene ser analizado.

[NotInSet]Tipo de dato: *switch*

Indica la búsqueda de un carácter que no se encuentra en el grupo de caracteres.

Sintaxis

```
StrFind'(
    [ Str '==' ] <expresión (IN) de string> ,
    [ ChPos '==' ] <expresión (IN) de num> ,
    [ Set'==' ] <expresión (IN) de string>
    [ '\' NotInSet ]
)'
```

Una función con un valor de retorno del tipo de dato *num*.

Información relacionada

Funciones de las cadenas

Descripción:

Resumen RAPID - *Funciones de cadenas*

Definición de una cadena

Tipos de Datos - *string*

Valores de las cadenas

Características Básicas -
Elementos Básicos

StrLen

Obtención de la longitud de la cadena

StrLen (String Length) sirve para encontrar la longitud actual de una cadena.

Ejemplo

```
VAR num len;  
len := StrLen("Robotics");
```

La variable *len* toma el valor de 8.

Valor de retorno

Tipo de dato: *num*

Es el número de caracteres dentro de la cadena (≥ 0).

Argumentos

StrLen (Str)

Str	(Cadena)	Tipo de dato: <i>string</i>
-----	----------	-----------------------------

Es la cadena en la que hay que contar el número de caracteres.

Sintaxis

```
StrLen'(  
[ Str ':=' ] <expresión (IN) de string>  
)'
```

Una función con un valor de retorno del tipo de dato *num*.

Información relacionada

Funciones de las cadenas

Definición de una cadena

Valores de las cadenas

Descripción:

Resumen RAPID - *Funciones de cadena*

Tipos de Datos - *string*

Características Básicas -
Elementos Básicos

StrMap

Mapa de una cadena

StrMap (String Mapping) sirve para crear una copia de una cadena en la que todos los caracteres son trasladados de acuerdo con una disposición específica de mapa.

Ejemplo

```
VAR string str;  
str := StrMap("Robotics","aeiou","AEIOU");
```

La variable *str* toma el valor "RObOtIcs".

```
str := StrMap("Robotics",STR_LOWER,STR_UPPER);
```

La variable *str* toma el valor "ROBOTICS".

Valor de retorno

Tipo de dato: *string*

La cadena es creada por translación de los caracteres de una cadena específica según lo especifican las cadenas "desde (from)" y "a (to)". Cada carácter de la cadena especificada, que se encuentre en la cadena "desde (from)" será sustituido por el carácter en la posición correspondiente en la cadena "a (to)". Los caracteres para los que no se ha definido ningún mapa serán copiados de forma idéntica en la cadena resultante.

Argumentos

StrMap (Str FromMap ToMap)

Str	(Cadena)	Tipo de dato: <i>string</i>
------------	----------	-----------------------------

La cadena que se desea trasladar.

FromMap	Tipo de dato: <i>string</i>
----------------	-----------------------------

Componente de índice de la cadena.

ToMap	Tipo de dato: <i>string</i>
--------------	-----------------------------

Componente de valor de la cadena.

Sintaxis

```
StrMap'('
  [ Str' := ] <expresión (IN) de string> ',' 
  [ FromMap' := ] <expresión (IN) de string> ',' 
  [ ToMap' := ] <expresión (IN) de string>
  ')'
```

Una función con un valor de retorno del tipo de dato *string*.

Información relacionada

Descripción en:

Funciones de las cadenas

Resumen RAPID - *Funciones de cadena*

Definición de una cadena

Tipos de Datos - *string*

Valores de las cadenas

Características Básicas -
Elementos Básicos

StrMatch**Búsqueda de una estructura
en una cadena**

StrMatch (*String Match*) sirve para buscar una estructura específica dentro de una cadena, empezando desde una posición específica.

Ejemplo

```
VAR num found;
found := StrMatch("Robotics",1,"bo");
La variable found toma el valor de 3.
```

Valor de retornoTipo de dato: *num*

La posición del carácter de la primera subcadena, en la posición especificada o después de ella, que corresponda con la estructura de cadena especificada. En el caso en que no se encuentre ninguna subcadena, el sistema devolverá la longitud de cadena +1.

Argumentos**StrMatch (Str ChPos Pattern)**

Str	<i>(Cadena)</i>	Tipo de dato: <i>string</i>
------------	-----------------	-----------------------------

La cadena en la que hay que buscar.

ChPos	<i>(Posición del carácter)</i>	Tipo de dato: <i>num</i>
--------------	--------------------------------	--------------------------

Iniciar la búsqueda de la posición del carácter. Se producirá un error de ejecución si la posición se encuentra fuera de la cadena.

Pattern		Tipo de dato: <i>string</i>
----------------	--	-----------------------------

Es la estructura de cadena que hay que buscar.

Sintaxis

```
StrMatch'(
  [ Str '==' ] <expresión (IN) de string> ',' 
  [ ChPos '==' ] <expresión (IN) de num> ',' 
  [ Pattern '==' ] <expresión (IN) de string>
)'
```

Una función con un valor de retorno del tipo de dato *num*.

Información relacionada

Funciones de las cadenas

Descripción en:

Resumen RAPID - *Funciones de cadena*

Definición de una cadena

Tipos de Datos - *string*

Valores de las cadenas

Características Básicas -
Elementos Básicos

StrMembComprobar si un carácter pertenece a un conjunto

StrMemb (String Member) sirve para comprobar si un carácter específico dentro de una cadena pertenece a un conjunto específico de caracteres.

Ejemplo

```
VAR bool memb;
```

```
memb := StrMemb("Robotics",2,"aeiou");
```

La variable *memb* toma el valor TRUE, ya que 'o' es un miembro del conjunto "aeiou".

```
memb := StrMemb("Robotics",3,"aeiou");
```

La variable *memb* toma el valor FALSE, ya que 'b' no es un miembro del conjunto "aeiou".

```
memb := StrMemb("S-721 68 VÄSTERÅS",3,STR_DIGIT);
```

La variable *memb* toma el valor TRUE.

Valor de retorno

Tipo de dato: *bool*

TRUE en el caso en que el carácter en la posición especificada de la cadena específica pertenece al conjunto específico de caracteres.

Argumentos

StrMemb (Str ChPos Set)

Str	<i>(Cadena)</i>	Tipo de dato: <i>string</i>
------------	-----------------	-----------------------------

La cadena en la cual se debe realizar la comprobación.

ChPos	<i>(Posición del carácter)</i>	Tipo de dato: <i>num</i>
--------------	--------------------------------	--------------------------

La posición del carácter que debe ser comprobada. Se generará un error de ejecución si la posición se encuentra fuera de la cadena.

Set	Tipo de dato: <i>string</i>
------------	-----------------------------

Conjunto de caracteres que deben ser comprobados.

Sintaxis

```
StrMemb'('
  [ Str ':=' ] <expresión (IN) de string> ',' 
  [ ChPos ':=' ] <expresión (IN) de num> ',' 
  [ Set':=' ] <expresión (IN) de string>
  ')'
```

Una función con un valor de retorno del tipo de dato *bool*.

Información relacionada

Funciones de las cadenas

Descripción en:

Resumen RAPID - *Funciones de cadena*

Definición de una cadena

Tipos de datos - *string*

Valores de las cadenas

Características Básicas -
Elementos Básicos

StrOrder Comprobar si las cadenas están ordenadas

StrOrder (String Order) sirve para comprobar el orden de dos cadenas de acuerdo con una secuencia de orden de caracteres específica.

Ejemplo

```
VAR bool le;  
  
le := StrOrder("FIRST","SECOND",STR_UPPER);
```

La variable *le* toma el valor TRUE, porque "FIRST" viene antes de "SECOND" en la secuencia de orden de caracteres STR_UPPER.

Valor de retorno

Tipo de dato: *bool*

TRUE en el caso en que la primera cadena venga antes de la segunda cadena (`Str1 <= Str2`) cuando los caracteres están ordenados según se ha especificado.

Se considerará que los caracteres que no están incluidos en el orden definido, seguirán los que están presentes.

Argumentos

StrOrder (Str1 Str2 Order)

Str1	(Cadena 1)	Tipo de dato: <i>string</i>
Es el primer valor de cadena.		
Str2	(Cadena 2)	Tipo de dato: <i>string</i>
Es el segundo valor de cadena.		
Order		Tipo de dato: <i>string</i>
Es la secuencia de caracteres que define el orden.		

Sintaxis

```
StrOrder'(
    [ Str1 ':=' ] <expresión (IN) de string > '',
    [ Str2 ':=' ] <expresión (IN) de string > '',
    [ Order ':=' ] <expresión (IN) de string >
)'
```

Una función con un valor de retorno del tipo de dato *bool*.

Información relacionada

Funciones de las cadenas

Descripción en:

Resumen RAPID - *Funciones de cadena*

Definición de una cadena

Tipos de Datos - *string*

Valores de las cadenas

Características Básicas -
Elementos Básicos

StrPart Obtención de una parte de una cadena

StrPart (String Part) sirve para tomar una parte de una cadena y considerarla como una cadena nueva.

Ejemplo

```
VAR string part;
```

```
part := StrPart("Robotics",1,5);
```

La variable *part* toma el valor "Robot".

Valor de retorno

Tipo de dato: *string*

Es la subcadena de la cadena especificada, que tiene la longitud específica y que empieza en la posición de carácter especificada.

Argumentos

StrPart (Str ChPos Len)

Str	(<i>Cadena</i>)	Tipo de dato: <i>string</i>
------------	-------------------	-----------------------------

La cadena de la que se tomará una parte.

ChPos	(<i>Posición del carácter</i>)	Tipo de dato: <i>num</i>
--------------	----------------------------------	--------------------------

Empieza la búsqueda de la posición del carácter. Se producirá un error de ejecución si la posición se encuentra fuera de la cadena.

Len	(<i>Longitud</i>)	Tipo de dato: <i>num</i>
------------	---------------------	--------------------------

Es la longitud de la porción de la cadena. Se producirá un error de ejecución cuando la longitud sea negativa o mayor que la longitud de la cadena, o si la subcadena se encuentra (parcialmente) fuera de la cadena.

Sintaxis

```
StrPart'(
    [ Str ':=> ] <expresión (IN) de string> ',' 
    [ ChPos ':=> ] <expresión (IN) de num> ',' 
    [ Len ':=> ] <expresión (IN) de num>
    ')'
```

Una función con un valor de retorno del tipo de dato *string*.

Información relacionada

Funciones de las cadenas

Descripción en:

Resumen RAPID - *Funciones de cadena*

Definición de una cadena

Tipos de Datos - *string*

Valores de las cadenas

Características Básicas -
Elementos Básicos

StrToByte**Conversión de una cadena
en un dato byte**

StrToByte (String To Byte) sirve para convertir una *cadena* con un formato de dato byte definido en un dato *byte* ...

Ejemplo

```
VAR string con_data_buffer{5} := ["10", "AE", "176", "00001010", "A"];  
VAR byte data_buffer{5};
```

```
data_buffer{1} := StrToByte(con_data_buffer{1});
```

El contenido de la componente de matriz *data_buffer{1}* será de 10 decimales después de la función *StrToByte* ...

```
data_buffer{2} := StrToByte(con_data_buffer{2}\Hex);
```

El contenido de la componente de matriz *data_buffer{2}* será de 174 decimales después de la función *StrToByte* ...

```
data_buffer{3} := StrToByte(con_data_buffer{3}\Okt);
```

El contenido de la componente de matriz *data_buffer{3}* será de 126 decimales después de la función *StrToByte* ...

```
data_buffer{4} := StrToByte(con_data_buffer{4}\Bin);
```

El contenido de la componente de matriz *data_buffer{4}* será de 10 decimales después de la función *StrToByte* ...

```
data_buffer{5} := StrToByte(con_data_buffer{5}\Char);
```

El contenido de la componente de matriz *data_buffer{5}* será de 65 decimales después de la función *StrToByte* ...

Valor de retorno

Tipo de dato: *byte*

El resultado de la operación de conversión representado en decimales.

Arguments

StrToByte (ConStr [\Hex] | [\Okt] | [\Bin] | [\Char])

ConStr *(Convert String)* Tipo de dato: *string*

El dato de cadena que se desea convertir.

En el caso en que el argumento switch opcional sea omitido, la cadena que debe ser convertida tiene un formato *decimal* (Dec).

[\Hex] *(Hexadecimal)* Tipo de dato: *switch*

La cadena que debe ser convertida tiene un formato *hexadecimal*.

[\Okt] *(Octal)* Tipo de dato: *switch*

La cadena que debe ser convertida tiene un formato *octal*.

[\Bin] *(Binario)* Tipo de dato: *switch*

La cadena que debe ser convertida tiene un formato *binario*.

[\Char] *(Caracter)* Tipo de dato: *switch*

La cadena que debe ser convertida tiene un formato de carácter *ASCII*.

Limitaciones

Según el formato de la cadena que debe ser convertida, los siguientes datos de cadena serán válidos:

Formato:	Long.cadena:	Alcance:
Dec: '0' - '9'	3	"0" - "255"
Hex: '0' - '9', 'a' - 'f', 'A' - 'F'	2	"0" - "FF"
Okt: '0' - '7'	3	"0" - "377"
Bin: '0' - '1'	8	"0" - "11111111"
Char: Cualquier carácter ASCII	1	Tabla ASCII

Los códigos de carácter en RAPID (por ejemplo "\07" para el carácter de control BEL) no podrán ser utilizados como argumentos en ConStr.

Sintaxis

```
StrToByte'(
    [ConStr ':='] <expresión (IN) de string>
    ['\' Hex ] | ['\' Okt] | ['\' Bin] | ['\' Char]
    ')';'
```

Una función con un valor de retorno del tipo de dato *byte*.

Información relacionada

Descripción:

Conversión de un byte en un dato de cadena

Instrucciones - *ByteToStr*

Otras funciones bit (byte)

Resumen RAPID - *Funciones Bit*

Otras funciones de cadena

Resumen RAPID - *Funciones Cadena*

StrToVal**Conversión de una cadena
en un valor numérico**

StrToVal (String To Value) sirve para convertir una cadena en un valor de cualquier tipo de dato.

Ejemplo

```
VAR bool ok;
VAR num nval;

ok := StrToVal("3.85",nval);
```

La variable *ok* toma el valor TRUE y *nval* toma el valor de 3,85.

Valor de retorno

Tipo de dato: *bool*

TRUE en el caso en que la conversión requerida haya sido satisfactoria, de lo contrario, será FALSE.

Argumentos**StrToVal (Str Val)**

Str	(Cadena)	Tipo de dato: <i>string</i>
------------	----------	-----------------------------

Un valor de cadena que contiene datos literales con un formato que corresponde al tipo de dato utilizado en el argumento *Val*. El formato es válido como para los agregados literales en RAPID.

Val	(Valor)	Tipo de dato: <i>CUALQUIER TIPO</i>
------------	---------	-------------------------------------

Es el nombre de la variable o del dato persistente de cualquier tipo para el almacenamiento del resultado de la conversión. El dato permanecerá el mismo en el caso en que la conversión requerida haya fallado.

Ejemplo

```
VAR string str15 := "[600, 500, 225.3]";
VAR bool ok;
VAR pos pos15;

ok := StrToVal(str15,pos15);
```

La variable *ok* adoptará el valor TRUE y la variable *p15* adoptará el valor que está especificado en la cadena *str15*.

Sintaxis

```
StrToVal'('
  [ Str ':='] <expresión (IN) de string> ',' 
  [ Val ':='] <var o pers (INOUT) de CUALQUIER TIPO>
  ')'
```

Una función con un valor de retorno del tipo de dato *bool*.

Información relacionada

Descripción en:

Funciones de las cadenas

Resumen RAPID - *Funciones de cadena*

Definición de una cadena

Tipos de Datos - *string*

Valores de las cadenas

Características Básicas -
Elementos Básicos

Tan**Cálculo del valor de la tangente**

Tan (Tangente) sirve para calcular el valor de la tangente a partir de un valor de ángulo.

Ejemplo

```
VAR num angulo;  
VAR num valor;
```

```
.
```

```
.
```

```
valor := Tan(angulo);
```

Valor de retorno

Tipo de dato: *num*

El valor de la tangente.

Argumentos

Tan (Angulo)

Angulo

Tipo de dato: *num*

El valor del ángulo, expresado en grados.

Sintaxis

```
Tan'('  
[Angulo ':='] <expresión (IN) de num>  
'')
```

Una función con un valor de retorno del tipo de dato *num*.

Información relacionada

Descripción:

Instrucciones y funciones matemáticas
Tangente del arco con valor de retorno
comprendido entre [-180, 180]

Resumen RAPID - *Matemáticas*

Funciones - *ATan2*

Tan

Funciones

TestDI

Comprobación de la activación de una entrada digital

TestDI sirve para comprobar si una entrada digital está activada.

Ejemplos

IF TestDI (di2) THEN . . .

Si el valor actual de la señal *di2* es igual a 1, entonces . . .

IF NOT TestDI (di2) THEN . . .

Si el valor actual de la señal *di2* es igual a 0, entonces . . .

WaitUntil TestDI(di1) AND TestDI(di2);

La ejecución del programa continua únicamente después de que las entradas *di1* y *di2* hayan sido activadas.

Valor de Retorno

Tipo de dato: *bool*

TRUE = El valor actual de la señal es igual a 1.

FALSE = El valor actual de la señal es igual a 0.

Argumentos

TestDI (Señal)

Señal

Tipo de dato: *signaldi*

El nombre de la señal que se desea comprobar.

Sintaxis

TestDI '('
[Señal ':='] < variable (**VAR**) of *signaldi* > ')'

Una función con un valor de retorno del tipo de dato *bool*.

Información relacionada

Lectura del valor de una señal
de entrada digital

Instrucciones de Entrada/Salida

Descristo en:

Funciones - *DInput*

Resumen RAPID -
Señales de Entrada y Salida

Trunc**Truncar un valor numérico**

Trunc (*Truncate*) sirve para truncar un valor numérico a un número especificado de decimales o a un número entero.

Ejemplo

```
VAR num val;
```

```
val := Trunc(0,38521\Dec:=3);
```

La variable *val* toma el valor de 0,385.

```
reg1 := 0,38521
```

```
val := Trunc(reg1\Dec:=1);
```

La variable *val* toma el valor de 0,3.

```
val := Trunc(0,38521);
```

La variable *val* toma el valor de 0.

Valor de retorno

Tipo de dato: *num*

El valor numérico truncado al número especificado de decimales.

Argumentos**Trunc (Val [\Dec])**

Val

(*Valor*)

Tipo de dato: *num*

El valor numérico que debe ser truncado.

\Dec]

(*Decimales*)

Tipo de dato: *num*

Es el número de decimales.

En el caso en que el número especificado de decimales sea 0 o si se omite el argumento, el valor será truncado a un valor entero.

El número de decimales no deberá ser negativo ni mayor que la precisión disponible para los valores numéricos.

Sintaxis

```
Trunc'(  
  [ Val ':=' ] <expresión (IN) de num>  
  [ \Dec ':=' <expresión (IN) de num> ]  
)'
```

Una función con un valor de retorno del tipo de dato *num*.

Información relacionada

Instrucciones y funciones matemáticas
Redondear un valor

Descripción:

Resumen RAPID - *Matemáticas*
Funciones - *Round*

ValToStr Conversión de un valor en una cadena

ValToStr (Value To String) sirve para convertir un valor de cualquier tipo de dato en una cadena.

Ejemplo

```
VAR string str;
VAR pos p := [100,200,300];
str := ValToStr(1,234567);
```

La variable *str* toma el valor de "1,234567".

```
str := ValToStr(TRUE);
```

La variable *str* toma el valor de "TRUE".

```
str := ValToStr(p);
```

La variable *str* toma el valor de "[100,200,300]".

Valor de retorno

Tipo de dato: *string*

El valor es convertido en una cadena con el formato RAPID estándar. Ello significa en principio 6 cifras significantes. Si la parte decimal es menor que 0,000005 o mayor que 0,999995, el número será redondeado a un número entero.

Se producirá un error de ejecución si la cadena resultante es demasiado larga.

Argumentos

ValToStr (Val)

Val

(*Valor*)

Tipo de dato: ANYTYPE

Un valor de cualquier tipo de dato.

Sintaxis

```
ValToStr'(
  [ Val ':=' ] <expresión (IN) de ANYTYPE>
)'
```

Una función con un valor de retorno del tipo de dato *string*.

Información relacionada

Funciones de las cadenas

Definición de una cadena

Valores de las cadenas

Descripción:

Resumen RAPID - *Funciones de cadena*

Tipos de Datos - *string*

Características Básicas -
Elementos Básicos

Datos y Programas Predefinidos

INDICE

	Página
1 Módulo User del Sistema	3
1.1 Contenido	3
1.2 Creación de nuevos datos en este módulo.....	3
1.3 Eliminación de estos datos	4

Datos y Programas Predefinidos

1 Módulo User del Sistema

Con vistas a facilitar la programación, el sistema dispone de una serie de datos predefinidos que son suministrados con el robot. Estos datos no deben ser creados y por lo tanto, podrán usarse directamente.

Utilizando estos datos, la programación inicial será más fácil. Sin embargo, se recomienda al usuario que atribuya sus propios nombres a los datos que desea utilizar a fin de que el programa sea más fácil de leer.

1.1 Contenido

El módulo *User* comprende cinco datos numéricos (registros), un dato del objeto de trabajo, un reloj y dos valores simbólicos para las señales digitales.

<u>Nombre</u>	<u>Tipo de dato</u>	<u>Declaración</u>
reg1	num	VAR num reg1:=0
reg2	.	.
reg3	.	.
reg4	.	.
reg5	num	VAR num reg5:=0
wobj1	wobjdata	PERS wobjdata wobj1:=wobj0
clock1	clock	VAR clock clock1
high	dionum	CONST dionum high:=1
low	dionum	CONST dionum low:=0

User es un módulo del sistema, lo cual significa que siempre está presente en la memoria de robot, independientemente del programa que está cargado.

1.2 Creación de nuevos datos en este módulo

Este módulo puede ser utilizado para crear aquellos datos y rutinas que deben permanecer siempre presentes en la memoria del programa, independientemente de qué programa ha sido cargado, por ejemplo, las herramientas y rutinas de servicio.

- Seleccionar la función **Ver: Módulos** de la ventana de Programa.
- Seleccionar el módulo del sistema *User* y apretar la tecla Retorno .
- Cambiar, crear datos y rutinas siguiendo el procedimiento normal (referirse al apartado de *Programación y Prueba*).

1.3 Eliminación de estos datos

Atención: Si se borra el módulo, la instrucción CallByVar no funcionará.

Para eliminar todos los datos (es decir, el módulo entero):

- Seleccionar la función **Ver: Módulos** de la ventana de Programa.
- Seleccionar el módulo *User*.
- Pulsar la tecla Borrar .

Para cambiar o eliminar datos individuales:

- Seleccionar la función **Ver: Datos** de la ventana de Programa.
- Seleccionar **Datos: En todos los módulos**.
- Seleccionar los datos deseados. Si no aparecen indicados, pulsar la tecla de función **Tipos** para seleccionar el tipo de datos correcto.
- Cambiar o eliminar siguiendo el procedimiento normal (referirse al apartado de *Programación y Prueba*).

INDICE

	Página
1 Programación Off-line	3
1.1 Formato de los archivos	3
1.2 Edición	3
1.3 Comprobación de la sintaxis.....	3
1.4 Ejemplos	4
1.5 Creación de instrucciones propias del usuario.....	5

Programación Off-line

Programación Off-line

1 Programación Off-line

Los programas en RAPID podrán ser fácilmente creados, mantenidos y almacenados en un ordenador normal. Toda la información podrá ser leída y cambiada directamente mediante un editor de texto normal. Este capítulo explica el procedimiento que se debe seguir para conseguirlo. Además de la programación off-line, se podrá utilizar la herramienta de computador QuickTeach.

1.1 Formato de los archivos

El robot almacena y lee los programas RAPID en el formato TXT (ASCII) y es capaz de manipular tanto formatos DOS como UNIX. Si el usuario utiliza un procesador de textos para editar los programas, éstos deberán almacenarse en el formato TXT (ASCII) antes de poder ser utilizados en el robot.

1.2 Edición

Cuando el programa es creado o modificado en un procesador de textos, toda la información será manipulada bajo la forma de texto. Esto significa que la información referente a datos y rutinas diferirá un poco respecto a lo que aparece visualizado en la unidad de programación.

Téngase en cuenta que el valor de una posición almacenada aparecerá visualizada como un * en la unidad de programación, mientras que el archivo de texto contendrá el valor de la posición utilizado (x, y, z, etc.).

En vistas a minimizar el riesgo de errores en la sintaxis (programas defectuosos), es aconsejable que el usuario utilice una plantilla o modelo. Dicho modelo puede tener la forma de un programa creado previamente en el robot o mediante QuickTeach. Estos programas pueden ser leídos directamente por un procesador de textos sin tener que ser convertidos.

1.3 Comprobación de la sintaxis

Los programas deberán ser sintácticamente correctos antes de ser cargados en el robot. Esto significa que el texto debe seguir las normas fundamentales del lenguaje RAPID. Se deberá utilizar uno de los siguientes métodos para detectar los errores del texto:

- Guardar el archivo en un disquete e intentar abrirlo en el robot. En el caso en que hayan errores de sintaxis, el programa no será aceptado y aparecerá un mensaje de error visualizado. Para obtener información sobre el tipo de error, el robot almacena una lista llamada PGM CPL1.LOG en el disco RAM interno. Copiar esta lista en un

Programación Off-line

disquete utilizando el Administrador de Archivos del robot. Abrir la lista en un procesador de textos para poder leer las líneas que son incorrectas y recibir una descripción del error.

- Abrir el archivo con QuickTeach o con ProgramMaker.
- Utilizar un programa de comprobación de sintaxis RAPID para PC.

Cuando el programa es sintácticamente correcto, podrá ser comprobado y editado en el robot. Para asegurarse de que todas las referencias a las rutinas y datos son correctos, se deberá utilizar el comando **Archivo: Comprobar Programa**. En el caso en que el programa haya sido modificado en el robot, podrá ser almacenado en un disquete y procesado o guardado en un PC.

1.4 Ejemplos

A continuación se muestran varios ejemplos de la apariencia de las rutinas en formato de texto:

```
% % %
VERSION: 1
LANGUAGE: ENGLISH
% % %
MODULE main
VAR intnum process_int ;
! Demo of RAPID program
PROC main()
    MoveL p1, v200, fine, gun1;
ENDPROC

TRAP InvertDo12
! Trap routine for TriggInt
TEST INTNO
CASE process_int:
InvertDO do12;
DEFAULT:
TPWrite "Unknown trap , number="\Num:=INTNO;
ENDTEST
ENDTRAP

LOCAL FUNC num MaxNum(num t1, num t2)
IF t1 > t2 THEN
    RETURN t1;
ELSE
    RETURN t2;
ENDIF
ENDFUNC
ENDMODULE
```

1.5 Creación de instrucciones propias del usuario

Con vistas a facilitar la programación, el usuario podrá personalizar sus propias instrucciones. Estas serán creadas bajo la forma de rutinas normales, pero cuando se está programando o realizando pruebas de funcionamiento, bajo la forma de instrucciones:

- Pueden sacarse de la Lista de Selección de Instrucciones y ser programadas como instrucciones normales.
- La rutina completa será ejecutada durante la ejecución paso a paso.
- Crear un módulo nuevo del sistema donde se podrán colocar las rutinas del usuario que funcionan como instrucciones. Alternativamente, el usuario podrá colocarlos en el módulo USER del sistema.
- Crear una rutina en este módulo del sistema con el nombre que se desea atribuir a la nueva instrucción. Los argumentos de la instrucción se definen bajo la forma de parámetros de rutina. Observar que el nombre de los parámetros aparecerán visualizados en la ventana durante la programación y por ello se deberán atribuir nombres que el usuario pueda reconocer.
- Colocar la rutina en una de las listas de selección Más Comunes.
- En el caso en que la instrucción deba comportarse de una manera especial durante la ejecución del programa hacia atrás, esto podrá llevarse a cabo mediante la forma de un gestor de ejecución hacia atrás. En el caso en que no haya este tipo de gestor, será imposible poder ir más allá de la instrucción durante la ejecución hacia atrás del programa (véase el Capítulo 13 de este manual - Características Básicas). Se podrá introducir un gestor de ejecución hacia atrás utilizando el comando **Rutina: Añadir Gestor de ejecución hacia atrás** en la ventana *Rutinas del Programa*.
- Llevar a cabo un test completo de la rutina de forma a que funcione con diferentes tipos de datos de entrada (argumentos).
- Cambiar el atributo del módulo a NOSTEPIN. La rutina completa será entonces ejecutada durante la ejecución paso a paso. Este atributo no obstante, deberá ser introducido en off-line.

Ejemplo: Para facilitar la manipulación de la pinza, se crearán dos instrucciones nuevas, *GripOpen* y *GripClose*. El nombre de la señal de salida es dada al argumento de instrucción, por ejemplo, *GripOpen gripper1*.

```
MODULE My_instr (SYSMODULE, NOSTEPIN)
PROC GripOpen (VAR signaldo Gripper)
    Set Gripper;
    WaitTime 0.2;
ENDPROC
PROC GripClose (VAR signaldo Gripper)
    Reset Gripper;
    WaitTime 0.2;
ENDPROC
ENDMODULE
```

Programación Off-line

INDICE

seamdata	Datos iniciales y finales de soldadura
weavedata	Datos de oscilación
welddata	Datos de soldadura
ArcC	Soldadura al arco con movimiento circular
arcdata	Datos de proceso de soldadura al arco
ArcKill	Aborto del proceso de soldadura al arco
ArcL	Soldadura al arco con movimiento lineal
ArcRefresh	Restablecer datos de soldadura al arco

seamdata Datos iniciales y finales de soldadura

Seamdata sirve para controlar el inicio y el final de la soldadura. *Seamdata* se utiliza también cuando se rearanca un proceso después de que una operación de soldadura haya sido interrumpida.

La fase de soldadura actual está controlada mediante los datos de soldadura *welddata*.

Descripción

Los datos iniciales y finales de soldadura describen datos cuyos valores, de forma general, pueden permanecer inalterados durante el ciclo completo de soldadura y también al soldar diferentes cordones. Estos datos se utilizan cuando se prepara la operación de soldadura, al encender el arco, al calentar después del encendido y también al finalizar la soldadura.

Los datos iniciales y finales de soldadura están incluidos en todas las instrucciones de soldadura al arco para facilitar el control de las fases iniciales y finales independientemente del lugar en que se producen las interrupciones o los rearanques.

Nota Algunos componentes de los datos iniciales y finales de soldadura dependerán de la configuración del robot. Si se omite una característica dada, el componente correspondiente se quedará fuera de los datos iniciales y finales de soldadura. Las condiciones que deben cumplirse para que existan los componentes, se encuentran descritas en los parámetros del sistema.

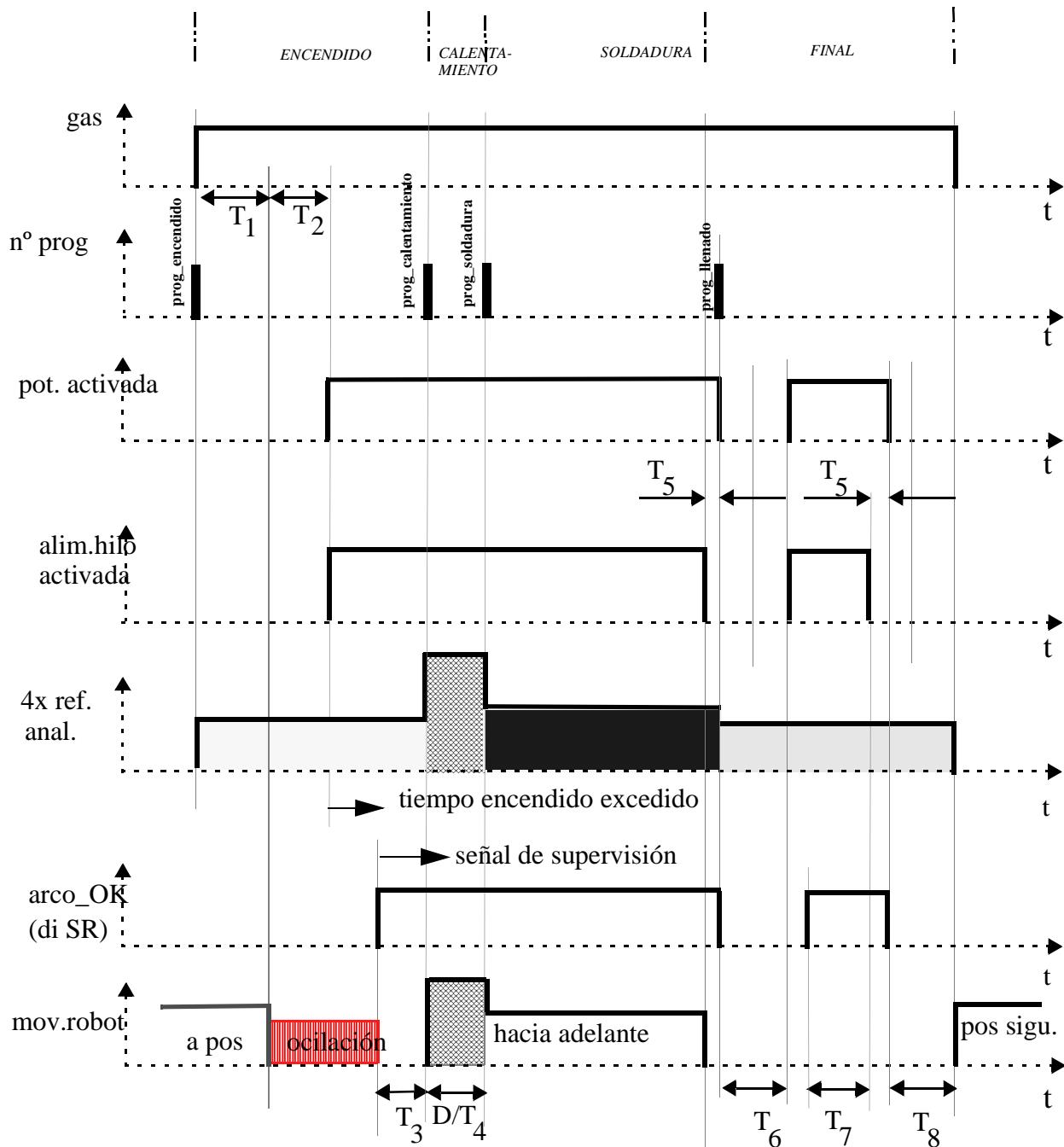
Todas las tensiones pueden expresarse de dos maneras diferentes (determinadas por el equipo de soldadura):

- Como valores absolutos (en este caso sólo se utilizarán los valores positivos).
- Como correcciones de los valores determinados en el equipo de proceso (en este caso, se utilizarán los valores tanto positivos como negativos).

El paso del electrodo de soldadura en este apartado se refiere a la soldadura MIG/MAG. En el caso de la soldadura TIG:

- La señal de hilo frío se obtiene de velocidad de hilo.
- El valor de referencia necesario de la *corriente de soldadura* podrá ser conectado a cualquiera de las tres salidas analógicas que no se utilizan. (La referencia de *tensión de soldadura* no será utilizada).

La secuencia de soldadura



T1: purga_max. gas / tiempo predef_soldadura
 T2: tiempo preflujo_gas
 T3: movimiento_encendido_retraso tiempo
 D/T4: distancia/tiempo calentamiento

T5: tiempo de postquemado
 T6: enfriamiento máx./tiempo predef_sold
 T7: tiempo de llenado
 T8: enfriamiento máx./tiempo postflujo_ga

Componentes

Grupo de componente: Encendido

tiempo_purgaTipo de dato: *num*

Se trata del tiempo (en segundos) utilizado para llenar los conductos de gas y la pinza de soldadura, con gas de protección denominado “gas de purga”.

En el caso en que la primera instrucción de soldadura contenga el argumento *\On* (arranque sobre la marcha), el flujo de gas será activado en el tiempo de purga especificado antes de que se alcance la posición programada.

Si el tiempo de posicionamiento a la posición de arranque de la soldadura es menor que el tiempo de purga del gas, o si el argumento *\On* no ha sido utilizado, el robot esperará en la posición de arranque hasta que el tiempo de purga del gas haya transcurrido.

tiempo_preflujoTipo de dato: *num*

Es el tiempo (en segundos) utilizado para el preflujo de la pieza a soldar, con gas protector, denominado “gas de preflujo”.

El robot está parado en la posición durante este tiempo antes de que el arco se encienda.

prog_enc*(programa de encendido)*Tipo de dato: *num*

Es la identidad (expresada como un número) de un programa de soldadura en un equipo de soldadura conectado. La información es enviada al equipo de soldadura para ser utilizada durante la fase de encendido del arco.

Véase Parámetros del Sistema *Soldadura al Arco - Equipo - tipo_portprog*.

tensión_encTipo de dato: *num*

Es la tensión de soldadura (en voltios) durante el encendido del arco.

El valor especificado es escalado y enviado a la salida analógica correspondiente, de acuerdo con los datos especificados en los parámetros del sistema, referentes a las señales analógicas.

velhilo_encTipo de dato: *num*

Es la velocidad de alimentación del electrodo de soldadura durante el encendido del arco.

La unidad está definida en el parámetro del sistema *Soldadura al Arco - Unidades - unidad_alimentación* y, de modo general, se expresa en m/minuto o en pulgadas por minuto.

El valor especificado es escalado y enviado a la salida analógica correspon-

diente, de acuerdo con los datos determinados en los parámetros del sistema para las señales analógicas.

corr_encTipo de dato: *num*

La corriente de soldadura durante el encendido del arco.

El valor especificado es escalado y enviado a la salida analógica correspondiente, de acuerdo con los datos determinados en los parámetros del sistema para las señales analógicas.

aj_tensión_enc(ajuste de la tensión de encendido) Tipo de dato: *num*

Es el ajuste de tensión de soldadura durante el encendido del arco.

El valor especificado es escalado y enviado a la salida analógica correspondiente, de acuerdo con los datos determinados en los parámetros del sistema para las señales analógicas.

Esta señal puede ser utilizada para otros propósitos cuando se necesita controlar un dispositivo mediante una señal de salida analógica.

aj_corr_enc(ajuste de la corriente de encendido) Tipo de dato: *num*

El ajuste de corriente durante el encendido del arco.

El valor especificado es escalado y enviado a la salida analógica correspondiente, de acuerdo con los datos determinados en los parámetros del sistema para las señales analógicas.

Esta señal puede ser utilizada para otros propósitos cuando se debe controlar un dispositivo mediante una señal de salida analógica.

retraso_mov_enc(retraso movimiento de encendido) Tipo de dato: *num*

Es el retraso (en segundos) a partir del tiempo en que el arco se considera estable en el encendido hasta que la fase de calentamiento haya comenzado. Las referencias de encendido permanecen válidas durante el retraso del movimiento de encendido.

arranque_oscil(tipo de arranque del arco) Tipo de dato: *num*

Tipo de encendido del arco por oscilación para el inicio de soldadura. El tipo de encendido por oscilación al rearanque no se verá afectado (será siempre por oscilación).

Tipos de arranque de oscilación:

0 Sin oscilación. No se producirá ninguna oscilación al inicio de soldadura.

1 Arranque por oscilación.

2 Arranque rápido. El robot no espera la señal OK de arco en el punto de arranque. No obstante, el encendido será considerado incorrecto si el tiempo de encendido ha sido excedido.

Grupo de componentes: Calentamiento**vel_cal** Tipo de dato: *num*

Es la velocidad de la soldadura durante el calentamiento al principio de la fase de soldadura.

La unidad utilizada está definida en el parámetro del sistema *Soldadura al Arco - Unidades- velocidad_unidad* y, de modo general, se expresa en mm/s o en pulgadas por minuto.

tiempo_cal Tipo de dato: *num*

Es el tiempo de calentamiento (en segundos) al principio de la fase de soldadura.

Se utiliza únicamente en el posicionamiento temporizado y cuando *dist_cal* o *vel_cal* es igual a cero.

dist_cal Tipo de dato: *num*

Es la distancia durante la cual los datos de calentamiento deben estar activados al principio de la soldadura.

La unidad utilizada está definida en los *Parámetros del Sistema - Soldadura al Arco - Unidades - longitud_unidad* y, de modo general, se expresa en mm. o en pulgadas.

prog_cal (*programa de calentamiento*) Tipo de dato: *num*

Es la identidad (expresada como un número) de un programa de soldadura en un equipo de soldadura conectado. La información es enviada al equipo de soldadura cuando el arco ha sido encendido y se utiliza durante el encendido.

Véase el Parámetro del Sistema *Soldadura al Arco - Equipo - tipo_portprog*.

tensión_cal Tipo de dato: *num*

Es la tensión de soldadura (en voltios) durante la fase de calentamiento.

El valor especificado es escalado y enviado a la salida analógica correspondiente, de acuerdo con los datos determinados en los parámetros del sistema para las señales analógicas.

velhilo_cal Tipo de dato: *num*

Es la velocidad de alimentación del electrodo de soldadura durante la fase de calentamiento.

La unidad utilizada está definida en el parámetro del sistema *Soldadura al Arco - Unidades - unidad_alimentación* y, de modo general, se expresa en m/minuto o en pulgadas por minuto.

El valor especificado es escalado y enviado a la salida analógica correspondiente, de acuerdo con los datos determinados en los parámetros del sistema para las señales analógicas.

corr_cal Tipo de dato: *num*

Es la corriente de soldadura durante el calentamiento.

El valor especificado es escalado y enviado a la salida analógica correspondiente, de acuerdo con los datos determinados en los parámetros del sistema para las señales analógicas.

(ajuste de la tensión de calentamiento)

Tipo de dato: *num*

Es el ajuste de la tensión para la fase de calentamiento.

El valor especificado es escalado y enviado a la salida analógica correspondiente, de acuerdo con los datos determinados en los parámetros del sistema para las señales analógicas.

Esta señal puede utilizarse para otros propósitos cuando se necesita controlar un dispositivo mediante una señal de salida analógica.

aj_corr_cal

(ajuste de la corriente de calentamiento)

Tipo de dato: *num*

Es el ajuste de corriente durante la fase de calentamiento.

El valor especificado es escalado y enviado a la salida analógica correspondiente, de acuerdo con los datos determinados en los parámetros del sistema para las señales analógicas.

Esta señal puede utilizarse para otros propósitos cuando se necesita controlar un dispositivo mediante una señal de salida analógica.

Grupo de componentes: Final

tiempo_enfr (*tiempo de enfriamiento*) Tipo de dato: num

Es el tiempo (en segundos) durante el cual el proceso se cierra antes de que otras actividades finales (como el llenado) tengan lugar.

tiempo_llen Tipo de dato: *num*

Es el tiempo de llenado del cráter (en segundos) en la fase final de la soldadura.

tiempo_postq (*tiempo de postquemado*) Tipo de dato: num

Es el tiempo (en segundos) durante el cual el electrodo de soldadura es post quemado cuando la alimentación del electrodo se ha detenido. Esto sirve para impedir que el electrodo quede pegado a la soldadura cuando se desactiva un proceso MIG/MAG.

El tiempo de postquemado se utiliza dos veces en la fase final; primero, cuando la fase de soldadura se acaba, en segundo lugar, después del llenado del cráter.

tiempo_enroll *(tiempo de enrollado)* Tipo de dato: *num*

Es el tiempo (en segundos) durante el cual el hilo es enrollado después de haber desconectado la alimentación de potencia. Esto sirve para impedir que el hilo quede pegado a la soldadura cuando se desactiva un proceso TIG.

Las funciones de *postquemado* y *enrollado* son mutuamente exclusivas.

tiempo_postflujo Tipo de dato: *num*

Es el tiempo (en segundos) requerido para purgar con gas protector después del final de un proceso. El postflujo del gas sirve para impedir que el electrodo de soldadura y el cordón se oxiden durante el tiempo de enfriamiento.

prog_llen *(programa de acabado)* Tipo de dato: *num*

La identidad (expresada como un número) de un programa de soldadura de un equipo de soldadura conectado. La información es enviada al equipo de soldadura cuando la fase de soldadura haya acabado y es utilizada cuando se realiza el llenado del cráter.

Véase el Parámetro del Sistema *Soldadura al Arco - Equipo - tipo_portprog*.

tensión_llen *(tensión del llenado del cráter)* Tipo de dato: *num*

La tensión de soldadura (en voltios) durante el llenado del cráter en la fase final de un proceso.

El valor especificado es escalado y enviado a la salida analógica correspondiente, de acuerdo con los datos determinados en los parámetros del sistema para las señales analógicas.

velhilo_llen *(alimentación del hilo del llenado del cráter)* Tipo de dato: *num*

Es la velocidad de alimentación del electrodo de soldadura en la fase del llenado del cráter.

La unidad utilizada está definida en el parámetro del sistema *Soldadura al Arco - Unidades - unidad_alimentación* y, de modo general, se expresa en m/minuto o en pulgadas por minuto.

El valor especificado es escalado y enviado a la salida analógica correspondiente, de acuerdo con los datos determinados en los parámetros del sistema para las señales analógicas.

corr_llen *(corriente de llenado del cráter)* Tipo de dato: *num*

Es la corriente de soldadura utilizada durante el llenado del cráter en la fase final de un proceso.

El valor especificado es escalado y enviado a la salida analógica correspondiente, de acuerdo con los datos determinados en los parámetros del sistema para las señales analógicas.

aj_tensión_llen (*ajuste de la tensión del llenado del cráter*) Tipo de dato: *num*

Es el ajuste de la tensión durante la fase de llenado del cráter.

El valor especificado es escalado y enviado a la salida analógica correspondiente, de acuerdo con los datos determinados en los parámetros del sistema para las señales analógicas.

Esta señal puede utilizarse para otros propósitos cuando algún dispositivo debe ser controlado mediante una señal de salida analógica.

aj_corr_llen (*ajuste de la corriente de llenado*) Tipo de dato: *num*

Es el ajuste de la corriente durante la fase de llenado del cráter.

El valor especificado es escalado y enviado a la salida analógica correspondiente, de acuerdo con los datos determinados en los parámetros del sistema para las señales analógicas.

Esta señal puede utilizarse para otros propósitos cuando algún dispositivo debe ser controlado mediante una señal de salida analógica.

Estructura

```
<dato del tipo seamdata>
<tiempo_purga_gas de num>
<tiempo_preflujo_gas de num>
<prog_enc de num>
<tensión_enc de num>
<velhilo_enc de num>
<corr_enc de num>
<aj_tensión_enc de num>
<aj_corr_enc de num>
<retraso_mov_enc de num>
<vel_cal de num>
<tiempo_cal de num>
<dist_cal de num>
<prog_cal de num>
<tensión_cal de num>
<velhilo_cal de num>
<corr_cal de num>
<aj_tensión_cal de num>
<aj_corr_cal de num>
<tiempo_enfr de num>
<tiempo_llen de num>
<tiempo_postq de num>
<tiempo_enroll de num>
<tiempo_postflujo_gas de num>
<prog_fin de num>
<tensión_llen de num>
<velhilo_llen de num>
```

<corr_llen de num>
<aj_tensión_llen de num>
<aj_corr_llen de num>

Observar que la estructura cambia dependiendo de la configuración del robot.

Información relacionada

Descripción en:

Datos de soldadura	Tipo de datos - <i>welldata</i>
Parámetros de instalación para el equipo y funciones de soldadura	Parámetros del Sistema - <i>Soldadura al Arco</i>
Fases del proceso y esquemas de tiempo	Resumen RAPID - <i>Soldadura al Arco</i>
Instrucciones de soldadura al arco	Instrucciones - <i>ArcL, ArcC</i>

,weavedata**Datos de oscilación**

Weavedata sirve para definir cualquier oscilación que se produzca durante la soldadura al arco.

La oscilación puede tener lugar durante las fases de calentamiento y de soldadura de una costura.

Descripción

La oscilación es un movimiento superpuesto a la trayectoria básica del proceso.

Existen tres tipos de trayectoria de oscilación entre las cuales se puede escoger: en zigzag, en forma de V y triangular. Estos tres tipos se encuentran indicados en la Figura 1 a Figura 3.

Todos los componentes de los datos de oscilación se aplican a la fase de calentamiento y a la fase de soldadura.

La unidad de los componentes de los datos de oscilación, que especifica una distancia y que está definida en el parámetro *Soldadura al Arco - Unidades - longitud_unidad* está de modo general expresada en mm o en pulgadas. (Estos componentes son longitudes de *ciclo_oscilación* (para oscilación de muñeca y geométrica), *_anchura*, *_altura*, *_desviación e intervalo*).

Nota Algunos de los componentes de los datos de oscilación dependen de la configuración del robot. Si se omite una característica dada, el componente correspondiente quedará fuera de los datos de oscilación. Las condiciones que deben cumplirse para que los componentes existan están descritas en los parámetros del sistema.

Componentes**forma_osc**

(*forma de la oscilación de soldadura*) Tipo de dato: *num*

La forma de la trayectoria de la oscilación en la fase de soldadura.

<u>Valor especificado</u>	<u>Trayectoria de oscilación</u>
0	Sin oscilación.
1	Oscilación en zigzag según se indica en la Figura 1.

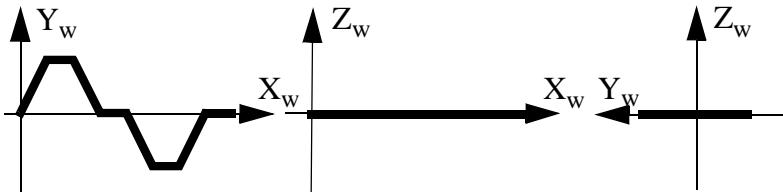


Figura 1 La oscilación en zigzag produce una oscilación horizontal respecto a la costura.

<u>Valor especificado</u>	<u>Tipo de oscilación</u>
2	Oscilación en forma de V según se indica en la Figura 2.

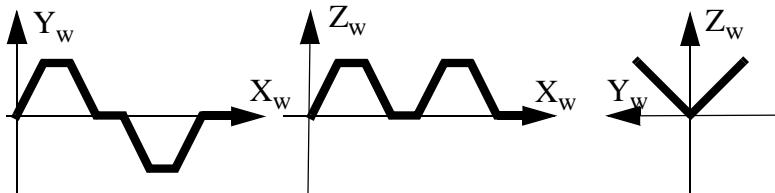


Figura 2 La oscilación en V produce una oscilación en forma de una V, vertical respecto a la costura.

3	Oscilación triangular segúnd se indica en la Figura 3.
---	--

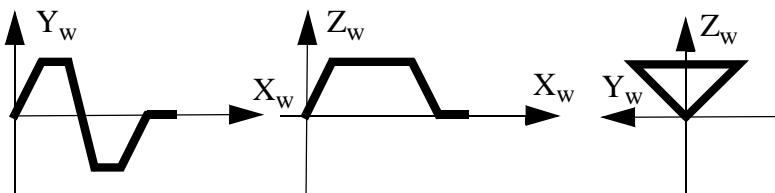


Figura 3 La oscilación triangular produce una oscilación en forma de triángulo, vertical respecto a la costura.

tipo_osc *(tipo de interpolación de oscilación de soldadura)*
Tipo de dato: num

Es el tipo de oscilación utilizado en la fase de soldadura.

<u>Valor especificado</u>	<u>Tipo de oscilación</u>
0	Oscilación geométrica. Durante la oscilación, todos los ejes son utilizados.
1	Oscilación de la muñeca.
2	Oscilación rápida. Se utilizan los ejes 1, 2 y 3.
3	Oscilación rápida. Se utilizan los ejes 4, 5 y 6.

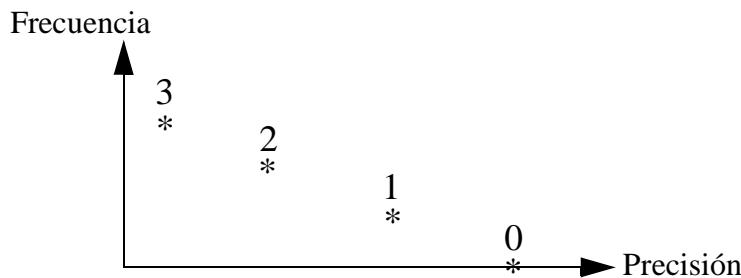


Figura 4 Diferencias entre los tipos de oscilación.

ciclo_oscilación

Tipo de dato: num

Existen dos conceptos en el componente *ciclo_oscilación*:

Longitud

El componente *ciclo_oscilación* está definido como una longitud del ciclo de oscilación en la fase de soldadura para los tipos de oscilación 0 y 1 (véase la Figura 5).

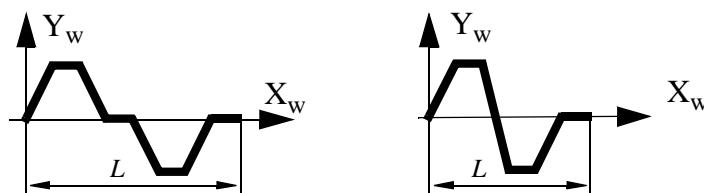
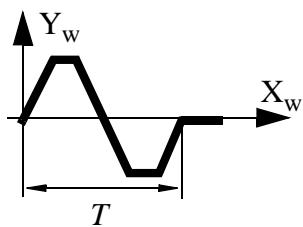


Figura 5 La longitud (L) del ciclo de oscilación para una oscilación en zig-zag, en forma de V y triangular.

Frecuencia

El componente *ciclo_oscilación* está definido como la frecuencia del ciclo de oscilación en la fase de soldadura para los tipos de oscilación 2 y 3 (véase la Figura 6).



T = Tiempo de ciclo de oscilación

f = Frecuencia de oscilación

$$f = \frac{1}{T}$$

Figura 6 La frecuencia (f) del ciclo de oscilación para una oscilación en zig-zag.

ancho_osc

Tipo de dato: num

Es la anchura de la trayectoria de oscilación en la fase de soldadura (véase la Figura 7).

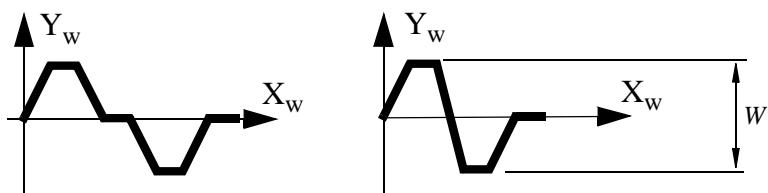


Figura 7 La anchura (W) de la oscilación para todos los tipos de trayectorias de oscilación.

altura_osc

Tipo de dato: num

Es la altura de la trayectoria de oscilación en una oscilación en forma de V o triangular (véase la Figura 8).

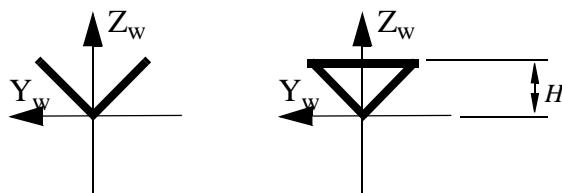


Figura 8 La altura (H) de la trayectoria de oscilación para una oscilación en forma de V y triangular.

interv_izq

Tipo de dato: num

Es la longitud del intervalo utilizado para obligar al TCP a moverse únicamente en la dirección de la costura en el punto de giro de la izquierda de la oscilación (véase la Figura 9).

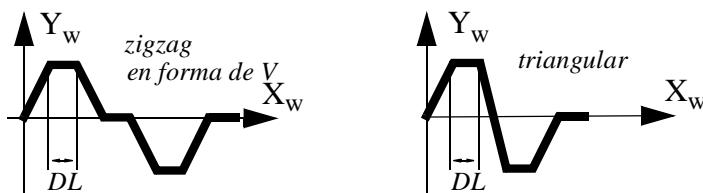


Figura 9 La longitud del intervalo de la izquierda (DL) para los diferentes tipos de oscilación.

interv_cent

Tipo de dato: num

Es la longitud del intervalo utilizado para obligar al TCP a moverse únicamente en la dirección de la costura en el punto central de la oscilación (véase la Figura 10).

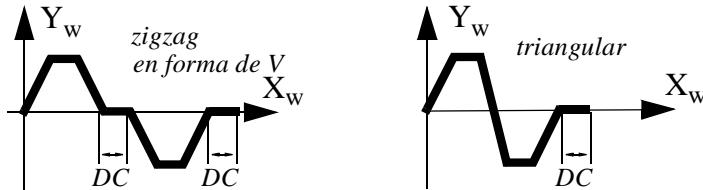


Figura 10 La longitud del intervalo central (DC) para los diferentes tipos de oscilación.

interv_der

Tipo de dato: num

Es la longitud del intervalo utilizado para obligar al TCP a moverse únicamente en la dirección de la costura en el punto de giro de la derecha de la oscilación (véase la Figura 11).

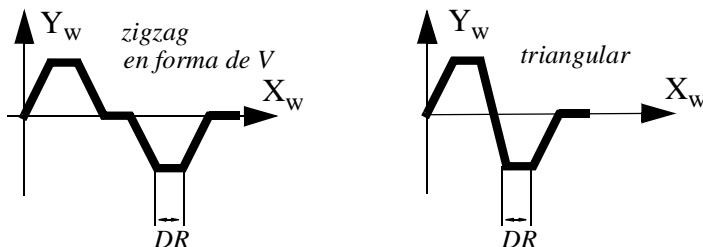


Figura 11 La longitud del intervalo de la derecha (DR) para los diferentes tipos de oscilación.

dir_osc

(ángulo de dirección de la oscilación)

Tipo de dato: num

Es el ángulo de dirección de la oscilación, horizontal respecto a la costura (véase la Figura 12). Un ángulo de cero grados produce una oscilación vertical respecto a la costura.

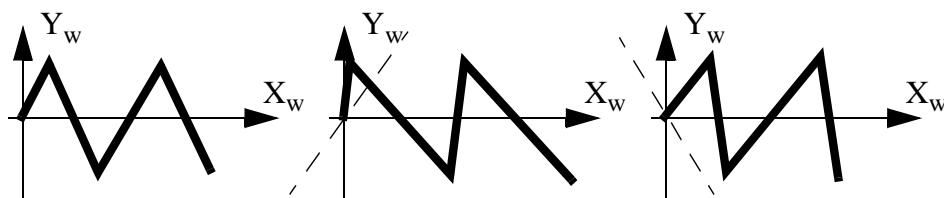


Figura 12 La forma de la trayectoria de oscilación a 0 grados y a un ángulo positivo y negativo.

incl_osc

(ángulo de inclinación de la oscilación)

Tipo de dato: num

Es el ángulo de inclinación del ángulo, vertical a la costura (véase la Figura 13). Un ángulo de cero grados produce una oscilación vertical respecto a la costura.

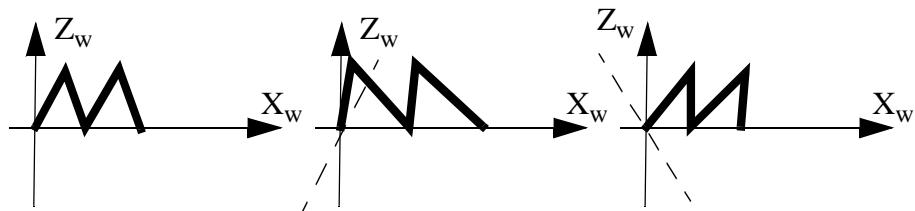


Figura 13 La oscilación en forma de V a 0 grados y a un ángulo positivo y negativo.

ori_osc

(ángulo de orientación de la oscilación)

Tipo de dato: num

Es el ángulo de orientación de la oscilación, horizontal-vertical respecto a la costura (véase la Figura 14). Un ángulo de cero grados produce una oscilación simétrica.

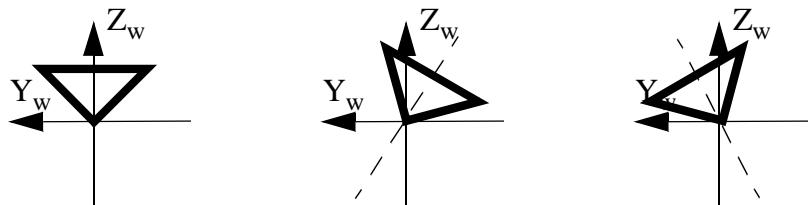


Figura 14 La oscilación triangular a 0 grados y a un ángulo positivo y negativo.

desv_osc

(desviación del centro de la oscilación)

Tipo de dato: num

Es la desviación horizontal respecto a la trayectoria de oscilación (véase la Figura 15). La desviación sólo podrá ser especificada para la oscilación en zigzag y no deberá ser mayor que la mitad de la anchura de la oscilación.

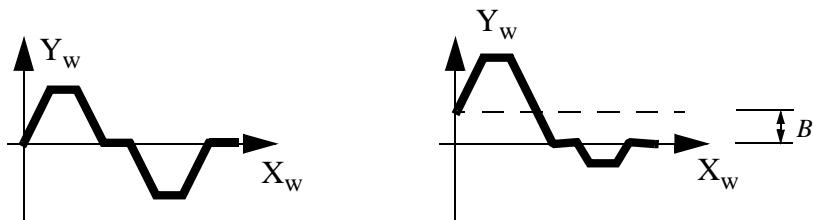


Figura 15 La oscilación en zig-zag con y sin desviación (B).

coord_izq

Tipo de dato: num

Es la posición de coordinación a la izquierda de la trayectoria de oscilación. Está especificada como un porcentaje de la anchura a la izquierda del centro de la oscilación. Cuando la oscilación se efectúa más allá de este punto, habrá una señal de salida digital que automáticamente se pondrá a uno, según se indica en la Figura 16. Este tipo de coordinación está concebida para el seguimiento de las costuras utilizando la función "WeldGuide" (Guía de Soldadura).

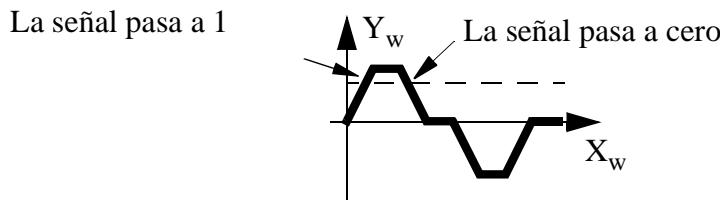


Figura 16 Cuando se utiliza "WeldGuide" (Guía de Soldadura), se requiere una señal de sincronización.

coord_der

Tipo de dato: num

Es la posición de coordinación a la derecha de la trayectoria de oscilación. Está especificada como un porcentaje de la anchura a la derecha del centro de la oscilación. Cuando la oscilación se efectúa más allá de este punto, habrá una señal de salida digital que automáticamente se pondrá en uno, según se indica en la Figura 17. Este tipo de coordinación está concebida para el seguimiento de las costuras utilizando la función "WeldGuide" (Guía de Soldadura).

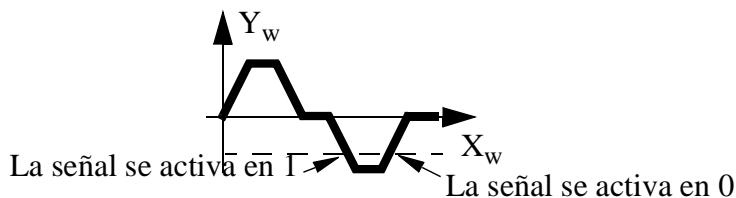


Figura 17 Cuando se utiliza "WeldGuide" (Guía de Soldadura), se requiere una señal de sincronización.

wg_track_on

Tipo de dato: num

Activación del sensor de la guía de soldadura de la costura.

Limitaciones

La frecuencia de oscilación máxima es de 2 Hz.

La inclinación de trayectoria de oscilación no deberá exceder la relación 1:10 (84 grados). Véase la Figura 18.

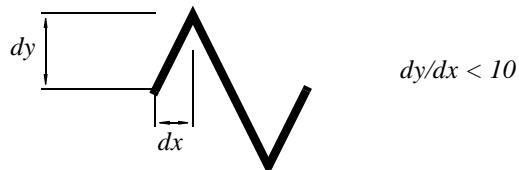


Figura 18 La trayectoria oscilación no deberá estar inclinada más que en la relación 1:10.

Estructura

<dato del tipo de weavedata>

<forma_osc de num>
<tipo_osc de num>
<long_osc de num>
<ancho_osc de num>
<altura_osc de num>
<interv_izq de num>
<interv_cent de num>
<interv_der de num>
<dir_osc de num>
<incl_osc de num>
<ori_osc de num>
<desv_osc de num>
<coord_izq de num>
<coord_der de num>
<wg_track_activado de num>

Información relacionada

Descripción:

Parámetros de instalación para el equipo y las funciones de soldadura

Parámetros del Sistema - *Soldadura al Arco*

Fases y tiempo de proceso

Resumen RAPID - *Soldadura al Arco*

Instrucciones de soldadura al arco Instrucciones - *ArcL, ArcC*

welldata

Datos de soldadura

Welldata sirve para controlar la soldadura durante la fase principal de soldadura, es decir, a partir del momento en que el arco está encendido hasta que la soldadura esté terminada.

Otras fases, como por ejemplo las fases de inicio y finales están controladas mediante los datos *seamdata*.

Descripción

Los datos de soldadura describen los datos que se cambian con frecuencia durante la soldadura de un cordón. Los datos de soldadura utilizados en una instrucción dada aplicados a una trayectoria, afectan a la soldadura hasta alcanzar la posición especificada. Utilizando instrucciones con datos de soldadura diferentes se podrá obtener un control óptimo del equipo de soldadura durante toda la costura.

Los datos de soldadura afectan a la soldadura cuando se ha establecido la fusión (después del calentamiento) al principio de un proceso.

En el caso de un *arranque sobre la marcha*, el arco no será encendido hasta que la posición de destino de la instrucción de soldadura al arco con el argumento *\On* haya sido alcanzada, lo que significa que los datos de soldadura no tienen ningún efecto en la soldadura en esta instrucción.

En el caso en que una instrucción de soldadura al arco se cambie por otra durante una soldadura, los datos de soldadura nuevos serán aplicados a partir de la posición central de la trayectoria esquina.

Nota Algunos de los componentes de los datos de soldadura dependen de la configuración del robot. En el caso en que se omita una característica dada, el componente correspondiente quedará fuera de los datos de soldadura. Las condiciones que deben cumplirse para que los componentes existan están descritas en los parámetros del sistema.

Todas las tensiones pueden expresarse de dos formas diferentes (determinadas por el equipo de soldadura):

- Como valores absolutos (en este caso, sólo se utilizan los valores positivos).
- Como correcciones de los valores especificados en el equipo de proceso (en este caso, se usarán tanto los valores positivos como los negativos).

El paso del electrodo de soldadura en este apartado se refiere a la soldadura MIG/

MAG. En el caso de la soldadura TIG:

- La señal de *hilo frío* se obtiene de *velocidad de hilo*.
- El valor de referencia necesario de la *corriente de soldadura* podrá ser conectado a cualquiera de las tres salidas analógicas que no se utilizan. (La referencia de *tensión de soldadura* no será utilizada).

Ejemplo

```
MoveJ p1, v100, z10, pistola1;
MoveJ p2, v100, fine, pistola1;
ArcL \On, p3, v100, cordón1, sold1, oscil1, fine, pistola1;
ArcL p4, v100, cordón1, sold2, oscil1, z10, pistola1;
ArcL \Off, p5, v100, cordón1, sold3, oscil3, fine, pistola1;
MoveJ p6, v100, z10, pistola1;
```

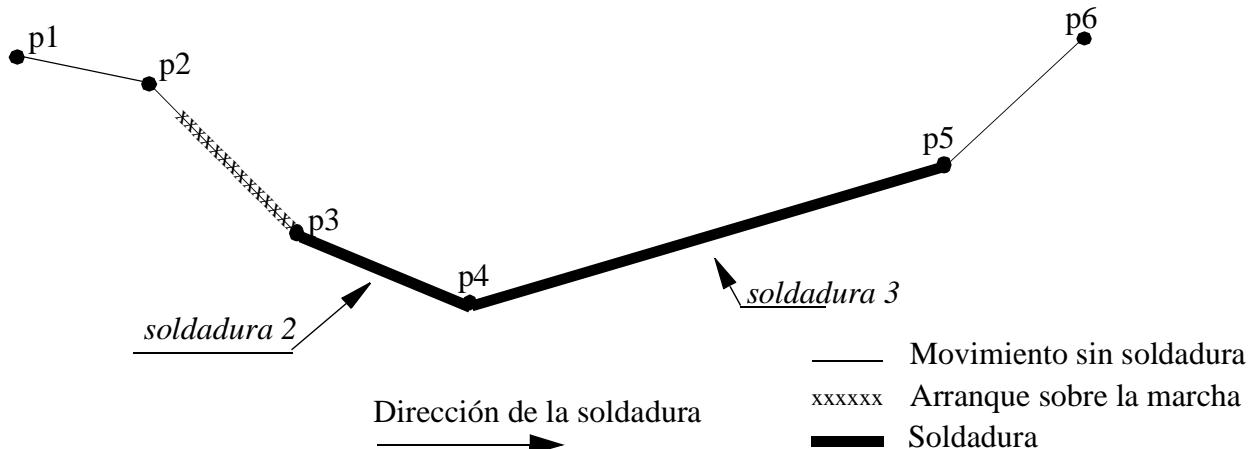
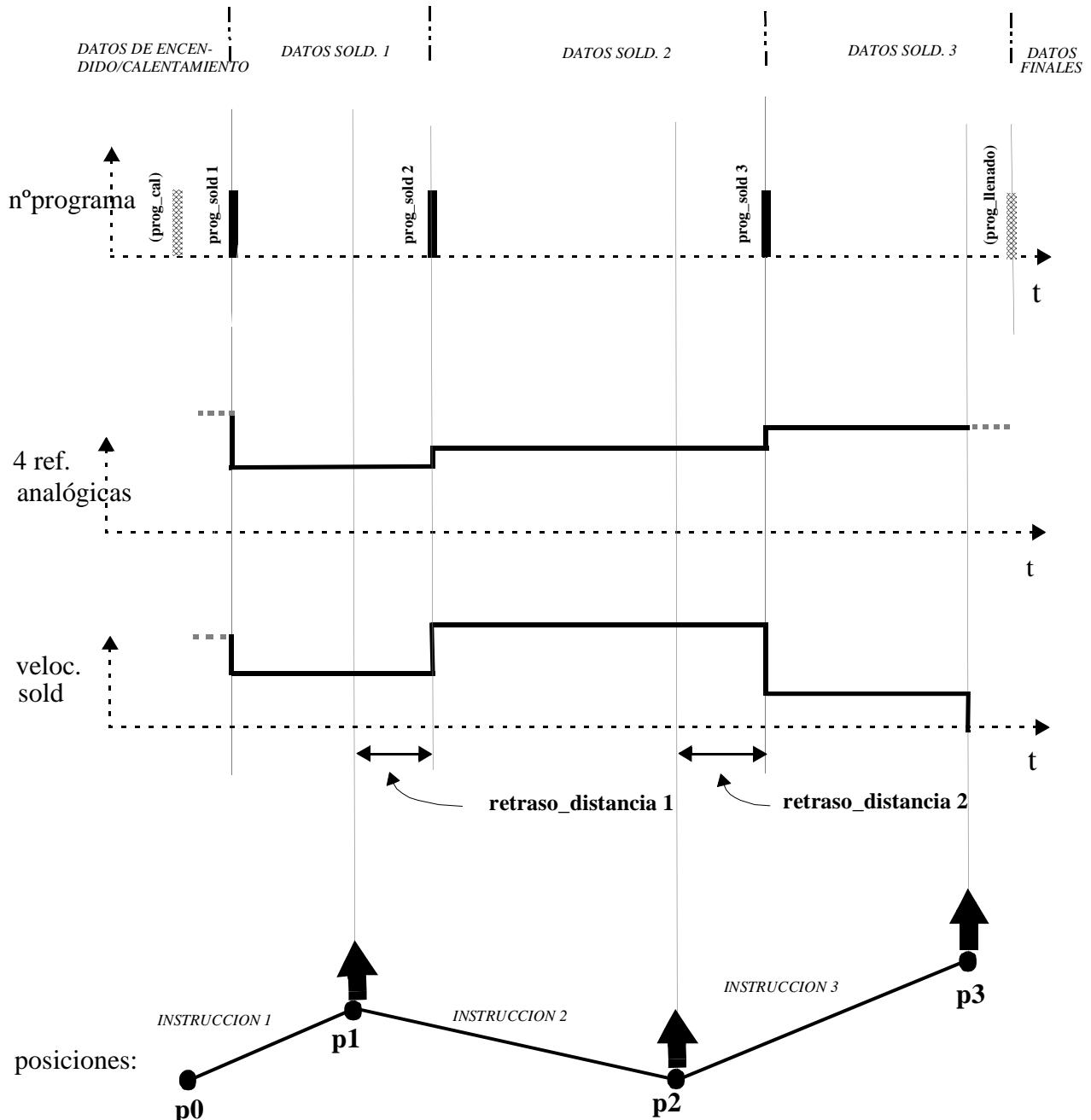


Figura 1 Los datos de soldadura, como la velocidad y la tensión de soldadura podrán cambiarse en cada posición.

El dato de soldadura es distinto para cada instrucción. Como que el argumento \On es utilizado en la primera instrucción, el primer dato de soldadura nunca es utilizado.

La secuencia de soldadura



Componentes

prog_sold *(programa de soldadura)* Tipo de dato: *num*

Es la identidad (expresada como un número) de los programas de soldadura que será enviada al equipo de soldadura.

Véase el Parámetro del Sistema *Soldadura al Arco - Equipo - tipo_portprog*.

vel_sold *Tipo de dato: num*

Es la velocidad de soldadura deseada.

La unidad está definida en el parámetro del sistema *Soldadura al Arco - Unidades - velocidad_unidad* y, de modo general, está expresada en mm/s o en pulgadas por minuto.

En el caso en que los movimientos de los ejes externos estén coordinados, la velocidad de soldadura será la velocidad relativa entre la herramienta y el objeto.

En el caso en que los movimientos de los ejes externos no estén coordinados, la velocidad de soldadura será la velocidad del TCP. La velocidad de los ejes externos está entonces descrita en los datos de velocidad de la instrucción. El eje más lento determina la velocidad para permitir que todos los ejes alcancen la posición de destino al mismo tiempo.

tensión_sold *Tipo de dato: num*

Es la tensión de soldadura (en voltios) durante la fase de soldadura.

El valor especificado es escalado y enviado a la salida analógica correspondiente, de acuerdo con los datos determinados en los parámetros del sistema para las señales analógicas.

velhilo_sold *Tipo de dato: num*

Es la velocidad de alimentación del electrodo de soldadura durante la fase de soldadura.

La unidad está definida en el parámetro del sistema *Soldadura al Arco - Unidades - unidad_alimentación* y, de modo general, está expresada en metros por minuto o en pulgadas por minuto.

El valor especificado es escalado y enviado a la salida analógica correspondiente, de acuerdo con los datos determinados en los parámetros del sistema para las señales analógicas.

corr_sold *Tipo de dato: num*

Es la corriente de soldadura durante la fase de soldadura.

El valor especificado es escalado y enviado a la salida analógica correspondiente, de acuerdo con los datos determinados en los parámetros del sistema para las

señales analógicas.

dist_retraso

Tipo de dato: *num*

Es la distancia de retraso (después de la posición de *destino*) para un cambio de un dato nuevo de soldadura en la siguiente instrucción de soldadura al arco.

La unidad está definida en los *Parámetros del Sistema - Soldadura al Arco - Unidades - longitud_unidad* y, de modo general, está expresada en mm o en pulgadas.

Generalmente, cuando se cambia una instrucción de soldadura al arco por otra, se genera un punto de paso. Ello producirá un punto de cambio en el medio de la trayectoria esquina. Utilizando una distancia de retraso, el nuevo dato de soldadura empezará a tener efecto más tarde (véase la Figura 2).

En una instrucción de *fin* de soldadura, la distancia de retraso no tendrá efecto.

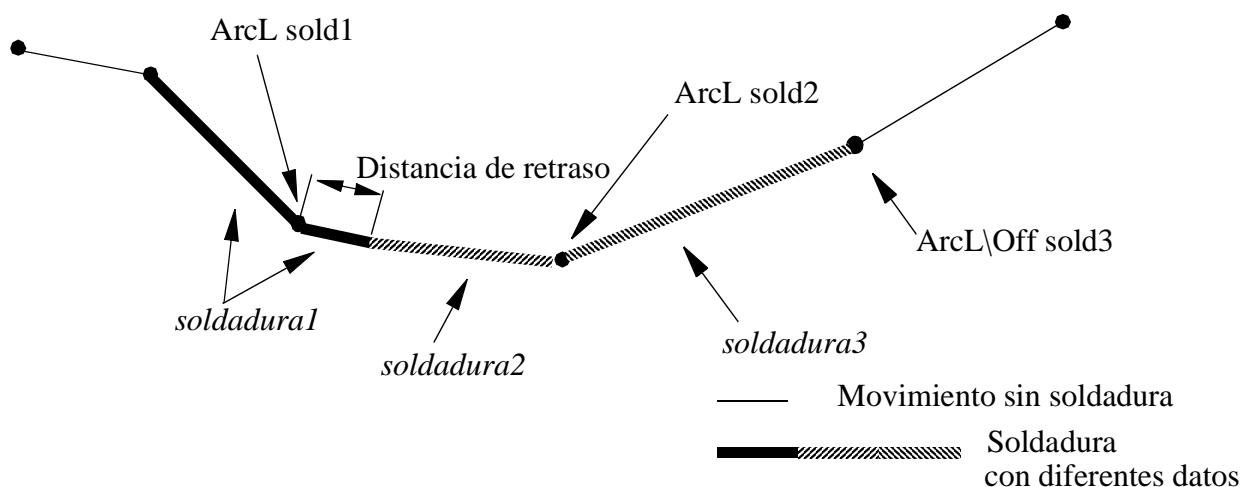


Figura 2 En este ejemplo, el cambio del dato de soldadura 1 por soldadura 2 está retrasado y soldadura 2 tiene una *dist_retraso* = 0. Así, la *distancia_retraso* en soldadura 3 no tendrá efecto.

La *dist_retraso* puede, por ejemplo, utilizarse en instrucciones *ArcC* para mover el cambio del dato de soldadura sin volver a programar las posiciones del círculo.

aj_tensión_sold

(ajuste de la tensión de soldadura) Tipo de dato: *num*

Es el ajuste de la tensión de soldadura durante la fase de soldadura.

El valor especificado es escalado y enviado a la salida analógica correspondiente, de acuerdo con los datos determinados en los parámetros del sistema para las señales analógicas.

Esta señal puede ser utilizada para otros propósitos cuando se necesita controlar un dispositivo mediante una señal de salida analógica.

aj_corr_sold

(ajuste de la corriente de soldadura) Tipo de dato: *num*

Es el ajuste de la corriente de soldadura durante la fase de soldadura.

El valor especificado es escalado y enviado a la salida analógica correspondiente, de acuerdo con los datos determinados en los parámetros del sistema para las señales analógicas.

Esta señal puede ser utilizada para otros propósitos cuando se necesita controlar un dispositivo mediante una señal de salida analógica.

vel_orig_sold (*velocidad original de soldadura*) Tipo de dato: *num*

Es la velocidad original de soldadura durante la fase de soldadura.

Nota: Usado a nivel interno por las funciones de ajuste.

tens_orig_sold (*tensión original de soldadura*) Tipo de dato: *num*

Es la tensión original de soldadura durante la fase de soldadura.

Nota: Usado a nivel interno por las funciones de ajuste.

alimhilo_orig_sold (*vel.aliment. hilo original sold.*) Tipo de dato: *num*

Es la velocidad de alimentación de hilo original de soldadura durante la fase de soldadura.

Nota: Usado a nivel interno por las funciones de ajuste.

Ejemplos

Se desea un tipo de soldadura como el de la Figura 3, con una tensión de soldadura de 30 V y una velocidad de alimentación de hilo de 15 m/min. La velocidad de soldadura es de 20 mm/s.

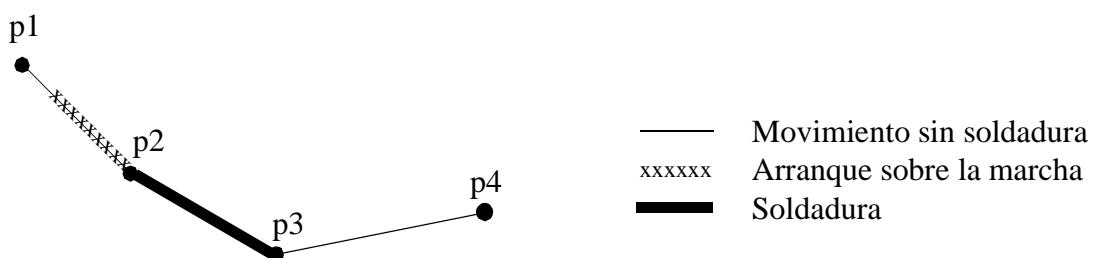


Figura 3. Soldadura entre dos puntos.

```
PERS welldata sold1 := [20,30,15,0];
```

```
MoveJ p1, v100, z20, pistola1;
ArcL \On, p2, v100, cordón1, sold1, nooscil, fine, pistola1;
ArcL \Off, p3, v100, cordón1, sold1, nooscil, fine, pistola1;
MoveJ p4, v100, z20, pistola1;
```

Los valores de los datos de soldadura de una soldadura como la de la Figura 3 son los siguientes:

<u>Componente</u>	<u>soldadura1</u>	
vel_sold	20 mm/s	Velocidad respecto a la costura
tensión_sold	30 V	Enviado a una señal de salida analógica
velhilo_sold	15 m/min.	Enviado a una señal de salida analógica
dist_retraso	0 mm	Sin retraso

La identidad del programa de soldadura, el ajuste de la tensión de soldadura y el ajuste de la corriente de soldadura no están activos en este ejemplo.

El argumento del dato de soldadura no tiene ningún efecto en la instrucción ArcL \On.

Se requiere un tipo de soldadura como el indicado en la Figura 4. La primera sección debe ser soldada utilizando una tensión de 50 V y una velocidad de alimentación de hilo de 20 m/min. Después de haber recorrido una cierta distancia en el arco circular, la tensión deberá ser aumentada a 55 V. La velocidad de soldadura es de 30 mm/s en cada sección.

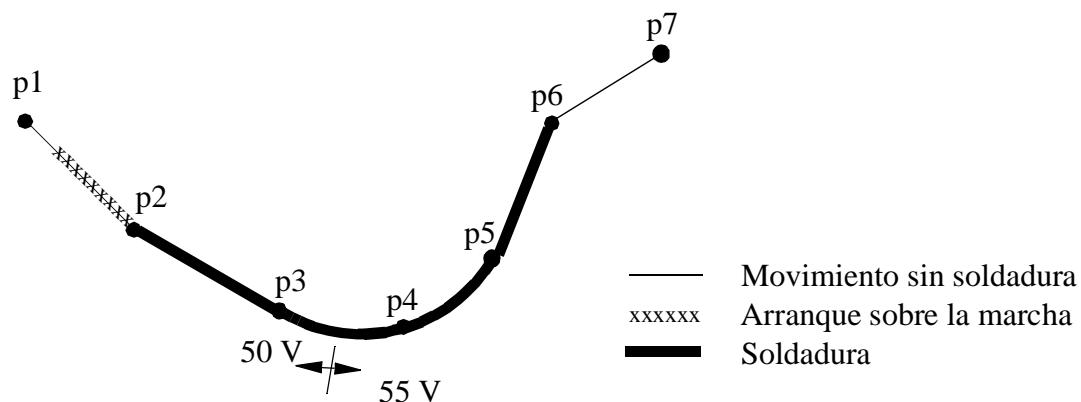


Figura 4 El dato de soldadura cambia después de cierta distancia especificada en la trayectoria circular.

```
PERS welldata sold1 := [10,30,50,20,0];
PERS welldata sold2 := [10,30,55,20,17];
```

```
MoveJ p1, v100, z20, pistola1;
ArcL \On, p2, v100, cordón1, sold1, nooscil, fine, pistola1;
ArcL p3, v100, cordón1, sold1, nooscil, z10, pistola1;
ArcC p4, p5, v100, cordón1, sold2, nooscil, z10, pistola1;
ArcL \Off, p6, v100, cordón1, sold2, nooscil, fine, pistola1;
MoveJ p7, v100, z20, pistola1;
```

Los valores de los datos de soldadura para una soldadura como la de la Figura 4 son los siguientes:

<u>Componente</u>	<u>sold.1</u>	<u>sold.2</u>	
prog_sold	10	10	Identidad enviada al equipo de soldadura
vel_sold	30 mm/s	30 mm/s	
tensión_sold	50 V	55 V	
velhilo_sold	20 m/min.	20 m/min.	
dist_retraso	0 mm	17 mm	sold.2 está retrasada de 17 mm

El ajuste de la tensión de soldadura y el ajuste de la corriente de soldadura no están activos en este ejemplo.

El argumento de datos de soldadura no tiene ningún efecto en la instrucción ArcL \On.

Estructura

```
<datos del tipo de welldata>
<prog_sold de num>
<veloc_sold de num>
<tensión_sold de num>
<velhilo_sold de num>
<corr_sold de num>
<dist_retraso de num>
<aj_tensión_sold de num>
<aj_corr_sold de num>
<vel_orig_sold de num>
<ten_orig_sold de num>
<alimhilo_orig_sold>
```

Observar que la estructura cambia según la configuración del robot.

Información relacionada

Descripción:

Datos iniciales y finales de soldadura

Tipos de datos - *seamdata*

Parámetros de instalación para el equipo y funciones de soldadura

Parámetros del Sistema - *Soldadura al Arco*

Fases del proceso

Resumen RAPID - *Soldadura al Arco*

Instrucciones de soldadura al arco

Instrucciones - *ArcL, ArcC*

ArcC Soldadura al arco con movimiento circular

ArcC1

ArcC2

ArcC (Arc Circular) sirve para soldar en una trayectoria circular. La instrucción controla y monitoriza el proceso completo de soldadura según lo siguiente:

- El punto central de la herramienta se mueve de forma circular a la posición de destino especificada.
- Todas las fases, como las fases de inicio y finales del proceso de soldadura, están controladas.
- El proceso de soldadura está monitorizado continuamente.

La única diferencia entre *ArcC*, *ArcC1* y *ArcC2* radica en que están conectadas a sistemas de proceso diferentes configurados en los Parámetros del Sistema. Aunque *ArcC* haya sido utilizado en los ejemplos, *ArcC1* o *ArcC2* podrán utilizarse también del mismo modo.

Ejemplo

```
MoveL ....
ArcL \On, p1, v100, cordón1, sold5, nooscil, fine, pistola1;
ArcC \Off, p2, p3, v100, cordón1, sold5, nooscil, fine, pistola1;
MoveL ....
```

Este ejemplo suelda un cordón circular entre los puntos *p1* y *p3* (pasando por el punto *p2*) según se indica en la Figura 1.

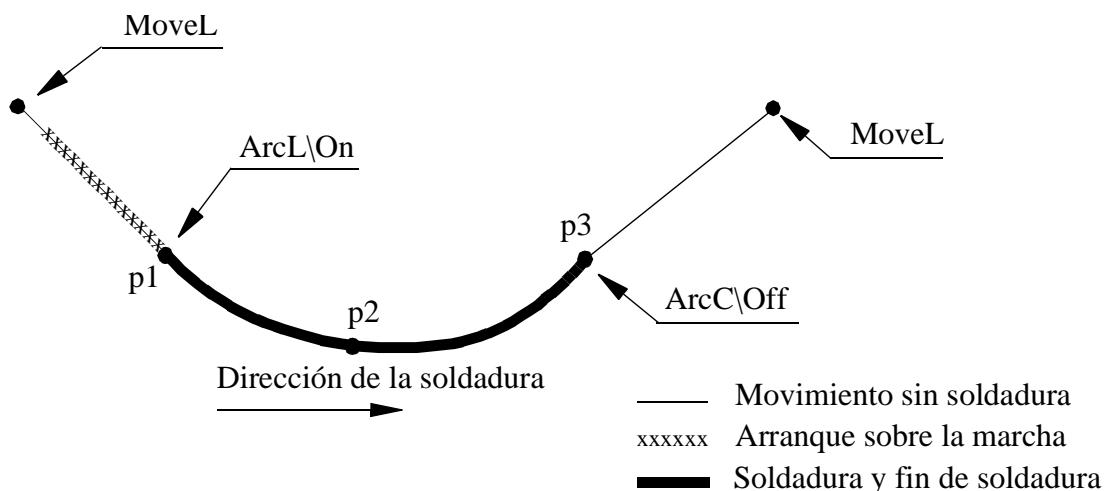


Figura 1 Soldadura con arranque sobre la marcha.

En la trayectoria hasta *p1*, se llevan a cabo los preparativos para el inicio de la soldadura, como el preflujo del gas. El proceso y el movimiento de soldadura actual empezarán entonces en la posición *p1* y acabarán en *p3*. Los procesos de inicio y final están determinados por *cordón1* y el proceso de soldadura por *sold5*. Los datos de oscilación se llevarán a cabo de acuerdo con *nooscil*. (Sin oscilación, si el valor del componente *forma_osc* es igual a cero.)

V100 especifica la velocidad alcanzada durante el arranque sobre la marcha hasta *p1*.

Argumentos

ArcC [\\On] | [\\Off] PuntCirc AlPunto Veloc [\\T] Cordón Sold Oscil Zona [\\Z] Herram [\\WObj]

[\\On]

Tipo de dato: *switch*

El argumento *On* sirve para obtener un *arranque sobre la marcha* (véase la Figura 1) que, a su vez, producirá tiempos de ciclo más cortos.

El argumento *On* sólo podrá utilizarse en la primera instrucción de soldadura al arco para obtener un cordón. Como que las instrucciones finales no pueden incluir el argumento *On*, la soldadura con arranque sobre la marcha deberá incluir por lo menos dos instrucciones.

Los preparativos de inicio de un arranque sobre la marcha, por ejemplo, la purga del gas, se llevarán a cabo en la trayectoria hacia la posición de inicio de la soldadura.

Cuando no se utiliza el argumento *On*, la soldadura empieza en la posición anterior a la instrucción *ArcC* (véase la Figura 2) y el robot permanecerá parado en la posición anterior mientras *todas* las actividades de inicio de soldadura se están llevando a cabo.

Tanto si se utiliza como si no se utiliza un arranque sobre la marcha, la posición de inicio de la soldadura será siempre un punto de paro - independientemente de lo que esté especificado en el argumento *Zona* para esta posición.

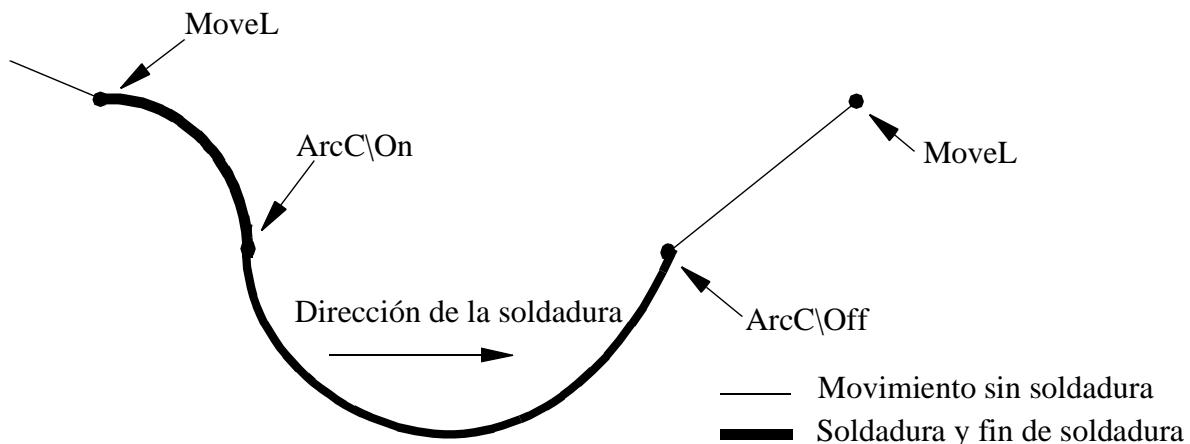


Figura 2 Si la soldadura es arrancada sin el argumento *On*, la soldadura empieza en la posición anterior.

[\\Off]

Tipo de dato: *switch*

Si se utiliza el argumento *Off*, la soldadura acaba cuando el robot alcance la posición de destino. Independientemente de lo que esté especificado en el argumento *Zona*, la posición de destino será un punto de paro.

Si a una instrucción *ArcC* sin el argumento *Off* le sigue una instrucción *MoveJ*, por ejemplo, la soldadura acabará, pero de una forma incontrolada. Las instruc-

ciones lógicas, como *Set do1*, sin embargo, podrán utilizarse entre dos instrucciones de soldadura al arco sin terminar el proceso de soldadura.

PuntCircTipo de dato: *robtarget*

Es el punto circular del robot. El punto circular es una posición situada en el círculo entre el punto de arranque y el punto de destino. Para una mayor precisión, deberá estar situado a medio camino entre los puntos de arranque y de destino. Si está situado demasiado cerca del punto de arranque o de destino, puede ocurrir que el robot genere un mensaje de aviso. El punto circular es definido como una posición con nombre o es almacenado directamente en la instrucción (marcado con un asterisco * en la instrucción).

AlPuntoTipo de dato: *robtarget*

Es la posición de destino del robot y de los ejes externos. Suele estar definido como una posición con nombre o es almacenado directamente en la instrucción (indicado por un asterisco * en la instrucción).

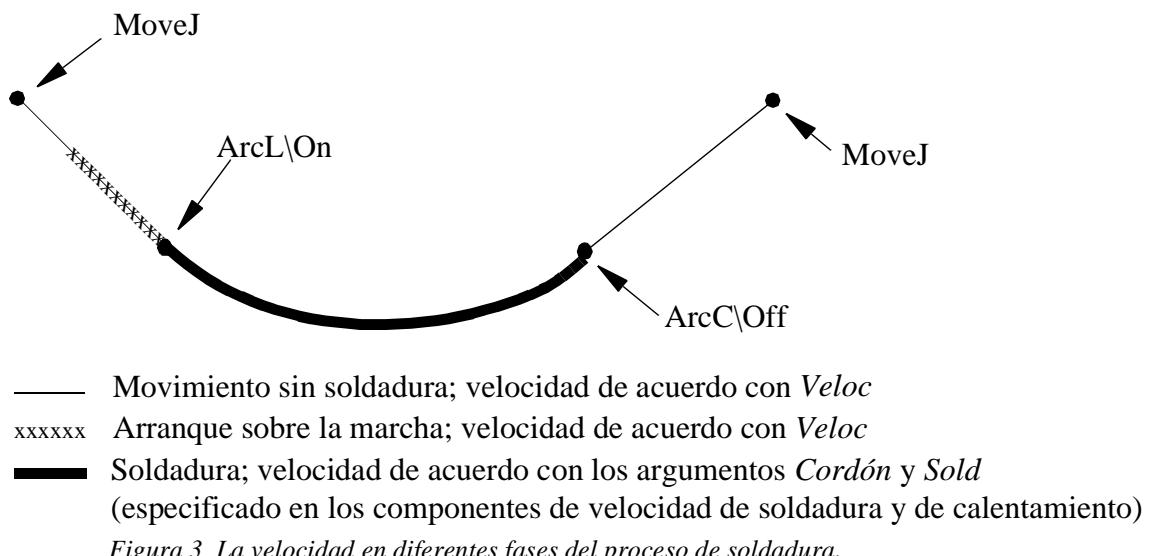
VelocidadTipo de dato: *speeddata*

La velocidad del TCP está controlada por el argumento *Veloc* en los siguientes casos:

- Cuando se utiliza el argumento *\On* (preparativos del inicio de la soldadura en un arranque sobre la marcha).
- Cuando el programa es ejecutado instrucción por instrucción (sin soldadura).

La velocidad del TCP *durante la soldadura* es la misma que la utilizada para los argumentos *Cordón* y *Sold* (Véase la Figura 3).

Los datos de velocidad describen también la velocidad de la reorientación de la herramienta y la velocidad de cualquier eje externo no coordinado.



[\T]	<i>(Tiempo)</i>	Tipo de dato: <i>num</i>
El argumento <i>\T</i> sirve para especificar el tiempo total de los movimientos determinado en segundos directamente en la instrucción. El tiempo será así sustituido por la velocidad especificada en los argumentos <i>Veloc</i> , <i>Cordón</i> y <i>Sold</i> .		
T		
Este argumento puede utilizarse cuando, por ejemplo, uno o más ejes externos no coordinados participan en el movimiento. Los ejes externos no coordinados deberán, no obstante, evitarse ya que, cuando se utilizan, el programa será más difícil de ajustar. En vez de ello, utilízense ejes externos coordinados. La oscilación es desactivada durante la ejecución de instrucciones ArcX con argumentos <i>\T</i> .		
Cordón		Tipo de dato: <i>seamdata</i>
Los datos iniciales y finales de soldadura describen las fases de inicio y finales de un proceso de soldadura.		
El argumento <i>Cordón</i> está incluido en todas las instrucciones de soldadura al arco, de forma que, independientemente de la posición del robot cuando se interrumpe el proceso, se produce un final y un rearranque de soldadura adecuados.		
Normalmente los mismos datos iniciales y finales de soldadura se utilizan en todas las instrucciones de un cordón.		
Sold		Tipo de dato: <i>welddata</i>
Los datos de soldadura describen la fase de soldadura de un proceso de soldadura.		
Los datos de soldadura a menudo son cambiados de una instrucción a la siguiente durante la soldadura de un cordón.		
Oscil		Tipo de dato: <i>weavedata</i>
Los datos de oscilación describen la oscilación que debe tener lugar durante las fases de calentamiento y de soldadura. La soldadura sin oscilación se obtiene especificando por ejemplo, el dato de oscilación <i>nooscil</i> . (No hay oscilación si el valor del componente <i>forma_osc</i> es igual a cero.)		
Zona		Tipo de dato: <i>zonedata</i>
Los datos de zona definen la distancia a la que deben estar los ejes, de la posición programada antes de poder empezar el movimiento hacia la posición siguiente.		
En el caso de un punto de paso, se generá una trayectoria esquina más allá que esta posición. En el caso de un punto de paro (<i>punto fino</i>), el movimiento es interrumpido hasta que todos los ejes hayan alcanzado el punto programado.		
Un punto de paro se genera siempre automáticamente en la posición de inicio de una soldadura (incluso en el caso de un <i>arranque sobre la marcha</i>) y en la posición final de soldadura <i>controlada</i> . Los puntos de paso, como <i>z10</i> , deberán utilizarse para todas las demás posiciones de soldadura.		
Los datos de soldadura cambian a la siguiente instrucción de soldadura al arco en el punto central de una trayectoria esquina. (a menos que esté retrasado por el		

componente *dist_retraso* del argumento *Sold*).

[\Z]	(Zona)	Tipo de dato: <i>num</i>
--------	--------	--------------------------

Este argumento sirve para especificar la precisión de posicionamiento del TCP del robot directamente en la instrucción. El tamaño de la zona está especificada en mm y es sustituida en la zona correspondiente determinada en los datos de zona. El argumento \Z es muy útil también cuando se trata de ajustar trayectorias esquina individuales.

Herram		Tipo de dato: <i>tooldata</i>
---------------	--	-------------------------------

Es la herramienta utilizada en el movimiento. El TCP de la herramienta es el punto que se mueve a la posición de destino especificada. El eje z de la herramienta deberá ser paralelo al elemento final.

[\WObj]	(Objeto de trabajo)	Tipo de dato: <i>wobjdata</i>
-----------	---------------------	-------------------------------

Es el objeto de trabajo (sistema de coordenadas) al que se refiere la posición del robot en la instrucción.

Cuando se omite este argumento, la posición del robot se referirá al sistema de coordenadas mundo. No obstante, deberá especificarse, si se utiliza un TCP estacionario o ejes externos coordinados.

\WObj podrá utilizarse si se ha definido un sistema de coordenadas para el objeto correspondiente o para el cordón de soldadura.

Ejecución del programa

Control del equipo de proceso

El equipo de proceso está controlado por el robot de tal forma que todo el proceso y cada una de sus fases están coordinadas con los movimientos del robot.

Movimiento

El robot y los ejes externos se mueven a la posición de destino según lo siguiente:

- El TCP de la herramienta se mueve de forma circular a una velocidad programada constante. Cuando se utilizan ejes coordinados, se moverán de forma circular a una velocidad programada constante respecto al objeto de trabajo.
- La herramienta es reorientada a intervalos regulares durante toda su trayectoria.
- Los ejes externos no coordinados se ejecutan a una velocidad constante, lo que significa que alcanzan su destino al mismo tiempo que los ejes del robot.

En el caso en que se exceda la velocidad programada de reorientación o de los ejes externos, estas velocidades serán limitadas, reduciendo la velocidad del TCP.

La posición de destino se refiere:

- al sistema de coordenadas del objeto especificado si se utiliza el argumento `\WObj` ;
- al sistema de coordenadas mundo si no se utiliza el argumento `\WObj` .

Limitaciones

En la oscilación, la distancia entre las posiciones programadas deberán ser más largas que el tiempo periódico de la oscilación. Si la distancia es más pequeña y si hay un cambio significativo del ángulo de la trayectoria, la trayectoria de oscilación será distorsionada.

No se deberá nunca rearrancar la instrucción *ArcC* después de haber pasado el punto de ciclo. De lo contrario, el robot no emprenderá la trayectoria programada (posicionamiento en torno a la trayectoria circular en otra dirección comparada con la programada).

Gestión de errores

El proceso está supervisado por una serie de señales de entrada. Si se detecta algo anormal, la ejecución del programa será detenida. Pero si, no obstante, se programa un gestor de errores, los errores que se definen a continuación podrán ser remediados sin detener el proceso de producción. Véase el ejemplo en la instrucción *RestoPath*.

Constante de errores (valor *ERRNO*)

Descripción

AW_START_ERR	Error de condición de inicio; supervisión de gas, agua, o antorcha
AW_IGNI_ERR	Error de encendido; supervisión del arco
AW_WELD_ERR	Error de soldadura; supervisión del arco
AW_EQIP_ERR	Error del equipo de soldadura; supervisión de tensión, de corriente, agua o gas durante la soldadura
AW_WIRE_ERR	Error de hilo; supervisión de hilo
AW_STOP_ERR	Soldadura interrumpida por la entrada de paro del proceso

La supervisión del proceso está determinada como una parte de la configuración del equipo de proceso.

Al *inicio* del proceso, el robot comprueba que se hayan cumplido las siguientes *condi-*

ciones preliminares:

- paro_proceso
- agua_OK
- gas_OK
- antorcha_OK

En el caso en que, después de que el comando de inicio haya sido ejecutado, no se indica ningún perfil de inicio aprobado en la entrada digital, *arc_OK*, dentro de un período de tiempo predeterminado, el inicio del proceso será interrumpido.

Una vez que el proceso haya comenzado, todas las entradas de supervisión seleccionadas serán continuamente monitorizadas:

- paro_proceso, agua_OK, gas_OK, arc_OK, tensión_OK, corriente_OK, velhilo_OK.

La supervisión wirestick_err será comprobada al final de la soldadura.

Ejemplo

```
MoveL ...
ArcC \On, *, *, v100, cordón1, sold5, oscil1, fine, pistola1\Wobj:=wobj1;
ArcC *, *, v100, cordón1,sold5, oscil1, z10, pistola1\Wobj:=wobj1;
ArcL *, v100, cordón1,sold5, oscil1, z10, pistola1\Wobj:=wobj1;
ArcC \Off, *, *, v100, cordón1,sold3, oscil3, fine, pistola1\Wobj:=wobj1;
MoveL ...
```

En este ejemplo, se lleva a cabo una soldadura en la que los datos de soldadura y los datos de oscilación son cambiados en la parte final de la soldadura, según se indica en la Figura 4. Observar que una instrucción de soldadura al arco deberá utilizarse para cambiar la dirección de la trayectoria a pesar de que no haya ningún dato de soldadura que se haya cambiado.

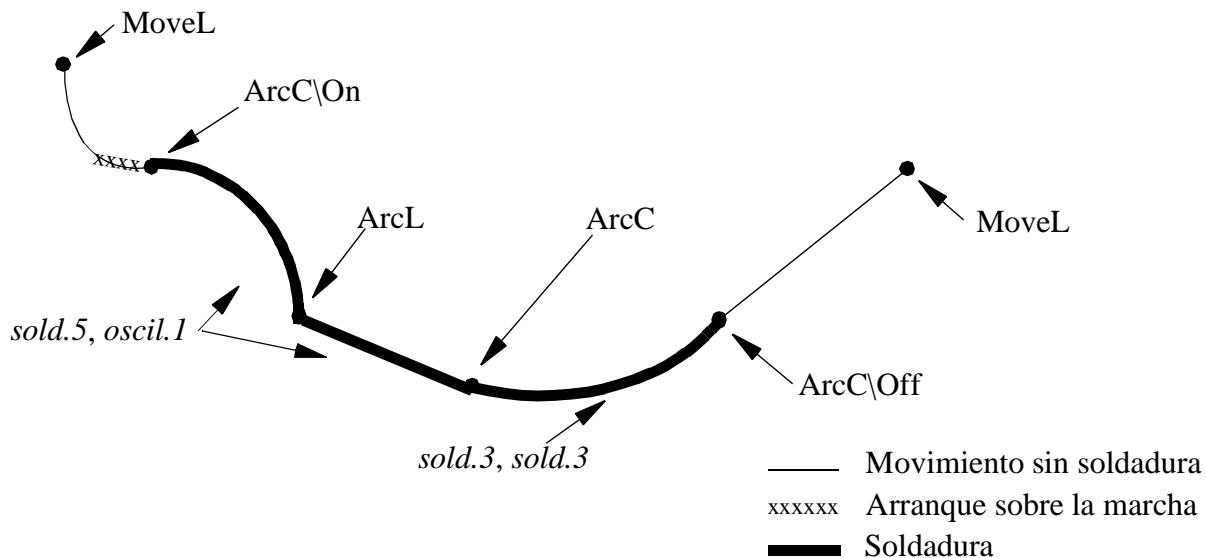


Figura 4 La dirección y los datos de soldadura podrán cambiarse programando varias instrucciones de soldadura al arco.

En este ejemplo, se presupone que se ha utilizado un eje externo coordinado para el movimiento. En este caso, el objeto de trabajo, *wobj1* deberá estar especificado en la instrucción.

Sintaxis

```
ArcC
[ '\'On',' ] | [ '\'Off',' ]
[ PuntCirc ':=' ] < expresión (IN) de robtarg > ',' 
[ AlPunto ':=' ] < expresión (IN) de robtarg > ',' 
[ Veloc ':=' ] < expresión (IN) de speeddata >
[ ( '\' T ':=' < expresión (IN) de num > ) ] ',' 
[ Cordón ':=' ] < persistente (PERS) de seamdata > ',' 
[ Sold ':=' ] < persistente (PERS) de welddata > ',' 
[ Oscil ':=' ] < persistente (PERS) de weavedata > ',' 
[ Zona ':=' ] < expresión (IN) de zonedata >
[ '\' Z ':=' < expresión (IN) de num > ] ',' 
[ Herram ':=' ] < persistente (PERS) de tooldata >
[ '\' WObj ':=' < persistente (PERS) de wobjdata > ] ','
```

Información relacionada

Realización de una soldadura lineal
Otras instrucciones de posicionamiento
Definición de la velocidad
Definición de los datos de zona
Definición de las herramientas
Definición de los objetos de trabajo
Definición de los datos iniciales y finales de soldadura
Definición de los datos de soldadura
Definición de los datos de oscilación
Parámetros de instalación para el equipo y funciones de soldadura
Movimientos en general
Sistemas de coordenadas
Fases del proceso y subactividades

Descripción:

Instrucciones - *ArcL*
Resumen RAPID - *Movimiento*
Tipos de Dato - *speeddata*
Tipos de Dato - *zonedata*
Tipos de Dato - *tooldata*
Tipos de Dato - *wobjdata*

Tipos de Dato - *seamdata*
Tipos de Dato - *welddata*
Tipos de Dato - *weavedata*
Parámetros del Sistema - *Soldadura al arco*
Principios de Movimiento
Principios de Movimiento - *Sistemas de coordenadas*
Resumen RAPID - *Soldadura al Arco*

arcdat

Datos de proceso de soldadura al arco

Arcdata (Arc Process Data) sirve para definir una serie de parámetros de proceso de soldadura al arco que son esenciales para el proceso:

- Los últimos parámetros utilizados (velocidad, velocidad de alim. de hilo y tensión).
 - Los datos estadísticos (Valores medios y máx./mín. de tensión y corriente en la última costura realizada).

Este tipo de dato normalmente no tiene que usarse dado que se puede acceder a estos valores utilizando la variable del sistema ARC_DATA.

Descripción

Los últimos parámetros utilizados (almacenados en la variable del sistema ARC_DATA) están afectados en cualquier circunstancia.

Los siguientes requisitos son aplicables para los datos estadísticos:

- El sistema es un sistema ARCITEC.
 - El sistema está configurado de forma a utilizar las entradas analógicas para tensión y corriente. En este caso se deberá usar un equipo de medida externa.

Componentes

req_vel	Tipo de dato: <i>num</i>
Ultima velocidad utilizada.	
req_volt	Tipo de dato: <i>num</i>
Ultima tensión de soldadura utilizada.	
req_wfd_sp	Tipo de dato: <i>num</i>
Ultima velocidad de alimentación de hilo utilizada.	
act_volt	Tipo de dato: <i>num</i>
Tensión de soldadura medida utilizada.	
min_volt	Tipo de dato: <i>num</i>
Valor más bajo medido de la tensión de soldadura durante la costura actual (o última costura).	

max_volt	Tipo de dato: <i>num</i>
Valor más alto medido de la tensión de soldadura durante la costura actual (o última costura).	
mean_volt	Tipo de dato: <i>num</i>
Valor medio de la tensión de soldadura medido durante la costura actual (o última costura).	
act_curr	Tipo de dato: <i>num</i>
Corriente de soldadura medida en este momento.	
min_curr	Tipo de dato: <i>num</i>
Valor más bajo medido de la corriente de soldadura durante la costura actual (o de la última costura).	
max_curr	Tipo de dato: <i>num</i>
Valor más alto medido de la corriente de soldadura durante la costura actual (o de la última costura).	
mean_curr	Tipo de dato: <i>num</i>
Valor medio de la corriente de soldadura medido durante la costura actual (o de la última costura).	

Ejemplo

```
ArcL\On ...
ArcL\Off ...
TPWrite "Valor medio de tensión para la última costura=
"\Num:=ARC_DATA.mean_volt;
```

Cuando la costura ha acabado, el valor medio de la tensión del parámetro de proceso aparecerá visualizado en la unidad de programación.

Estructura

```
<data object de arcdata>
  <req_vel de num>
  <req_volt de num>
  <req_wfd_sp de num>
  <act_volt de num>
  <min_volt de num>
  <max_volt de num>
  <mean_volt de num>
  <act_curr de num>
```

<min_curr de num>
<max_curr de num>
<mean_curr de num>

Información relacionada

Descripción en:

Parámetros de instalación para el equipo
de soldadura y funciones

Parámetros del Sistema - *Soldadura al
Arco*

Fases de proceso y programas de temporizaciónResumen RAPID - *Soldadura al Arco*
Instrucciones de soldadura al arco

Instrucciones - *ArcL, ArcC*

ArcL**ArcL1****ArcL2**

ArcL (*Arc Linear*) sirve para soldar en una trayectoria recta. La instrucción controla y monitoriza el proceso completo de soldadura según lo siguiente:

- El punto central de la herramienta se mueve de forma lineal a la posición de destino especificada.
- Todas las fases, como las fases de inicio y finales del proceso de soldadura, están controladas.
- El proceso de soldadura está monitorizado continuamente.

La única diferencia entre *ArcL*, *ArcL1* y *ArcL2* radica en que están conectadas a sistemas de proceso diferentes según la configuración de los Parámetros del Sistema. Aunque *ArcL* haya sido utilizado en los ejemplos, *ArcL1* o *ArcL2* podrán utilizarse también del mismo modo.

Ejemplo

```
MoveJ . . .
ArcL \On, p1, v100, cordón1, sold5, nooscil, fine, pistola1;
ArcL \Off, p2, v100, cordón1, sold5, nooscil, fine, pistola1;
MoveJ . . .
```

Este ejemplo suelda un cordón recto entre los puntos *p1* y *p2*, según se indica en la Figura 1.

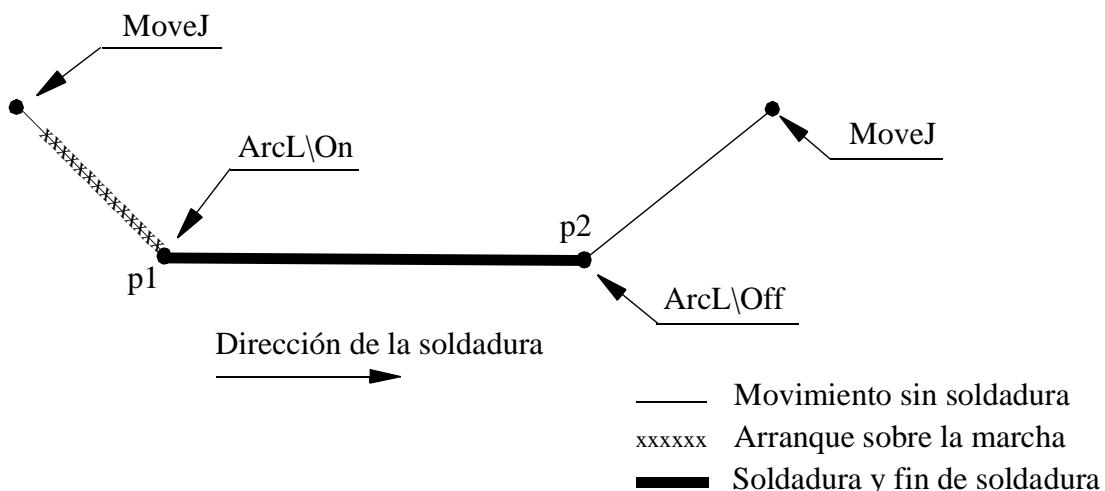


Figura 1 Soldadura con arranque sobre la marcha.

En la trayectoria hasta *p1*, se llevan a cabo los preparativos para el inicio de la soldadura, como el preflujo del gas. El proceso y el movimiento de soldadura actual empezarán entonces en la posición *p1* y acabarán en *p3*. El proceso de inicio y de final está determinado por *cordón1* y el proceso de soldadura por *sold5*. Los datos de oscilación se llevarán a cabo de acuerdo con *nooscil*. (Sin oscilación, si el valor del componente *forma_osc* es igual a cero.)

V100 especifica la velocidad alcanzada durante el arranque sobre la marcha hasta *p1*.

Arguments

ArcL [\\On] | [\\Off] AlPunto Veloc [\\T] Cordón Sold Oscil Zona [\\Z] Herram [\\WObj]

[\\On]

Tipo de dato: *switch*

El argumento *On* sirve para obtener un *arranque sobre la marcha* (véase la Figura 1) que, a su vez, producirá tiempos de ciclo más cortos.

El argumento *On* sólo podrá utilizarse en la primera instrucción de soldadura al arco para obtener un cordón. Como que las instrucciones finales no pueden incluir el argumento *On*, la soldadura con arranque sobre la marcha deberá incluir por lo menos dos instrucciones.

Los preparativos de inicio de un arranque sobre la marcha, por ejemplo, la purga del gas, se llevarán a cabo en la trayectoria hacia la posición de inicio de la soldadura.

Cuando no se utiliza el argumento *On*, la soldadura empieza en la posición anterior a la instrucción *ArcL* (véase la Figura 2) y el robot permanecerá estacionario en la posición anterior mientras *todas* las actividades de inicio de soldadura se están llevando a cabo.

Tanto si se utiliza como si no se utiliza un arranque sobre la marcha, la posición de inicio de la soldadura será siempre un punto de paro - independientemente de lo que esté especificado en el argumento *Zona* para esta posición.

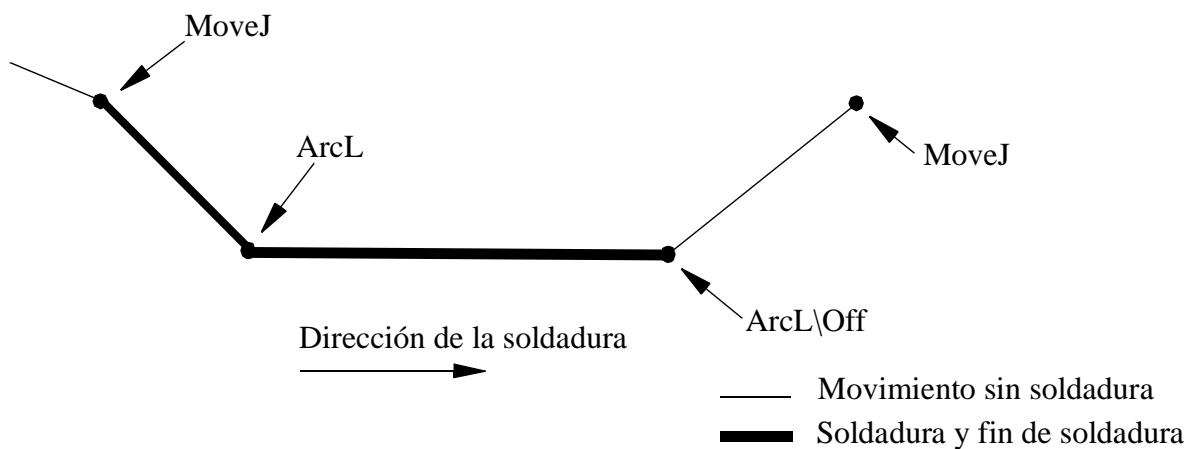


Figura 2 Si la soldadura es arrancada sin el argumento *On*, la soldadura empieza en la posición anterior.

[\\Off]

Tipo de dato: *switch*

Si se utiliza el argumento *Off*, la soldadura acaba cuando el robot alcance la posición de destino. Independientemente de lo que esté especificado en el argumento *Zona*, la posición de destino será un punto de paro.

Si a una instrucción *ArcL* sin el argumento *Off* le sigue una instrucción *MoveJ*,

por ejemplo, la soldadura acabará, pero de una forma incontrolada. Las instrucciones lógicas, como *Set do1*, sin embargo, podrán utilizarse entre dos instrucciones de soldadura al arco sin terminar el proceso de soldadura.

AlPuntoTipo de dato: *robtarget*

Es la posición de destino del robot y de los ejes externos. Suele estar definida como una posición con nombre o es almacenada directamente en la instrucción (indicada por un * en la instrucción).

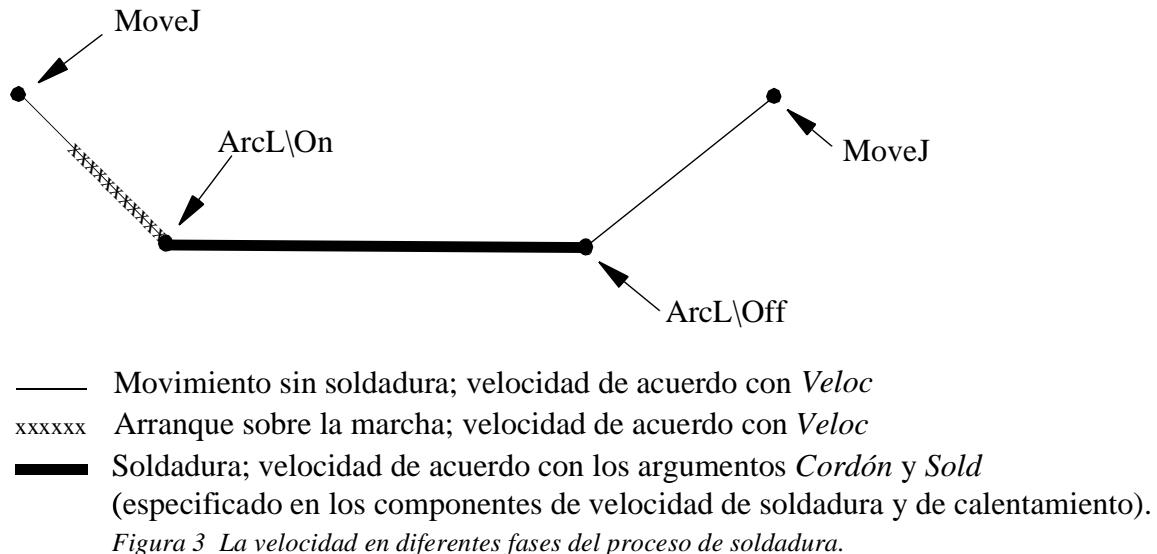
VelocTipo de dato: *speeddata*

La velocidad del TCP está controlada por el argumento *Veloc* en los siguientes casos:

- Cuando se utiliza el argumento *\On* (preparativos del inicio de la soldadura en un arranque sobre la marcha).
- Cuando el programa es ejecutado instrucción por instrucción (sin soldadura).

La velocidad del TCP *durante la soldadura* es la misma que la utilizada para los argumentos *Cordón* y *Sold*. (Véase la Figura 3)

Los datos de velocidad describen también la velocidad de la reorientación de la herramienta y la velocidad de cualquier eje externo no coordinado.

**[\T]**

(Tiempo)

Tipo de dato: *num*

El argumento *\T* sirve para especificar el tiempo total del movimiento determinado en segundos directamente en la instrucción. El tiempo será así sustituido por la velocidad especificada en los argumentos *Veloc*, *Cordón* y *Sold*.

Este argumento puede utilizarse cuando, por ejemplo, uno o más ejes externos no coordinados participan en el movimiento. Los ejes externos no coordinados deberán, no obstante, evitarse ya que, cuando se utilizan, el programa será más difícil de ajustar. En vez de ello, utilíicense ejes externos coordinados. La oscila-

ción es desactivada durante la ejecución de las instrucciones ArcX con argumentos \T.

CordónTipo de dato: *seamdata*

Los datos iniciales y finales de soldadura describen las fases de inicio y finales de un proceso de soldadura.

El argumento *Cordón* está incluido en todas las instrucciones de soldadura al arco, de forma que, independientemente de la posición del robot cuando se interrumpe el proceso, se produce un final y un rearranque de soldadura adecuados.

Normalmente los mismos datos iniciales y finales de soldadura se utilizan en todas las instrucciones de un cordón.

SoldTipo de dato: *welddata*

Los datos de soldadura describen la fase de soldadura de un proceso de soldadura.

Los datos de soldadura a menudo son cambiados de una instrucción a la siguiente durante una soldadura de un cordón.

OscilTipo de dato: *weavedata*

Los datos de oscilación describen la oscilación que debe tener lugar durante las fases de calentamiento y de soldadura. La soldadura sin oscilación se obtiene especificando por ejemplo, el dato de oscilación *nooscil*. (Sin oscilación si el valor del componente *forma_osc* es igual a cero.)

ZonaTipo de dato: *zonedata*

Los datos de zona definen la distancia a la que deben estar los ejes, de la posición programada antes de poder empezar el movimiento hacia la posición siguiente.

En el caso de un punto de paso, se generá una trayectoria esquina más allá que esta posición. En el caso de un punto de paro (*punto fino*), el movimiento es interrumpido hasta que todos los ejes hayan alcanzado el punto programado.

Un punto de paro se genera siempre automáticamente en la posición de inicio de una soldadura (incluso en el caso de un *arranque sobre la marcha*) y en la posición final de soldadura *controlada*. Los puntos de paso, como *z10*, deberán utilizarse para todas las demás posiciones de soldadura.

Los datos de soldadura cambian a la siguiente instrucción de soldadura al arco en el punto central de una trayectoria esquina. (a menos que esté retrasado por el componente *retraso_distancia* del argumento *Sold*).

[\Z]

(Zona)

Tipo de dato: *num*

Este argumento sirve para especificar la precisión de posicionamiento del TCP del robot directamente en la instrucción. El tamaño de la zona está especificada en mm y es sustituida en la zona correspondiente determinada en los datos de zona. El argumento \Z es muy útil también cuando se trata de ajustar trayectorias

esquina individuales.

Herram

Tipo de dato: *tooldata*

Es la herramienta utilizada en el movimiento. El TCP de la herramienta es el punto que se mueve a la posición de destino especificada. El eje z de la herramienta deberá ser paralelo al elemento final.

[\WObj]

(*Objeto de trabajo*)

Tipo de dato: *wobjdata*

Es el objeto de trabajo (sistema de coordenadas) al que se refiere la posición del robot en la instrucción.

Cuando se omite este argumento, la posición del robot se referirá al sistema de coordenadas mundo. No obstante, deberá especificarse, si se utiliza un TCP estacionario o ejes externos coordinados.

\WObj podrá utilizarse si se ha definido un sistema de coordenadas para el objeto correspondiente o para el cordón de soldadura.

Ejecución del programa

Control del equipo de proceso

El equipo de proceso está controlado por el robot de tal forma que todo el proceso y cada una de sus fases están coordinadas con los movimientos del robot.

Movimiento

El robot y los ejes externos se mueven a la posición de destino según lo siguiente:

- El TCP de la herramienta se mueve de forma lineal a una velocidad programada constante. Cuando se utilizan ejes coordinados, se moverán de forma lineal a una velocidad programada constante respecto al objeto de trabajo.
- La herramienta es reorientada a intervalos regulares durante toda su trayectoria.
- Los ejes externos no coordinados se ejecutan a una velocidad constante, lo que significa que alcanzan su destino al mismo tiempo que los ejes del robot.

En el caso en que se exceda la velocidad programada de reorientación o de los ejes externos, estas velocidades serán limitadas, reduciendo la velocidad del TCP.

La posición de destino se refiere:

- al sistema de coordenadas del objeto especificado si se utiliza el argumento \WObj ;
- al sistema de coordenadas mundo si no se utiliza el argumento \WObj .

Limitaciones

En la oscilación, la distancia entre las posiciones programadas deberán ser más largas que el tiempo periódico de la oscilación. Si la distancia es más pequeña y si hay un cambio significativo del ángulo de la trayectoria, la trayectoria de oscilación será distorsionada.

Gestión de errores

El proceso está supervisado por una serie de señales de entrada. Si se detecta algo anormal, la ejecución del programa será detenida. Pero si, no obstante, se programa un gestor de errores, los errores que se definen a continuación podrán ser remediados sin detener el proceso de producción. Véase el ejemplo en la instrucción *RestoPath*.

Constante de errores (valor *ERRNO*)

Descripción

AW_START_ERR	Error de condición de inicio; supervisión de gas, agua o antorcha
AW_IGNI_ERR	Error de encendido; supervisión del arco
AW_WELD_ERR	Error de soldadura; supervisión del arco
AW_EQIP_ERR	Error del equipo de soldadura; supervisión de tensión, de corriente, agua o gas durante la soldadura
AW_WIRE_ERR	Error de hilo; supervisión de hilo
AW_STOP_ERR	Soldadura interrumpida por la entrada de paro del proceso

La supervisión del proceso está determinada como una parte de la configuración del equipo de proceso.

Al *inicio* del proceso, el robot comprueba que se hayan cumplido las siguientes *condiciones preliminares*:

- paro_proceso
- agua_OK
- gas_OK
- antorcha_OK

En el caso en que, después de que el comando de inicio haya sido ejecutado, no se indica ningún perfil de inicio aprobado en la entrada digital, *arc_OK*, dentro de un período de tiempo predeterminado, el inicio del proceso será interrumpido.

Una vez que el proceso haya comenzado, todas las entradas de supervisión seleccionadas serán continuamente monitorizadas:

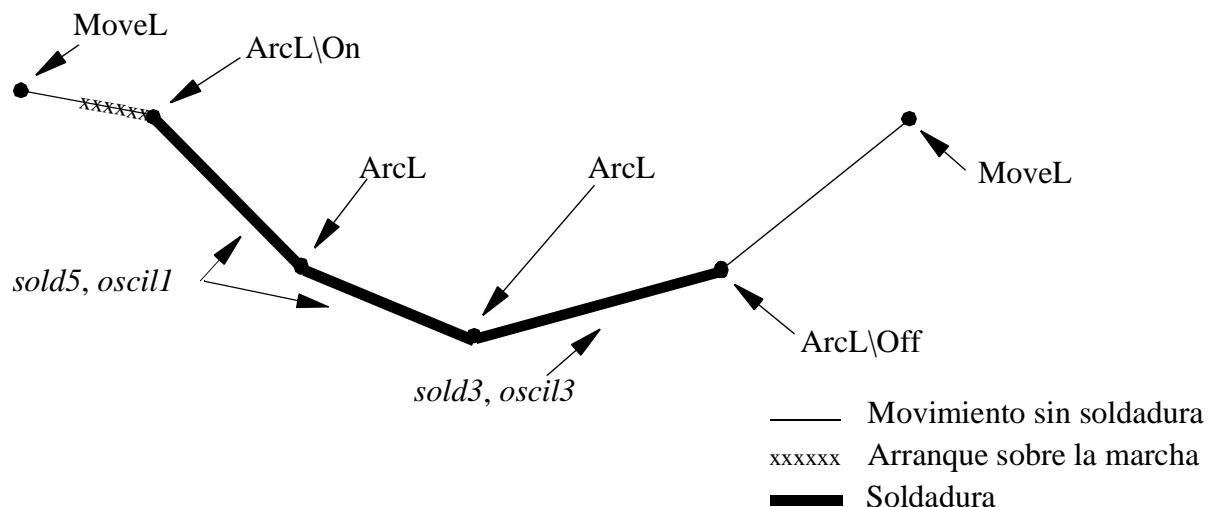
- paro_proceso, agua_OK, gas_OK, arc_OK, tensión_OK, corriente_OK, velhilo_OK.

La supervisión wirestick_err será comprobada al final de la soldadura.

Ejemplo

```
MoveL ...
ArcL \On, *, v100, cordón1, sold5, oscil1, fine, pistola1\Wobj:=wobj1;
ArcL *, v100, cordón1, sold5, oscil1, z10, pistola1\Wobj:=wobj1;
ArcL *, v100, cordón1, sold5, oscil1, z10, pistola1\Wobj:=wobj1;
ArcL \Off, *, v100, cordón1, sold3, oscil3, fine, pistola1\Wobj:=wobj1;
MoveL ...
```

En este ejemplo, se lleva a cabo una soldadura en la que los datos de soldadura y los datos de oscilación son cambiados en la parte final de la soldadura, según se indica en la Figura 4. Observar que una instrucción de soldadura al arco deberá utilizarse para cambiar la dirección de la trayectoria a pesar de que no haya ningún dato de soldadura que se haya cambiado.



En este ejemplo, se presupone que se ha utilizado un eje externo coordinado para el movimiento. En este caso, el objeto de trabajo, *wobj1* deberá estar especificado en la instrucción.

Sintaxis

ArcL

```
[ '\'On',' ] | [ '\'Off',' ]
[ AlPunto ':=' ] < expresión (IN) de robtarget > ',' 
[ Veloc ':=' ] < expresión (IN) de speeddata >
[ ( '\' T ':=' < expresión (IN) de num > ) ] ',' 
[ Cordón ':=' ] < persistente (PERS) de seamdata > ',' 
[ Sold ':=' ] < persistente (PERS) de welddata > ',' 
[ Oscil ':=' ] < persistente (PERS) de weavedata > ',' 
[ Zona ':=' ] < expresión (IN) de zonedata >
[ '\' Z ':=' < expresión (IN) de num > ] ',' 
[ Herram ':=' ] < persistente (PERS) de tooldata >
[ '\' WObj ':=' < persistente (PERS) de wobjdata > ] ';'
```

Información relacionada

Descripción:

Realización de una soldadura circular	Instrucciones - ArcC
Otras instrucciones de posicionamiento	Resumen RAPID - Movimiento
Definición de la velocidad	Tipos de Dato - speeddata
Definición de los datos de zona	Tipos de Dato - zonedata
Definición de las herramientas	Tipos de Dato - tooldata
Definición de los objetos de trabajo	Tipos de Dato - wobjdata
Definición de los datos iniciales y finales de soldadura	Tipos de Dato - seamdata
Definición de los datos de soldadura	Tipos de Dato - welddata
Definición de los datos de oscilación	Tipos de Dato - weavedata
Parámetros de instalación para el equipo y funciones de soldadura	Parámetros del Sistema - Soldadura al arco
Movimientos en general	Principios de Movimiento
Sistemas de coordenadas	Principios de Movimiento - Sistemas de coordenadas
Fases del proceso y subactividades	Resumen RAPID - Soldadura al Arco

ArcKill Aborto del proceso de soldadura al arco

ArcKill se utiliza en gestores de error avanzados para abortar el proceso de soldadura al arco.

Ejemplo

```
PROC main()
    WeldSeam1;
    WeldSeam2;

    ERROR
        TPReadFk ans "The weld failed", "", "", "", "Service", "OK";

        TEST ans
        CASE 4:
            service_routine;
            TRYNEXT;

        DEFAULT:
            TRYNEXT
        ENDTEST

    ENDPROC

    PROC WeldSeam1()
        ! Soldar la costura
        ArcL \On, p1, v100, seam1, weld5, noweave, fine, gun1;
        ArcL \Off, p2, v100, seam1, weld5, noweave, fine, gun1;

        ERROR
            TEST ERRNO
            CASE AW_IGNI_ERR:
                ! Intentar reiniciar el proceso en caso de tratarse de un error de encendido.
                RETRY;

            DEFAULT:
                ! Abortar el proceso de soldadura al arco y pasar el error a la rutina principal
                ArcKill;
                RAISE
            ENDTEST

    ENDPROC
```

Sintaxis

ArcKill ‘;’

Información relacionada

Descripción:

Realización de una soldadura circular

Instrucciones - *ArcC*

Realización de una soldadura lineal

Instrucciones - *ArcL*

ArcRefresh

Restablecer datos de soldadura al arco

ArcRefresh sirve para ajustar los parámetros de proceso de soldadura al arco durante la ejecución del programa.

Ejemplo

```

PROC PulseWeld()
    ! Activación de una interrupción temporizada de 2 Hz
    CONNECT intno1 WITH TuneTrp;
    ITimer ,0.5 ,intno1;

    ! Soldar la costura
    ArcL  \On, p1, v100, seam1, weld5, noweave, fine, gun1;
    ArcL  \Off, p2, v100, seam1, weld5, noweave, fine, gun1;

    IDElete intno1;

ENDPROC

TRAP TuneTrp
    ! Modificar la componente de tensión_sold de los datos de soldadura activos.

    IF HighValueFlag =TRUE THEN
        weld5.weld_voltage := 10;
        HighValueFlag := FALSE;
    ELSE
        weld5.weld_voltage := 15;
        HighValueFlag := TRUE;
    ENDIF

    ! Ordena al control de proceso el restablecimiento de los parámetros de proceso
    ArcRefresh;

ENDTRAP

La tensión de soldadura será cambiada entre 10 y 15 voltios
por la rutina de tratamiento de interrupciones a una frecuencia de dos Hz.

```

Sintaxis

ArcRefresh ‘;’

Información relacionada

Realización de una soldadura circular
Realización de una soldadura lineal
Definición de datos de soldadura
Definición de datos de oscilación
Parámetros de instalación para el equipo de soldadura y funciones de soldadura

Descripción:

Instrucciones - *ArcC*
Instrucciones - *ArcL*
Tipos de datos - *welldata*
Tipos de datos - *weavedata*
Parámetros del Sistema - *Soldadura al Arco*

ÍNDICE

gundata	Dado da pinça de solda a ponto
spotdata	Dado de solda a ponto
SpotL	Solda a ponto com movimento
Módulo do sistema	SWUSER
Módulo do sistema	SWUSRC
Módulo do sistema	SWUSRF
Módulo do sistema	SWTOOL

gundata

Dado da pinça de solda a ponto

Gundata é usado para definir dado específico da pinça de solda ponto, para controlar a pinça de uma maneira otimizada no processo de solda.

Descrição

Gundata é usado em instruções de solda a ponto e tem a seguinte estrutura:

- Número de pares de eletrodos.
 - Número de níveis de pressão.
 - Flag para indicar se um sinal é necessário para testar a precisão do fechamento da pinça.
 - Flag para indicar se um sinal é necessário para testar a precisão na abertura da pinça.
 - Contador para número de soldas feitas em número máximo permitido (um contador com valor máximo para cada par).
 - Tempos de fechamento.
 - Tempos de abertura.

Componentes

nof_tips	<i>(número de eletrodos)</i>	Tipo de dado: <i>num</i>
Número de pares de eletrodos (1 ou 2)		
nof_plevels	<i>(número de níveis de pressão)</i>	Tipo de dado: <i>num</i>
Número de níveis de pressão (1 - 4)		
close_request	<i>(solicitar fechamento)</i>	Tipo de dado: <i>bool</i>
Se o flag é TRUE, o nível de pressão ordenado é testado antes da solda poder ser iniciada.		
open_request	<i>(solicitar abertura)</i>	Tipo de dado: <i>bool</i>
Se o flag é TRUE, o sinal de abertura da pinça é testado antes do próximo movimento ser liberado. O tempo é definido pelo <i>sw_go_timeout</i> . (Veja Dado Pré-definido e Programas- Módulo do Sistema SWUSER)		
Nota. O open_time deve ter terminado antes do teste ser feito (veja abaixo).		
Se o flag é FALSE, o próximo movimento é sempre liberado depois do open_time (veja abaixo).		
tip1_counter		Tipo de dado: <i>num</i>
Contador para o número de soldas feitas como primeiro par de eletrodos. O contador é		

automaticamente incrementado. O solo contador é opcional. O programa do usuário devezerar o contador.

tip2_counter Tipo de dado: *num*

Contador para o número de soldas feitas como o segundo parâmetro dos eletrodos. O contador é automaticamente incrementado. O solo contador é opcional. O programa do usuário devezerar o contador.

tip1_max¹ Data type: *num*

Número máximo de soldas para serem feitas pelo primeiro parâmetro dos eletrodos antes da manutenção dos mesmos serem necessária. Este parâmetro deve ser usado para troca automática dos eletrodos, etc.

tip2_max¹ Tipo de dado: *num*

Valor máximo do número de soldas permitidas como segundo parâmetro dos eletrodos antes da manutenção dos mesmos serem necessária. É usado se deseja o programa do usuário.

close_time1 Tipo de dado: *num*

Tempo [s] para fechar a pinça se a pressão p1 é ativada.

close_time2 Tipo de dado: *num*

Tempo [s] para fechar a pinça se a pressão p2 é ativada.

close_time3 Tipo de dado: *num*

Tempo [s] para fechar a pinça se a pressão p3 é ativada.

close_time4 Tipo de dado: *num*

Tempo [s] para fechar a pinça se a pressão p4 é ativada.

build_up_p1¹ Tipo de dado: *num*

Tempo [s] decorrido para atingir a pressão do momento onde a pinça foi fechada. A pressão p1 tem que ser setada primeiro. Este parâmetro deve ser usado para medir o tempo de fechamento automaticamente.

build_up_p2¹ Tipo de dado: *num*

Tempo [s] decorrido para atingir a pressão onde a pinça foi fechada. A pressão p2 tem que ser setada primeiro.

build_up_p3² Tipo de dado: *num*

Tempo [s] decorrido para atingir a pressão onde a pinça foi fechada. A pressão p3 tem que ser setada primeiro.

1. To be defined only if used by the user program.

2. Para ser definido somente se usado pelo programa do usuário.

build_up_p4²Tipo de dado: *num*

Tempo[s] decorrido para atingir a pressão onde a pinça foi fechada. A pressão op4 tem que ser setada primeiro.

open_timeTipo de dado: *num*

O tempo[s] que sempre decorre entre a ordem de abertura da pinça e a liberação do próximo movimento ou o teste de abertura da pinça (veja acima).

Estrutura

```
< dataobject of gundata>
  <nof_tips of num>
  <nof_plevels of num>
  <close_request of bool>
  <open_request of bool>
  <tip1_counter of num>
  <tip2_counter of num>
  <tip1_max of num>
  <tip2_max of num>
  <close_time1 of num>
  <close_time2 of num>
  <close_time3 of num>
  <close_time4 of num>
  <build_up_p1 of num>
  <build_up_p2 of num>
  <build_up_p3 of num>
  <build_up_p4 of num>
  <open_time of num>
```

Informação relacionada

Descrito em:

Instrução de solda a ponto

Instruções - *SpotL*

Definição da ferramenta TCP, peso etc.

Tipos de dados - *tooldata*

spotdata

Dado de solda a ponto

Spotdata é usado para definir os parâmetros que controlam um temporizador das soldas e pinças de solda para soldar um certo ponto.

Descrição

Spotdata é referido em instruções de solda a ponto e contém dados de como controlar a solda na instrução atual.

Spotdata tem a seguinte estrutura:

- Número do programa no qual o temporizador de solda é usado.
 - Eletrodos a serem ativados (no caso de uma pinça dupla).
 - Pressão da pinça desejada (se conectada).
 - Número do temporizador de solda (se mais que um).

Componentes

prog_no (*número do programa*) Tipo de dado: *num*

Define o programa interno no temporizador de solda para ser usado pela solda.

A faixa permitida é de 0..**sw_prog_max** (definido no módulo do sistema SWU).

10 (número do par de eletrodos) Tipo de dado: num

Define o par de eletrodos a ser ativado. As seguintes alternativas estão disponíveis:

- 1: Par de eletrodos 1

2: Par de eletrodos 2

12: Pares de eletrodos 1 e 2. Ambos os pares estão fechados ao mesmo tempo. A solda é feita na sequência: primeiro par 1, depois par 2. Então, ambos os eletrodos são abertos juntos.

21: Pares de eletrodos 2 e 1. Ambos os pares estão fechados ao mesmo tempo. A solda é feita na sequência: primeiro par 2, depois par 1. Então, ambos os eletrodos são abertos juntos.

Para pinças com somente um par de eletrodos o valor deve ser 1.

gun_pressure (*pressão da pinça*) Tipo de dado: *num*

Define a pressão da pinça usada.

As seguintes alternativas estão d

As seguintes alternativas estão disponíveis. Pressione a opção 1.

timer_no (*número do temporizador de solda*) Tipo de dado: *num*

Define o temporizador de solda a ser usado. Somente é usado para certos tipos de temporizadores de solda.

Se somente um temporizador é usado (caso normal) o valor deve ser 1.

Exemplo

```
PERS spotdata spot1:= [16, 1,4,1];
```

O dado de solda spot1 é programado para um equipamento que contém somente uma pinça com temporizador de solda. Quando spot1 for usado, ocorrerá o seguinte:

- O programa de número 16 está controlando a solda.
- A pressão da pinça tem que alcançar o nível 4 antes da solda poder inicializar.

Estrutura

```
<dataobject of spotdata>
<prog_no of num>
<tip_no of num>
<gun_pressure of num>
<timer_no of num>
```

Informação relacionada

Descrito em:

InSTRUÇÃO DE SOLDA A PONTO

INSTRUÇÕES - SPOTL

DADO DA PINÇA

TIPOS DE DADOS - GUNDATA

SpotL Soldadura por puntos con movimiento

SpotL (SpotLinear) se utiliza en la soldadura por puntos para controlar el movimiento, la apertura/cierre de la pinza y el proceso de soldadura. *SpotL* mueve el TCP linealmente hasta la posición final.

Ejemplo

SpotL p100, vmax, spot10, gun7, tool7;

Esta es la única instrucción necesaria para realizar una operación completa de soldadura.

El TCP para *tool7* es movido siguiendo una trayectoria lineal a la posición *p100* con la velocidad proporcionada en *vmax*. La posición de soldadura es siempre una posición de paro dado que la soldadura se realiza siempre cuando el robot está parado. La pinza se cierra por adelantado conforme se dirige a la posición. La soldadura empieza y será supervisada hasta que se haya acabado y la pinza volverá a abrirse.

Nota El programa continua ejecutándose después de que la soldadura haya empezado y no será bloqueada hasta alcanzar la orden siguiente que contenga un movimiento de robot. Ello podrá ser inhabilitado mediante el interruptor \NoConc (véase a continuación).

El dato de soldadura por puntos *spot10* contiene los parámetros del equipo de soldadura.

El dato de la pinza *gun7* contiene datos de soldadura específicos de la pinza.

Argumentos

**SpotL AlPunto Veloc Punto Sold [\InPos] [\NoConc] [\Retract]
Pinza Herram [\WObj]**

AlPunto

Tipo de dato: *robtarget*

Es el punto de destino del robot y de los ejes externos. Está definido como una posición nombrada o es almacenado directamente en la instrucción (marcado con un asterisco * en la instrucción).

Velocidad

Tipo de dato: *speeddata*

Son los datos de velocidad que se aplican a los movimientos. Los datos de velocidad definen la velocidad del punto central de la herramienta, la reorientación de la herramienta y los ejes externos.

PuntoSold	Tipo de dato: <i>spotdata</i>
------------------	-------------------------------

Son los datos Spot, del punto soldado, asociados al equipo de proceso de soldadura.

[InPos]	Tipo de dato: <i>switch</i>
----------------	-----------------------------

El argumento opcional \InPos inhabilita el semicerre de la pinza. La pinza será cerrada en primer lugar cuando el robot haya alcanzado la posición final. Este argumento aumentará el tiempo de ejecución pero resulta muy útil en situaciones con poco espacio.

[NoConc]	Tipo de dato: <i>switch</i>
-----------------	-----------------------------

El argumento opcional \NoConc impide que el programa continúe su ejecución hasta que haya acabado la soldadura que se está realizando. Debería utilizarse cuando la instrucción siguiente es una instrucción lógica.

[Retract]	Tipo de dato: <i>switch</i>
------------------	-----------------------------

El argumento opcional \Retract hará que la pinza se abra en su apertura total (retract) después de la soldadura. En el caso en que este argumento sea omitido, la pinza se abrirá en su semi-apertura (work).

Pinza	Tipo de dato: <i>gundata</i>
--------------	------------------------------

Son los datos de la herramienta específica de soldadura que corresponden a la pinza que se está utilizando.

Herram	Tipo de dato: <i>tooldata</i>
---------------	-------------------------------

Es la herramienta que se está utilizando cuando se mueve el robot. El punto central de la herramienta es el punto movido a una posición de destino específica, que deberá ser la posición de los electrodos cuando la pinza está cerrada.

[WObj]	Tipo de dato: <i>wobjdata</i>
---------------	-------------------------------

Es el objeto de trabajo (sistema de coordenadas) al que se refiere la posición del robot en la instrucción.

Este argumento podrá ser omitido, y, en tal caso, la posición se referirá al sistema de coordenadas mundo. Por otra parte, si se utiliza un TCP estacionario o ejes externos coordinados, este argumento deberá ser especificado para poder realizar un movimiento lineal respecto al objeto de trabajo.

Personalización de SpotWare

El paquete de programa SpotWare ofrece al usuario una multitud de oportunidades para

personalizar y adaptar a cada instalación específica la instrucción *SpotL*:

- mediante los datos definidos por el usuario, que afectan el comportamiento interno en *SpotL*.
(Véase Programas y Datos Predefinidos - *Módulo del Sistema SWUSRF, SWUSRC*).
- mediante rutinas definidas por el usuario que son llamadas en ciertos puntos predefinidos de soldadura en la secuencia interna. (Véase Programas y Datos Predefinidos - *Módulo del Sistema SWUSRF, SWUSRC*).
- mediante el cambio de la configuración de E/S. (Véase Parámetros del Sistema-Soldadura por puntos).

Sin embargo, el cuerpo principal de esta instrucción *SpotL* está indicada en la configuración por defecto de la instalación.

Ejecución del programa

A continuación se indica la secuencia interna contenida en una instrucción *SpotL*:

- La pinza empieza a moverse hacia la posición.
- El número de programa de soldadura está activado para el temporizador externo de soldadura.
- La presión de la pinza está activada.
- La supervisión de la presoldadura se ha llevado a cabo.
- La pinza empieza a cerrarse en un momento definido antes de alcanzar la posición (en el caso en que el argumento \InPos no haya sido utilizado).
- La pinza se cierra cuando ha alcanzado la posición.
- Se espera la señal OK referente a la presión alcanzada.
- El contador de soldadura ha sido incrementado.
- El movimiento hacia la posición siguiente ha sido bloqueado.
- La señal de arranque ha sido enviada al temporizador de soldadura.
- La ejecución del programa continua a la siguiente instrucción de movimiento (en el caso en que el argumento \NoConc haya sido utilizado).
- Cuando la señal preparada del temporizador ha sido alcanzada, la pinza se abre.
- El movimiento es arrancado después de haber recibido una señal correspondiente o cierto tiempo después de la apertura de la pinza y después de que la supervisión de postsoldadura haya sido satisfactoria.

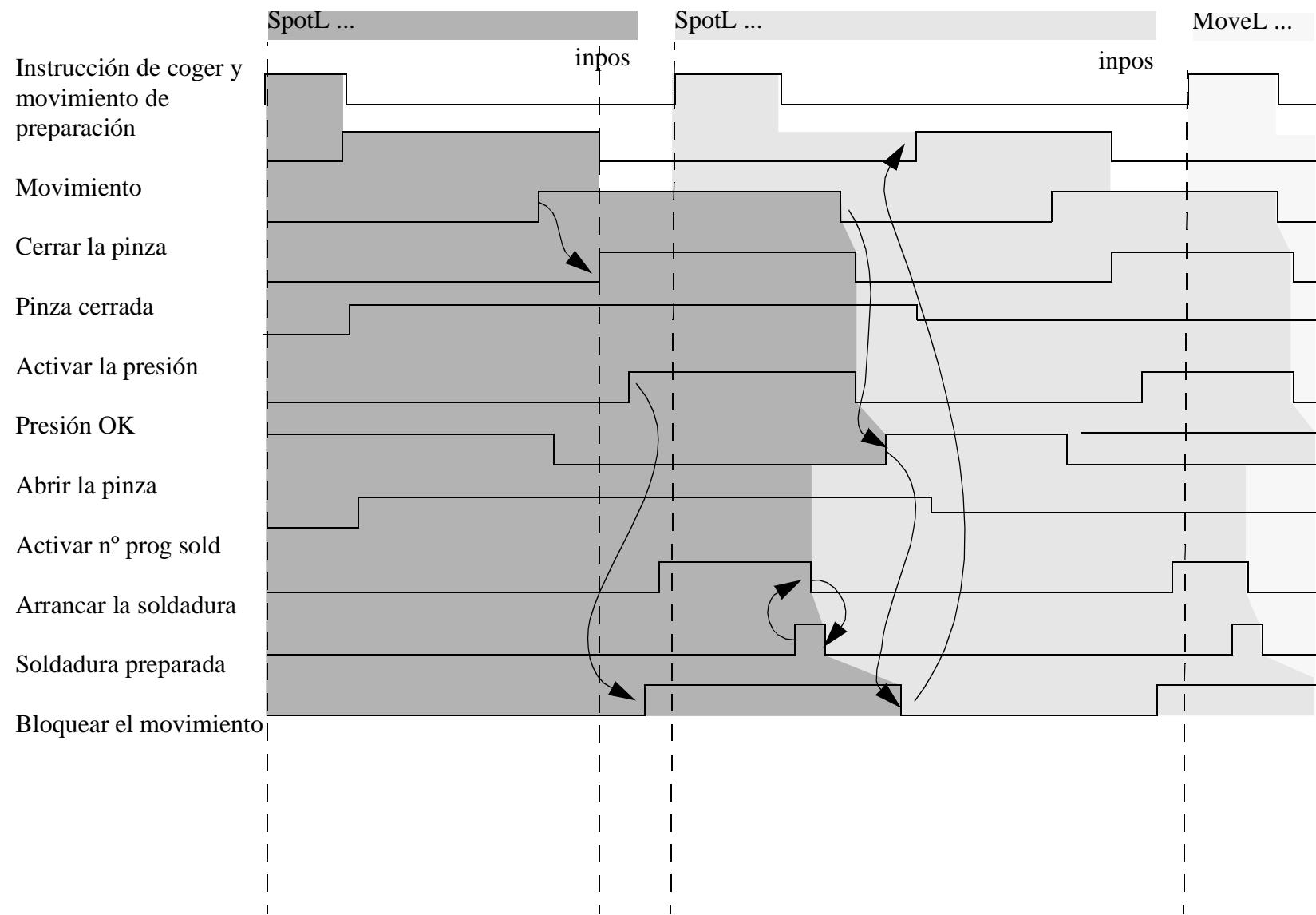


Figura 1 Secuencia de una soldadura por puntos.

Cerrado de la pinza

El cerrado de la pinza se activa en un momento dado definido antes de alcanzar la posición de soldadura, independientemente de la velocidad utilizada. La pinza debe estar cerrada cuando la pinza alcanza la posición final. El tiempo de cerrado de la pinza se encuentra especificado en los datos de pinza. Se refiere al tiempo necesario para cerrar la pinza a partir de la carrera de soldadura y dependerá de la presión seleccionada de la pinza (especificada en los datos de soldadura por puntos spot).

En los datos spot de soldadura por puntos, se podrá escoger entre un máximo de cuatro presiones de pinza. La presión de la pinza se activa tan pronto como empieza el movimiento hacia la posición. En el caso en que lo requieran los datos de pinza, el sistema realizará una comprobación para ver si la presión de la pinza ha sido alcanzada o no.

Cuando la pinza se está cerrando, si se desea, *work_select* podrá ser activado en 1. Véase Programas y Datos Predefinidos.

Soldadura

La señal de arranque es enviada al temporizador tan pronto como el robot haya alcanzado la posición final. Antes de que la señal de arranque sea enviada hay toda una serie de señales de supervisión que deberán ser recibidas. Se trata de rutinas definidas por el usuario. Véase Programas y Datos Predefinidos.

La señal de arranque estará activada durante todo el periodo que dura la soldadura. Será reinicializada después de la señal de soldadura preparada o después de la señal de tiempo de soldadura excedido.

Cuando se usa la opción para temporizadores disparados por el programa, el número de programa será activado con la señal de arranque y será reinicializado a cero después de una soldadura terminada o después del tiempo de soldadura excedido.

Apertura de la pinza

Dependiendo del parámetro \Retract la pinza puede tener una apertura total y una semi apertura, después de haber acabado la soldadura. La apertura es siempre supervisada, de forma que se espera una señal de apertura de la pinza. Esto permitirá el inicio del movimiento siguiente.

Asimismo, la pinza se abrirá después de un error de soldadura y también en otras situaciones de error.

Movimiento

Para ganar tiempo, el indicador de punto de arranque del programa se desplaza a la instrucción de movimiento siguiente, mientras el robot está todavía realizando la soldadura. Esto posibilita que el movimiento siguiente pueda arrancar inmediatamente tras haber recibido la orden.

Esto significa que las instrucciones que se encuentran después de la instrucción *SpotL*, que no sean de movimiento, como las instrucciones lógicas, puedan llevarse a cabo de

forma concurrente con la soldadura. En el caso en que no se desee esto, se deberá utilizar el parámetro *NoConc*.

Se podrá configurar una habilitación de movimiento para el movimiento siguiente que será una señal a partir de la pinza o cierto tiempo después de la señal soldadura preparada.

La posición final se refiere al sistema de coordenadas del objeto en *WObj*.

Personalización

Existe una amplia gama de herramientas para la personalización del paquete SpotWare, ofreciendo rutinas RAPID al usuario. Existen rutinas de base implementadas en el software de base. Las siguientes funciones podrán ser fácilmente programadas por el usuario:

Supervisión de presoldadura - *sw_preweld_sup*

Supervisión de postsoldadura - *sw_postweld_sup*

Cerrado de la pinza - *sw_close_gun*

Apertura de la pinza - *sw_open_gun*

Determinación de la presión - *sw_set_pressure*

Cálculo del tiempo de semicerre - *sw_close_time*

Recuperación de errores - *sw_error_recover*

Observar que el código por defecto tiene el mismo comportamiento que el descrito anteriormente para SpotWare.

Véase Programas y Datos Predefinidos - *Módulos del Sistema SWUSRC*.

Secuencia interna: La señal de arranque podrá ser enviada al temporizador inmediatamente - si ha sido configurada de esta forma - después de que el sistema haya recibido la supervisión de la soldadura previa. Véase Programas y Datos Predefinidos - *Módulo del Sistema SWUSRC*.

Arranque y paro del programa

Paro durante el movimiento y rearranque

El robot se detiene en la trayectoria. En el caso en que la señal de cierre de la pinza ya haya sido enviada, la pinza se volverá a abrir en cuanto el robot esté parado.

Al rearanque, el robot continua hacia la posición programada, vuelve a cerrar la pinza, y lleva a cabo normalmente la secuencia contenida en la instrucción *SpotL*.

Paro durante la soldadura y rearranque

La soldadura está acabada. La validación de la soldadura se realiza después del paro y la pinza se abre. Si no se ha utilizado \NoConc, el indicador de programa dejará la instrucción actual y apuntará a la siguiente instrucción que contenga un movimiento de robot.

Ejecución instrucción por instrucción

Hacia adelante

El movimiento y la soldadura han sido ejecutados.

Hacia atrás

Dependiendo de la instrucción \Retract, la pinza está activada en una posición de apertura total o semi apertura.
El movimiento se ejecuta hacia atrás.

Soldadura simulada

Simulación total de un temporizador

Se activa colocando *sw_sim_weld* en TRUE. Ello tendrá por efecto la inhabilitación de la señal de arranque para el temporizador. El tiempo de simulación está especificado en *sw_sim_time*. No se realiza ninguna supervisión previa de la soldadura.

Simulación en el temporizador

Se activa colocando *sw_inhib_weld* en TRUE. Ello tendrá por efecto la desactivación de la señal *current_enable* para la soldadura siguiente. No se realiza ninguna supervisión previa de la soldadura.

Gestión de errores

Eventos en una situación de error

Cuando la instrucción *SpotL* ha sido parada por una supervisión, ocurrirá lo siguiente:

- Se activa la señal *process_error*.
- Aparece un mensaje de error en el visualizador.
- El mensaje de error es registrado en una lista.

Situaciones de error

Pueden ocurrir las siguientes situaciones de error:

- Error de parámetro de instrucción.
- Error de supervisión antes de la soldadura.
- Error de soldadura.
- Error de supervisión después de la soldadura y antes del movimiento siguiente.
- Error de supervisión para la presión de la pinza.
- Error de supervisión para el cierre de la pinza (SpotWare Plus).
- Error de supervisión para la apertura de la pinza.

Todas las actividades de supervisión descritas anteriormente, excepto el error de soldadura y el error de parámetro, podrán ser modificadas por el usuario.

Para parar la ejecución con un mensaje de error, bastará con introducir el mensaje deseado en la cadena de error asignada y aparecerá en la unidad de programación.

La acción que se describe a continuación ha sido programada por defecto.

Véase también Programas y Datos Predefinidos.

Error de parámetro de instrucción

El error ocurre cuando se llama la instrucción *SpotL* que contiene parámetros incorrectos. El programa se detiene.

El parámetro deberá ser cambiado y la instrucción correspondiente deberá ser rearrancada desde el principio.

Supervisión antes de la soldadura

Las supervisiones contenidas en *sw_preweld_sup* se están llevando a cabo. Véase Programas y Datos Predefinidos - *Módulo del Sistema SWUSRC*.

Error de soldadura

Un error de soldadura ocurre cuando por ejemplo la señal preparado del temporizador de soldadura no ha sido activada a un momento determinado (*sw_wr_timeout*). La instrucción *SpotL* podrá ser configurada para volver a soldar automáticamente un cierto número de veces antes de que aparezca visualizado el error y que se detenga la ejecución del programa, en la espera de una actividad manual.

- La señal de arranque está desactivada.
- La pinza se abre.
- La señal de error de proceso está activada.
- Se ofrecen las siguientes alternativas manuales:

Modo automático: **Servicio / Resoldar**

Modo manual: **Servicio / Saltar / Resoldar** (véase la ventana de diálogo de la Figura 2).

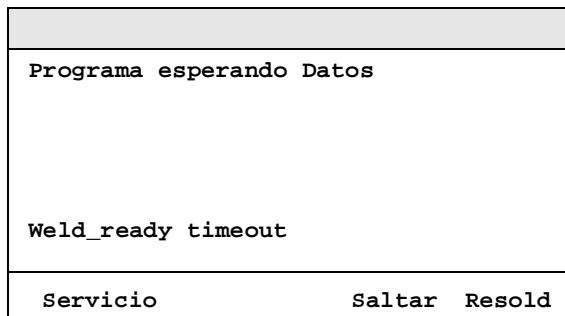


Figura 2 La ventana de diálogo para un error de soldadura.

Rearranque seleccionando la opción Servicio

Esta opción no podrá utilizarse en el modo de ejecución “stepwise forward”, ejecución paso a paso hacia adelante.

- La señal de error de proceso es reinicializada.
- La rutina definida por el usuario *sw_service_wf* se está ejecutando, por ejemplo moviéndose a una posición de inicio.
- El robot ha regresado al punto de soldadura por puntos.
- Regreso a la ventana de diálogo como se indica en la Figura 2.

Rearranque seleccionando la opción Saltar

- La señal *reset_fault* ha sido pulsada.
- La señal de error de proceso es reinicializada.
- La ejecución del programa continúa omitiendo la soldadura incorrecta.

Rearranque seleccionando la acción manual *SwRunProc*

- El mismo resultado que con la opción *Resoldar*.

Rearranque seleccionando la opción *Resoldar*

- La señal *reset_fault* ha sido pulsada.
- La señal de error de proceso ha sido reinicializada.
- La pinza se cierra.
- La señal de arranque es activada con un retraso de tiempo de *sw_reset_time2* y la ejecución del programa es reanudada.

Supervisión después de la soldadura

Las supervisiones contenidas en la rutina definida por el usuario *sw_postweld_sup* se están ejecutando. Véase Programas y Datos Predefinidos - *Módulo del Sistema SWUSRC*.

Error de cierre de la pinza

El error ocurre cuando no se alcanza la presión determinada después de un tiempo específico. La señal *p1_ok* es supervisada si *gun_pressure=1* en los datos de soldadura por puntos, spotdata etc.

- La señal de error de proceso está activada.
- Se presentará la siguiente alternativa para seleccionar manualmente: **Servicio / Reintento**
- Seleccionar **Servicio**: la rutina definida por el usuario *sw_service_cg* se ejecuta.
- Regreso a la alternativa para seleccionar manualmente: **Servicio / Reintento**.
- Seleccionar **Reintento**: La pinza se cierra y el sistema volverá a comprobar la presión.

Error de apertura de la pinza

El error ocurre cuando la pinza no se ha abierto después de cierto tiempo. La señal *tip1_open* o *tip1_open* y *tip2_open* si *nof_tips=2* contenida en los datos de la pinza.

- La señal de error de proceso está activada.
- Se presenta la siguiente alternativa para seleccionar manualmente: **Servicio / Reintento**
- Seleccionar **Servicio**: la rutina definida por el usuario *sw_service_og* se ejecuta.
- Regreso a la ventana de diálogo según se indica en la Figura 2.
- Seleccionar **Reintento**: la pinza se está abriendo y las señales son comprobadas de nuevo por el sistema.

Recuperación de error definida por el usuario

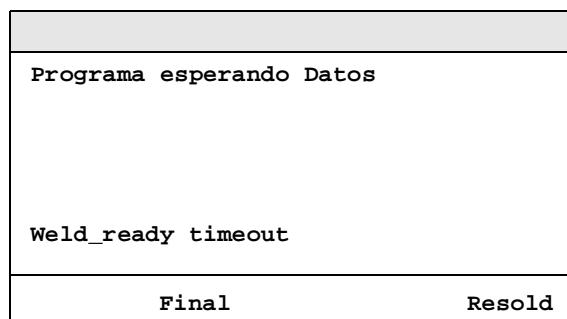
Cuando la recuperación de error definida por el usuario está activada, y que se produce alguno de los errores anteriores, excepto cuando se trata de errores de parámetros, el sistema llama la rutina *sw_user_recover*. Los parámetros de entrada comportan información acerca del caso de error y del texto de error seleccionado.

Esta rutina permite la personalización de la respuesta de la manipulación del error, es decir, el esquema de la unidad de programación y como proseguir. Véase Datos y Programas Predefinidos - *Módulo del Sistema, SWUSRC*.

Nota: El sistema SpotWare no lleva a cabo ningún registro de errores estándar cuando se usa esta opción.

Recuperación de error cuando ejemplo programa 0 ha sido detenido

Cuando la tarea de programa 0 ha sido detenida (programa usuario parado/ejecución paso a paso/acción manual) la acción de servicio no será seleccionable ya que contiene movimientos del robot. En vez de ello, se podrá seleccionar “Final”, que tendrá por efecto abortar el proceso actual.



Acciones manuales

Las acciones manuales permiten la ejecución de funciones de soldadura por puntos sin tener que programar una instrucción SpotL. Podrá utilizarse como una herramienta para comprobar el código definido por el usuario antes de la programación de la línea de programa.

Las acciones manuales ejecutan el mismo código de proceso que cuando se ejecuta SpotL, es decir, que proporciona una recuperación total de errores.

Las acciones manuales se ejecutan en cualquier estado del sistema excepto cuando una instrucción SpotL se está ejecutando. Las acciones manuales son activadas por salidas

virtuales asignadas.

Ver Parámetros del Sistema - *Soldadura por Puntos*.

Comunicación

La instrucción *SpotL* comunica con su equipo utilizando señales paralelas.

Para una descripción completa de la configuración de las E/S, véase Parámetros del Sistema - *Soldadura por Puntos*.

Algunos temporizadores de soldadura con interface serie son aceptados. En tales casos el interface paralelo de *SpotL* sigue siendo válido y los mensajes de comunicación serie aparecen visualizados en un mapa interno. Véase información separada.

Manipulación del corte de potencia

Al rearanque del sistema después de un corte de potencia, ocurrirá que:

- todas las señales de salida de la soldadura por puntos se activan en el estado anterior así como las señales de cierre de la pinza y las señales de arranque de la soldadura.

Al rearanque del programa después de un corte de potencia, ocurrirá que:

- el robot regresa a la trayectoria y prosigue con la ejecución del programa interrumpido.
- en el caso en que el corte de potencia se haya producido cuando un proceso de soldadura estaba activo, el punto de soldadura actual volverá a ser automáticamente soldado.

Sintaxis

SpotL

```
[ AlPunto'=' ] < expresión (IN) de robtarget > ','  
[ Veloc'=' ] < expresión (IN) de speeddata > ','  
[ SoldadPuntos'=' ] < persistente (PERS) de spotdata >  
[ '\' InPos ]  
[ '\' NoConc ]  
[ '\' Retract] ','  
[ Pinza'=' ] < persistente (PERS) de gundata > ','  
[ Herram'=' ] < persistente (PERS) de tooldata >  
[ '\' WObj '=' < persistente (PERS) de wobjdata > ] ';
```

Información relacionada

Otras instrucciones de posicionamiento
Definición de la velocidad
Definición de los datos de zona
Definición de la herramienta
Definición de los objetos de trabajo
Definición de los datos de soldadura por puntos
Definición de los datos de pinza
Características de la soldadura por puntos
Herramientas para la personalización
Configuración de E/S
Movimiento en general
Sistemas de coordenadas

Descripción:

Resumen RAPID - *Movimiento*
Tipos de datos - *speeddata*
Tipos de datos - *zonedata*
Tipos de datos - *tooldata*
Tipos de datos - *wobjdata*

Tipos de datos - *spotdata*
Tipos de datos - *gundata*
Resumen RAPID - *Soldadura por puntos*
Programas y Datos Predefinidos - *Módulo del Sistema SwUsrF, SwUsrC*
Parámetros del sistema - *Soldadura por puntos*
Principios de Movimiento y de E/S
Principios de Movimiento y de E/S - *Sistemas de coordenadas*

Módulo del Sistema

SWUSRC

El módulo del sistema SWUSRC contiene datos y rutinas destinados a ser utilizados para la personalización de la aplicación de soldadura por puntos SpotWare. Dichos datos y rutinas son utilizados y ejecutados normalmente por todas las aplicaciones RAPID de SpotWare.

Nota: Cualquier cambio de rutina llevado a cabo en SWUSRC requerirá los tres pasos siguientes para su aplicación (véase también Parámetros del Sistema Multitarea):

- **salvar en memoria ram**
- **tocar la configuración multitarea**
- **un ARRANQUE EN CALIENTE**

Los nombres están predefinidos y son utilizados a nivel interno cuando se usa una instrucción *SpotL*. Por esta razón no deberán ser cambiados.

Contenido

Datos

A continuación se indican los datos globales predefinidos:

Nombre	Declaración	Descripción
sw_weld_counter	PERS num sw_weld_counter := 0	Contador de puntos de soldadura. El contador será automáticamente actualizado en la instrucción <i>SpotL</i> .
sw_gp_timeout	PERS num sw_gp_timeout := 2	Tiempo excedido de la presión de la pinza [s]
sw_wr_timeout	PERS num sw_wr_timeout := 2	Tiempo excedido de soldadura lista [s]
sw_go_timeout	PERS num sw_go_timeout := 2	Tiempo excedido de la apertura de la pinza [s]
sw_prog_max	PERS num sw_prog_max := 63	Valor máximo para n°_prog en los datos de soldadura por puntos (<i>spotdata</i>).
sw_inhib_weld	PERS bool sw_inhib_weld := FALSE	Bandera para activar el temporizador de soldadura en el modo de simulación. Si TRUE: La señal habilitar_corriente (<i>current_enable</i>) queda borrada cuando se ejecuta la siguiente instrucción <i>SpotL</i> . Si FALSE: La señal habilitar_corriente (<i>current_enable</i>) queda activada cuando se ejecuta la siguiente instrucción <i>SpotL</i> .

Nombre	Declaración	Descripción
sw_sim_weld	PERS bool sw_sim_weld := FALSE	Bandera para simular a nivel interno el temporizador de soldadura. Si TRUE: La señal de arranque de soldadura no está activada.
sw_spot_id_req	PERS bool sw_spot_id_req := FALSE	Bandera para reclamar la identidad de puntos de soldadura de los puntos programados. Si TRUE: Se podrá coger la identidad del punto correspondiente (el <i>nombre</i> del parámetro spot del dato de soldadura correspondiente) utilizando la función SwGetCurrSpotID. Si se está utilizando el enlace serie para Bosch PSS 5000, los programas de soldadura serán reclamados mediante el <i>nombre</i> de los datos de soldadura por puntos, en lugar de utilizar el número de programa.
sw_sim_time	PERS num sw_sim_time := 0.5	Temporizador de soldadura simulado si sw_sim_weld = TRUE.
sw_aut_reweld	PERS num sw_aut_reweld := 0	Número de reintentos automáticos para volver a soldar después de haber ocurrido la señal de tiempo excedido de soldadura lista (weld_ready timeout).
sw_parity	PERS num sw_parity:= 0	0 = ninguna, 1 = impar, 2 = par
sw_reset_time1	PERS num sw_reset_time1 := 0.5	Reiniciar el pulso [s]
sw_reset_time2	PERS num sw_reset_time2 := 0.5	Tiempo de espera cada vez después de haber cambiado habilitar_corriente (current_enable) y después del pulso de reinicialización [s].
sw_async_weld	PERS num sw_async_weld := FALSE	TRUE significa que el arranque de la soldadura ha sido determinado independientemente de inpos, es decir, inmediatamente después del cierre de la pinza.
sw_servo_corr	PERS num sw_servo_corr :=0,061	Tiempo de corrección para retraso interno. Este valor afecta el tiempo de semicerre. El tiempo correcto que depende de la configuración del movimiento es automáticamente determinado a la puesta en marcha.

Nombre	Declaración	Descripción
sw_start_type	PERS num sw_start_type:=0	Comunica al sistema como debe arrancar el proceso de soldadura en el temporizador: 0: El arranque del proceso de soldadura es disparado por una señal de arranque específica (por ejemplo, start1). El número de programa está previamente determinado. 1: El proceso de soldadura es disparado por las salidas de los números de programa. No se precisa determinar previamente el número de programa.
sw_user_recover	PERS num sw_user_recover := FALSE	TRUE significa que la función del usuario <i>sw_error_recover</i> es llamada en caso de un error de proceso. SpotWare no lleva a cabo ningún registro de errores. FALSE significa que se utiliza la recuperación de errores estándar.

Los siguientes datos predefinidos se usan en las acciones manuales para trabajar en:

<u>Nombre</u>	<u>Declaración</u>
sw_man_gun	PERS gundata gun1:=[1, 4, TRUE, TRUE, 0, 0, 20000, 20000, 0.050, 0.050, 0.050, 0.050, 0, 0, 0, 0, 0]
sw_man_spot	PERS spotdata spot1 := [1, 1, 1, 1];
sw_man_retr	PERS num sw_man_retr:=0;

Rutinas

Las siguientes rutinas predefinidas están instaladas con la aplicación. Todas ellas son utilizadas por la instrucción *SpotL*.

Estas rutinas tienen atribuidas una serie de funciones por defecto pero podrán ser fácilmente cambiadas.

Rutinas de supervisión definidas por el usuario para SpotL:

Las siguientes rutinas son llamadas por la secuencia SpotL.

PROC sw_preweld_sup ()

supervisión de la presoldadura en la soldadura por puntos

La rutina es ejecutada antes de cada inicio del proceso de soldadura.

En el caso en que se haya asignado un texto de error a la cadena de texto errtext, entonces el sistema será automáticamente detenido y aparecerá el mensaje de error en el visualizador con una posibilidad de rearranque de la rutina de supervisión y de reanudar la ejecución del programa.

Funciones atribuidas por defecto:

En el caso en que la soldadura simulada o que las funciones de inhabilitación de la soldadura no estén activadas, entonces se comprobarán las siguientes señales de entrada digital:

timer_ready
flow_ok
temp_ok
current_ok

En el caso de un error, un mensaje de error adecuado será asignado a la cadena de texto errtext.

En el caso en que timer_ready = 0, entonces se producirá un intento de reinicialización del temporizador y una comprobación de la entrada, antes de que el error aparezca indicado.

PROC sw_postweld_sup ()

supervisión de la postsoldadura en la soldadura por puntos

La rutina es ejecutada después del proceso de soldadura y antes de que se realice el movimiento a la posición siguiente.

En el caso en que se haya asignado un texto de error a la cadena de texto errtext, entonces el sistema será automáticamente detenido y aparecerá el mensaje de error en el visualizador con una posibilidad de rearranque de la rutina de supervisión y de reanudar la ejecución del programa.

Funciones atribuidas por defecto:

La señal work_select estará activada o reinicializada de acuerdo con la posición del interruptor Retract (apertura total de la pinza) en la instrucción.

Rutinas de supervisión independientes definidas por el usuario:

Las siguientes rutinas son llamadas independientemente de la secuencia SpotL.

PROC sw_sup_init ()

iniciar supervisión de soldadura por puntos

La rutina es ejecutada al arranque en caliente. Aquí los números de interrupción del usuario utilizados por la rutina de tratamiento de interrupciones sw_sup_trap deberán ser inicializados.

Funciones atribuidas por defecto:

Conexión y activación de las señales e interrupciones utilizadas por la rutina sw_sup_trap.

PROC sw_motor_on ()

motor activado para la soldadura por puntos

Rutina llamada por la rutina de tratamiento de interrupciones sw_sup_trap.

Funciones atribuidas por defecto:

Conexión al número de interrupción imotor_on. Activar las salidas weld_power y pulse_water_on.

PROC sw_motor_off ()

motor desactivado para la soldadura por puntos

Rutina llamada por la rutina de tratamiento de interrupciones sw_sup_trap.

Funciones atribuidas por defecto:

Conexión al número de interrupción imotor_off. Reinicializar la salida weld_power.

PROC sw_proc_ok ()

proceso de soldadura por puntos ok

Rutina llamada por la rutina de tratamiento de interrupciones sw_sup_trap.

Funciones atribuidas por defecto:

Conexión al número de interrupción iproc_ok. Activar las salidas weld_power y pulse_water_on.

PROC sw_proc_error ()

error de proceso en soldadura por puntos

Rutina llamada por la rutina de tratamiento de interrupciones *sw_sup_trap*.

Funciones atribuidas por defecto:

Conexión al número de interrupción iproc_error. Reinicializar la salida weld_power.

PROC sw_sup_curren ()

habilitar corriente de supervisión en soldadura por puntos

Rutina llamada por la rutina de tratamiento de interrupciones *sw_sup_trap*.

Funciones atribuidas por defecto:

Conexión al número de interrupción icurr_enable. Activar las salidas weld_power y pulse water_on.

PROC sw_sup_currdis ()

inhabilitar corriente de supervisión en soldadura por puntos

Rutina llamada por la rutina de tratamiento de interrupciones *sw_sup_trap*.

Funciones atribuidas por defecto:

Conexión al número de interrupción icurr_disable. Reinicializar la salida weld_power.

Rutinas de tratamiento de interrupciones definidas por el usuario:

Las interrupciones en estas rutinas de tratamiento de interrupciones estarán siempre preparadas para la ejecución independientemente del estado actual del sistema.

TRAP sw_sup_trap ()

Rutina de tratamiento de interrupción de la supervisión en la soldadura por puntos

Rutina de tratamiento de interrupciones conectada a la supervisión.

Funciones atribuidas por defecto:

Control de las señales de salida water_start y weld_power dependiendo del estado del sistema. Referirse a Parámetros del Sistema - *Soldadura por puntos*.

Rutinas de proceso y funciones definidas por el usuario para la instrucción SpotL:

Las siguientes rutinas son llamadas por la secuencia SpotL.

PROC sw_close_gun

cierre de la pinza en la soldadura por puntos

La rutina es ejecutada cada vez que se ordena un cierre de la pinza. Nota: También para las acciones manuales.

Si un texto de error es asignado a la cadena de texto errtext_close, entonces el sistema será automáticamente detenido con el texto de error en el visualizador y con una posibilidad de rearrancar la rutina de supervisión y de continuar la ejecución del programa.

Funciones atribuidas por defecto:
Referirse a Instrucciones - *SpotL*.

PROC *sw_open_gun* (*num context*)

apertura de la pinza en la soldadura por puntos

La rutina es ejecutada cada vez que se ordena una apertura de una pinza. Nota:
También para las acciones manuales.

Parámetros:

- contexto: razón de la apertura. Un valor negativo sirve para dejar de lado los tests de apertura de la pinza e ignorar el mensaje errtext_open cuando se requiere una apertura instantánea en una situación de error.

Si un texto de error es asignado a la cadena de texto errtext_close, entonces el sistema será automáticamente detenido con el texto de error en el visualizador y con una posibilidad de rearrancar la rutina de supervisión y de continuar la ejecución del programa.

Funciones atribuidas por defecto:
Referirse a Instrucciones - *SpotL*.

PROC *sw_set_pressure* ()

activación de la presión en la soldadura por puntos

La rutina es ejecutada durante la fase de preparación del proceso de soldadura por puntos.

Funciones atribuidas por defecto:
Activación del grupo de salida de presión de acuerdo con el parámetro presión_pinza.

Referirse a Instrucciones - *SpotL*.

FUNC *num sw_error_recover* (*num error_type, string err_text*)

recuperación de errores de soldadura por puntos

La función es llamada en una situación de error siempre y cuando la variable *sw_user_recover* está activada en TRUE.

Parámetros:

- **error_type**: el tipo de error que ha ocurrido. Se pueden dar los siguientes casos:
 - SW_WELD_ERR: tiempo excedido error de soldadura
 - SW_OG_ERR: error de apertura de pinza. Error registrado por *sw_open_gun*
 - SW_CG_ERR: error de cierre de pinza. Error registrado por *sw_close_gun*
 - SW_PRESUP-ERR: error de supervisión de presoldadura. Error registrado por *sw_preweld_sup*
 - SW_POSUP_ERR: error de supervisión de postsoldadura. Error registrado por *sw_postweld_sup*
- **error_text**: cadena de texto retornada por la función que ha registrado el error. En caso de un error de soldadura, aparecerá el mensaje estándar “weld ready timeout”.

El valor de retorno de esta función define como SpotWare debe continuar después de haber ocurrido el error. Hay tres valores de retorno posibles:

- SW_RETRY: la acción que ha producido el error se vuelve a ejecutar.
- SW_CANCEL: el proceso de soldadura por puntos que se está realizando es abandonado.
- SW_SERVICE: la rutina de servicio correspondiente al error producido se está ejecutando. Observar que no existen rutinas de servicio para la supervisión de presoldadura y postsoldadura.

Módulo del sistema

SWUSRF

El módulo del sistema SWUSRF contiene datos y rutinas destinados a ser utilizados para la personalización de la aplicación de soldadura por puntos SpotWare. Dichos datos y rutinas son utilizados y ejecutados por la función RAPID de primer plano de SpotWare.

Los nombres de las rutinas están predefinidos y son utilizados a nivel interno cuando se usa una instrucción *SpotL*. Por esta razón no deberán ser cambiados.

Contenido

Datos

A continuación se indican los datos globales predefinidos:

<u>Nombre</u>	<u>Declaración</u>	<u>Descripción</u>
sw_inpos	PERS bool sw_inpos := FALSE	Si TRUE: El semicerre de la pinza ha sido desactivado
sw_close_corr	VAR num sw_close_corr := 100	El factor de corrección utilizado cuando el tiempo de semicerre de la pinza ha sido calculado [%]. Será automáticamente reinicializado a 100 después de cada cálculo de semicerre. Es preferible que este dato sea cambiado en una instrucción de asignación separada justo antes de la instrucción <i>SpotL</i> implicada.

Normalmente, se recomienda utilizar el módulo SWUSRF para almacenar variables de los tipos de datos de soldadura (*spotdata*), datos de pinza (*gundata*) y datos de la herramienta (*tooldata*), utilizados para la aplicación. Las siguientes variables son predefinidas pero sus nombres y valores podrán ser cambiados libremente.

<u>Nombre</u>	<u>Declaración</u>
gun1	PERS gundata gun1 := [1, 4, TRUE, TRUE, 0, 0, 20000, 20000, 0.050, 0.050, 0.050, 0.050, 0, 0, 0, 0]
toolg1	PERS tooldata toolg1 := [TRUE, [[0, 0, 0], [1, 0, 0, 0]], [-1, [0, 0, 0], [1, 0, 0, 0], 0, 0, 0]]
spot1	PERS spotdata spot1 := [1, 1, 1, 1]

Rutinas

Las siguientes rutinas predefinidas están instaladas con la aplicación. Todas ellas son utilizadas por la instrucción *SpotL*.

Estas rutinas tienen atribuidas una serie de funciones por defecto pero podrán ser fácilmente cambiadas.

Rutinas de servicio definidas por el usuario:

Después de la ejecución de una rutina de servicio, el robot se moverá a la posición de soldadura interrumpida utilizando una velocidad reducida. Aparecerá visualizado el texto «Rutina de servicio lista» y se indicarán las antiguas teclas de función.

PROC sw_service_cg();

cierre de la pinza_servicio_soldadura por puntos

La rutina es ejecutada cuando se produce el error de pinza «Presión de pinza no alcanzado» y que se pulsa la tecla de función «Servicio».

Apretando la función Reintentó de la rutina de servicio, se producirá un nuevo intento de cierre de la pinza y a continuación la ejecución del programa continuará.

Funciones atribuidas por defecto:

Ninguna función está atribuida, sólo aparecerá visualizado un texto de información y se indicará la tecla «Retorno». Para finalizar la rutina de servicio, se deberá apretar la tecla «Retorno».

PROC sw_service_og();

apertura de la pinza_servicio_soldadura por puntos

La rutina es ejecutada cuando se produce el error de pinza «Tiempo excedido de la apertura de la pinza» y que se pulsa la tecla de función «Servicio».

Apretando la función Reintentó de la rutina de servicio, se producirá un nuevo intento de apertura de la pinza y a continuación, la ejecución del programa continuará.

Funciones atribuidas por defecto:

Ninguna función está atribuida, sólo aparecerá visualizado un texto de información y se indicará la tecla «Retorno». Para finalizar la rutina de servicio, se deberá apretar la tecla «Retorno».

PROC sw_service_wf();

fallo de soldadura_servicio_soldadura por puntos

La rutina es ejecutada cuando se produce el error «Tiempo excedido de soldadura lista» y que se pulsa la tecla de función «Servicio».

Apretando la función Reintentó de la rutina de servicio, se producirá un nuevo intento

de soldadura del punto y a continuación, la ejecución del programa continuará.

Funciones atribuidas por defecto:

Ninguna función está atribuida, sólo aparecerá visualizado un texto de información y se indicará la tecla «Retorno». Para finalizar la rutina de servicio, se deberá apretar la tecla «Retorno».

Funciones y rutinas generales del usuario:

Las siguientes rutinas están conectadas a diferentes entradas en el software de base en RAPID de la aplicación SpotWare.

FUNC num sw_close_time (spotdata spot, gundata gun);

cálculo del tiempo de cierre en soldadura por puntos

La función es ejecutada durante la preparación del movimiento. Retorna el tiempo de semicierre de la pinza.

Parámetros de entrada:

Punto de soldadura del tipo spotdata: punto de soldadura actual proporcionado en la instrucción SpotL en preparación.

Pinza del tipo gundata: pinza actual proporcionada en la instrucción SpotL en preparación.

Funciones atribuidas por defecto:

El tiempo resultante dependerá de los cuatro niveles de presión en los datos de pinza (gundata).

Módulo del Sistema **SWTOOL**

El módulo del sistema SWTOOL contiene funciones de datos y rutinas. El módulo ha sido declarado NOVIEW y contiene utilidades que se utilizan como una caja de herramientas cuando se realiza la personalización de SpotWare. Este módulo suele estar accesible en todas las tareas RAPID de SpotWare.

Contenido

Valores de retorno de función

SW_OK

SW_RETRY

SW_SERVICE

SW_ERROR

SW_TIMOUT

SW_CANCEL

Tipos de recuperación de error para el usuario (véase la función *sw_user_recover* en *SWUSRC*)

SW_WELD_ERR

SW(CG)_ERR

SW(OG)_ERR

SW_PRESUP_ERR

SW_POSUP_ERR

Rutinas

Las siguientes rutinas predefinidas vienen instaladas con la aplicación. Sirven para modificar la instrucción *SpotL*.

PROC SwSetCurrSpot (spotdata spot)

SpotWeldSetCurrentSpot

La función cambia el parámetro spotdata de datos de soldadura del proceso de soldadura por puntos que está en curso de ejecución.

PROC SwSetCurrGun (gundata gun)

SpotWeldSetCurrentGun

La función cambia el parámetro gundata de datos de pinza del proceso de soldadura por puntos que está en curso de ejecución.

PROC SwSetCurrRetr (num retr)

SpotWeldSetCurrentRetract

La función cambia el parámetro retract (apertura de la pinza) del proceso de soldadura por puntos que está en curso de ejecución.

retr 0: semi-apertura

retr 1: apertura total

PROC SwGetCurrNoConc (bool noconc)

SpotWeldSetCurrentNoConcurrency

La función cambia el parámetro NoConc del proceso de soldadura por puntos que está en curso de ejecución.

noconc TRUE: NoConc está presente

noconc FALSE: NoConc no está presente

PROC SwSetCurrSpotID (string spot_id)

SpotWeldSetCurrentSpotIdentity

La función cambia el *nombre* del parámetro spotdata de datos de soldadura del proceso de soldadura por puntos que está en curso de ejecución.

Funciones

Las siguientes funciones predefinidas vienen instaladas con la aplicación.

FUNC spotdata SwGetCurrSpot

SpotWeldGetCurrentSpot

La función retorna el contenido del parámetro spotdata de datos de soldadura por puntos del proceso de soldadura por puntos que está en curso de ejecución.

FUNC gundata SwGetCurrGun

SpotWeldGetCurrentGun

La función retorna el contenido del parámetro gundata de datos de pinza del proceso de soldadura por puntos que está en curso de ejecución.

FUNC num SwGetCurrRetr

SpotWeldGetCurrentRetract

La función retorna el contenido del parámetro retract (apertura de la pinza) del proceso de soldadura por puntos que está en curso de ejecución.

Valor de retorno 0: semi-apertura

Valor de retorno 1: apertura total

FUNC bool SwGetCurrNoConc

SpotWeldGetCurrentNoConcurrency

La función retorna el contenido del parámetro NoConc del proceso de soldadura por puntos que está en curso de ejecución.

Valor de retorno TRUE: NoConc está presente

Valor de retorno FALSE: NoConc no está presente

FUNC string SwGetCurrSpotID

SpotWeldGetCurrentSpotIdentity

La función retorna el *nombre* del parámetro spotdata de datos de soldadura por puntos del proceso de soldadura por puntos que está en curso de ejecución.

FUNC num SwWaitInput (VAR signaldi input, num value |num MaxWait)

SpotWeldWaitInput

Espera hasta que la señal de entrada haya alcanzado el valor. Se puede añadir como parámetro opcional un tiempo máximo *MaxWait*. Cuando se ejecuta una rutina de usuario llamada por el proceso de soldadura por puntos, en la que se supone que el proceso esperará una señal de entrada, esta función sirve para dejar que el sistema abandone el proceso actual.

Valor de retorno SW_OK: La señal estaba activada en la salida *value*.

Valor de retorno SW_TIMEOUT: El tiempo especificado en *MaxWait* ha sido excedido.

Valor de retorno SW_CANCEL: El proceso SpotWare ha recibido un aborto y desea cancelar el proceso actual. Provocará un retorno desde la rutina del usuario actual.

FUNC num SwWaitOutput (VAR signaldo output, num value |num MaxWait)

SpotWeldWaitOutput

Espera hasta que la señal de salida haya alcanzado el valor. Se puede añadir como parámetro opcional un tiempo máximo *MaxWait*. Cuando se ejecuta una rutina de usuario llamada por el proceso de soldadura por puntos, en la que se supone que el proceso esperará una señal de salida, esta función sirve para dejar que el sistema abandone el proceso actual.

Valor de retorno SW_OK: La señal estaba activada en *value*.

Valor de retorno SW_TIMEOUT: El tiempo especificado en *MaxWait* ha sido excedido.

Valor de retorno SW_CANCEL: El proceso SpotWare ha recibido un aborto y desea cancelar el proceso actual. Provocará un retorno desde la rutina del usuario actual.

FUNC num SwErrorAck (VAR num input_key, string alert, string ktxt1, string ktxt2, string ktxt3, string ktxt4, string ktxt5)

SpotWeldErrorAcknowledge

Esta función corresponde a una función TPReadFK modificada que puede ser abortada por SpotWare. Los parámetros corresponden a los de la función TPReadFK estándar. Cuando esta rutina no es utilizada para los diálogos de la unidad de programación, SpotWare queda bloqueada hasta que se libere desde la unidad de programación. La función es utilizada por *sw_error_recover* en el módulo *SWUSRC*.

Valor de retorno SW_OK: ejecución normal, el resultado de apretar una tecla de función reside en *input_key*.

Valor de retorno SW_CANCEL: el proceso de soldadura por puntos ha sido abortado. Retorno desde la rutina del usuario que ha llamado.

INDICE

ggundata	Datos de pistola de aplicación de adhesivo
GlueC	Aplicación de adhesivo con un movimiento circular
GlueL	Aplicación de adhesivo con un movimiento lineal
Módulo del Sistema	GLUSER

ggundata**Datos de pistola de aplicación de adhesivo**

Ggundata sirve para definir los datos específicos de la pistola de aplicación de adhesivo, que serán utilizados posteriormente para controlar la pistola de aplicación de adhesivo de la mejor forma durante el proceso de aplicación de adhesivo.

Observar que el TCP y el peso de la pistola de aplicación de adhesivo están definidos en *tooldata*.

Descripción

Ggundata se utiliza en instrucciones de aplicación de adhesivo y tiene la siguiente estructura:

- La pistola que se debe usar (1 o 2).
- Tiempo para semiapertura y semicerre de la pistola.
- Tipo de flujo1 y flujo2.
- Tiempos de preactivación de flujo1 y flujo2 en posición activada.
- Tiempos de preactivación de los cambios de flujo1 y flujo2.
- Tiempos de preactivación de flujo1 y flujo2 en posición desactivada.
- Tiempos para retraso en la pistola de adhesivo para diferentes velocidades
- Valores de la velocidad a la que se activará el valor máximo lógico de las salidas analógicas.

Componentes

ggun_no *(número de la pistola a usar)* Tipo de dato: *num*

Número a definir si este dato *ggundata* es para la pistola 1 o la pistola 2.

gl_on_time *(tiempo de semiapertura)* Tipo de dato: *num*

Tiempo en segundos necesario para la apertura de la pistola.

gl_off_time *(tiempo de semicerre)* Tipo de dato: *num*

Tiempo en segundos necesario para el cierre la pistola.

f11_type *(tipo de flujo1)* Tipo de dato: *num*

Tipo de la señal de flujo1, activada en: ninguna, fija o proporcional, donde ninguna = 0, fija = 1 y proporcional = 2.

f11_on_time (*preactivación tiempo flujo1*)

Tipo de dato: *num*

Tiempo en segundos necesario para activar el flujo1 en la pistola cuando la instrucción ha sido programada con el argumento \On.

f11_time (*preactivación tiempo cambio flujo1*)

Tipo de dato: *num*

Tiempo en segundos necesario para activar el flujo1 en la pistola cuando la instrucción ha sido programada sin los argumentos \On y \Off (es decir, el tiempo necesario para cambiar el flujo de un valor específico a otro).

f11_off_time (*preactivación tiempo reinicialización flujo1*)

) Tipo de dato: *num*

Tiempo en segundos necesario para reinicializar el flujo1 en la pistola cuando la instrucción ha sido programada con el argumento \Off.

f11_delay (*retraso flujo1*) Tipo de dato: *num*

Tiempo en segundos para compensar el retraso de la pistola de adhesivo para diferentes velocidades del TCP.

f11_refspeed (*vel. referencia adhesivo*) Tipo de dato: *num*

Velocidad de referencia de aplicación de adhesivo en mm/s. Normalmente es la velocidad máx. de aplicación de adhesivo para esta pistola. Se utiliza en el cálculo del valor de flujo1 para las señales proporcionales de velocidad. Este valor deberá ser > 0 también cuando el tipo de flujo=fijo.

f12_type (*tipo de flujo2*) Tipo de dato: *num*

Tipo de señal de flujo2, activada en: ninguna, fija o proporcional, donde ninguna = 0, fija = 1 y proporcional = 2.

f12_on_time (*preactivación tiempo flujo2*)

Tipo de dato: *num*

Tiempo en segundos necesario para activar el flujo2 de la pistola cuando la instrucción ha sido programada con el argumento \On.

f12_time (*preactivación tiempo cambio flujo2*)

Tipo de dato: *num*

Tiempo en segundos necesario para activar el flujo2 en la pistola cuando la instrucción ha sido programada sin los argumentos \On y \Off (es decir, el tiempo necesario para cambiar el flujo de un valor específico a otro).

f12_off_time (*preactivación tiempo reinicialización flujo2*)

)Tipo de dato: *num*

Tiempo en segundos necesario para reinicializar el flujo2 en la pistola cuando la instrucción ha sido programada con el argumento \Off.

f12_delay (*retraso flujo2*) Tipo de dato: *num*

Tiempo en segundos para compensar el retraso de la pistola de adhesivo para diferentes velocidades del TCP.

f12_refspeed (*vel. referencia adhesivo*) Tipo de dato: *num*

Velocidad de referencia de aplicación de adhesivo en mm/s. Normalmente es la velocidad máx. de aplicación de adhesivo para esta pistola. Se utiliza en el cálculo del valor de flujo2 para las señales proporcionales de velocidad. Este valor deberá ser > 0 también cuando el tipo de flujo=fijo.

Limitaciones

Número máximo de pistolas: 2

Compensación máx. retraso real (*f11_delay* y *f12_delay*) suele ser de: 40 - 60 ms.

Si se cambia el parámetro del sistema (*EventPresetTime*), la compensación del retraso alcanzará el grado correspondiente.

Los valores de los diferentes tiempos dentro del conjunto de datos deberá encontrarse entre 0 y 1 segundos.

El valor de la velocidad de referencia deberá ser > 0.

Datos predefinidos

El dato predefinido *ggun1* define la utilización de la pistola número 1, un flujo1 y un flujo2 del tipo 2, es decir, que se utilizan valores de flujo proporcionales y todos los tiempos están puestos en cero y la velocidad de referencia es de 1000 mm/s para cada flujo¹.

PERS ggun1 := [1,0,0,2,0,1,0,0,0,1000,2,0,1,0,0,0,1000];

Estructura

< dataobject de ggundata>

1. El tiempo predeterminado para los flujos 1 y 2 es de 0,1 s. (Las señales de flujo normalmente serán activadas antes de que la pistola se haya abierto).

<ggun_no de num>
<gl_on_time de num>
<gl_off_time de num>
<fl1_type de num>
<fl1_on_time de num>
<fl1_time de num>
<fl1_off_time de num>
<fl1_delay de num>
<fl1_refspeed de num>
<fl2_type de num>
<fl2_on_time de num>
<fl2_time de num>
<fl2_off_time de num>
<fl2_delay de num>
<fl2_refspeed de num>

Información relacionada

Instrucción de aplicación de adhesivo

Descripción:

Instrucciones - *GlueL / GlueC*

GlueC**Aplicación de adhesivo con un movimiento circular**

GlueC (GlueCircular) se utiliza en aplicación de adhesivo para controlar el movimiento, la apertura de la pistola y el proceso de aplicación de adhesivo. *GlueC* mueve el TCP siguiendo una trayectoria circular hacia la posición final.

Ejemplo 1

```
GlueL \On, p1, v250, ggun1 \F1:=100 \F2:=80, z30, tool7;
GlueC p2, p3, v250, ggun1 \F1:=90 \F2:=70, z30, tool7;
GlueL \Off, p4, v250, ggun1, z30, tool7;
```

1. El TCP para *tool7* se mueve siguiendo una trayectoria lineal hacia la posición *p1* con la velocidad proporcionada en *v250*. Debido al argumento *\On* la pistola se abre y el flujo de adhesivo empieza según los datos proporcionados en *ggundata ggun1* con anticipación en su camino hacia *p1*. El flujo de adhesivo empieza con los valores de porcentajes proporcionados por los parámetros *\F1:=100* y *\F2:=80*.
2. El TCP se moverá a continuación de *p1* hacia *p3* con los valores de flujo proporcionados por la instrucción de adhesivo precedente. Antes de alcanzar *p3*, los valores de flujo pasan al 90% y al 70% respectivamente. El momento en que esto ocurre está especificado en *ggun1*.
3. El TCP se moverá a continuación de *p3* hacia *p4* con los valores de flujo proporcionados por la instrucción de adhesivo precedente. Debido al argumento *\Off* las salidas serán reinicializadas de acuerdo con los tiempos proporcionados en *ggun1* antes de alcanzar *p3*.

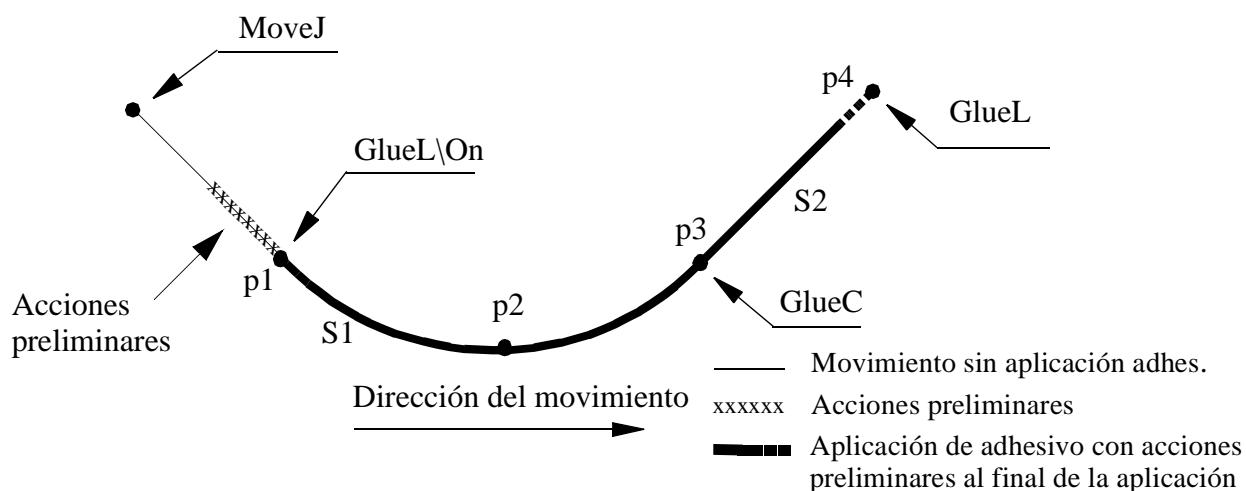


Figura 1 Ejemplo 1 de aplicación de adhesivo

S1: Flow1=100, Flow2=80, es decir que los valores de adhesivo proporcionados

por la primera instrucción están activos.

S2: Flow1=90, Flow2=70, es decir que los valores de adhesivo proporcionados por la segunda instrucción están activos.

Argumentos

GlueC [|On|][|Off|] [|Conc] PuntoCirc AlPunto Vel PistolaAdhesivo [|F1] [|F2] [|D] Zona Herram [|WObj]

[|On|]

Tipo de dato: *switch*

El argumento *|On* se usa en la primera instrucción de aplicación de adhesivo para arrancar el proceso de aplicación de adhesivo (véase la Figura 1).

El argumento sólo podrá utilizarse en la primera instrucción de aplicación de adhesivo para llevar a cabo la apertura necesaria de la pistola y la activación del flujo por anticipado. Si se ejecutan dos instrucciones consecutivas con el argumento *|On* se producirá un mensaje de error.

Dado que la instrucción no puede contener a la vez los argumentos *|On* y *|Off* juntos, la trayectoria de adhesivo deberá tener por lo menos 2 instrucciones, una que contiene el argumento *|On* y otra que contiene el argumento *|Off*.

Las acciones preliminares, que se realizan en la trayectoria hacia la posición programada, sirven para activar las salidas de apertura de la pistola y de aplicación analógica de adhesivo.

[|Off|]

Tipo de dato: *switch*

El argumento *|Off* se usa en la última instrucción de aplicación de adhesivo y sirve para terminar la aplicación una vez se ha alcanzado la posición programada. En su camino hacia la posición final, la salida para la apertura de la pistola así como las salidas de flujo, serán reinicializadas según los valores de tiempo especificados en los datos ggundata.

De esta forma es imposible terminar la aplicación de adhesivo sin utilizar el argumento *|Off* durante los movimientos.

[|Conc|]

(Concurrente)

Tipo de dato: *switch*

Mientras el robot se está moviendo, se van ejecutando diversas instrucciones. Este argumento se utiliza en las mismas situaciones que el argumento correspondiente en otras instrucciones de Movimiento pero en la aplicación de adhesivo es muy útil para permitir velocidades más elevadas cuando existen instrucciones de aplicación de adhesivo consecutivas que están juntas unas a otras.

Con el argumento *|Conc*, el número de instrucciones de movimiento sucesivas está limitado a 5. En una sección de programa que incluye la secuencia *Store-Path-Restopath*, no se permitirán instrucciones de movimiento con el argumento *|Conc*.

Si se omite este argumento, la instrucción siguiente sólo será ejecutada después de que el robot haya alcanzado la zona especificada.

PuntCircTipo de dato: *robtarget*

Es el punto circular del robot. El punto circular es una posición situada en el círculo entre el punto de arranque y el punto de destino. Para una mayor precisión, deberá estar situado a medio camino entre los puntos de arranque y de destino. Si está situado demasiado cerca del punto de arranque o de destino, puede ocurrir que el robot genere un mensaje de aviso. El punto circular es definido como una posición con nombre o es almacenado directamente en la instrucción (si está marcado con un asterisco * en la instrucción).

AlPuntoTipo de dato: *robtarget*

Es la posición de destino del robot y de los ejes externos. Suele estar definido como una posición con nombre o es almacenado directamente en la instrucción (si está marcado por un asterisco * en la instrucción).

VelocidadTipo de dato: *speeddata*

El dato de velocidad se aplica a los movimientos. Los datos de velocidad describen la velocidad del TCP, de la reorientación de la herramienta y la velocidad de los ejes externos.

PistolaAdhesivoTipo de dato: *ggundata*

Son los datos específicos de la pistola, por ejemplo, el tiempo de reacción para activar un flujo, etc., para la pistola en uso.

\[F1]

(Flujo1)

Tipo de dato: *num*

El argumento \F1 proporciona un valor de porcentaje para ajustar el flujo1 para la parte siguiente de la trayectoria. Si no se ha programado ningún valor, se utilizará el mismo valor que se utilizó en la instrucción precedente de aplicación de adhesivo. Si no se ha programado ningún valor en una instrucción de aplicación de adhesivo con el argumento \On se utilizará el valor 0.

\[F2]

(Flujo2)

Tipo de dato: *num*

El argumento \F2 proporciona un valor de porcentaje para ajustar el flujo2 para la parte siguiente de la trayectoria. Si no se ha programado ningún valor, se utilizará el mismo valor que se utilizó en la instrucción precedente de aplicación de adhesivo. Si no se ha programado ningún valor en una instrucción de aplicación de adhesivo con el argumento \On, se utilizará el valor 0.

\[D]

(Distancia)

Tipo de dato: *num*

El argumento opcional \D ofrece la posibilidad de llevar a cabo todas las acciones preliminares durante la instrucción a una distancia determinada (mm) con antelación a la posición programada.

Zona	Tipo de dato: <i>zonedata</i>
Son los datos de zona para el movimiento. Los datos de zona describen el tamaño de la trayectoria esquina generada.	
Herram	Tipo de dato: <i>tooldata</i>
Es la herramienta utilizada en el movimiento del robot. El TCP de la herramienta es el punto que se mueve a la posición de destino especificada, y deberá corresponder con la posición del elemento final cuando la pistola está abierta.	
[WObj]	<i>(Objeto de trabajo)</i>
Es el objeto de trabajo (sistema de coordenadas) al que se refiere la posición del robot en la instrucción.	Tipo de dato: <i>wobjdata</i>
Cuando se omite este argumento, la posición del robot se referirá al sistema de coordenadas mundo. No obstante, deberá especificarse, si se utiliza un TCP estacionario o ejes externos coordinados para poder realizar un movimiento lineal respecto al objeto de trabajo.	

Personalización de GlueWare

El paquete de programa GlueWare ofrece al usuario una multitud de oportunidades para personalizar y adaptar a cada instalación específica la instrucción *GlueC*:

- mediante los datos definidos por el usuario, que afectan el comportamiento interno en *GlueC*. (Véase Programas y Datos Predefinidos - *Módulo del Sistema GLUSER*).
- mediante el cambio de la configuración de E/S. (Véase Parámetros del Sistema-*Aplicación de adhesivo*).

Sin embargo, el cuerpo principal de esta instrucción *GlueC* está indicada en la configuración por defecto de la instalación.

Ejecución del programa

A continuación se indica la secuencia interna contenida en una instrucción *GlueC*:

- La pinza empieza a moverse hacia la posición.
- Si se está utilizando el argumento *\On*, la salida de apertura de la pistola *DO_Ggun1* o *DO_Ggun2* se activará en el momento específico antes de que la posición haya sido alcanzada. Si se está utilizando el argumento *\D*, la salida se activará a la distancia especificada además de los tiempos especificados, antes de que la posición haya sido alcanzada.
- Los valores para *flujo1* y *flujo2* se activan en las salidas analógicas *AO_G1Flow1* y *AO_G1Flow2*, y *AO_G2Flow1* y *AO_G2Flow2* respectivamente en el tiempo específico antes de que la posición haya sido

alcanzada. Si se está utilizando el argumento `\D`, las salidas se activarán a la distancia especificada además de los tiempos especificados, antes de que la posición haya sido alcanzada.

- Una vez que la posición programada ha sido alcanzada, la ejecución del programa continuará con la instrucción siguiente.

Cálculo de los valores de flujo de adhesivo

Véase la descripción correspondiente de GlueL.

Arranque y paro del programa

Paro durante el proceso de aplicación de adhesivo

Si ocurre un paro cuando el proceso de aplicación de adhesivo está activo, las salidas de adhesivo son eliminadas y aparecerá visualizado un mensaje de error. Cuando se ordena el rearranque, las demás instrucciones del conjunto actual de instrucciones de adhesivo serán llevadas a cabo como instrucciones de posicionamiento normales. La aplicación de adhesivo será rearrancada con la siguiente instrucción de adhesivo que contenga un argumento `\On`.

Ejecución instrucción por instrucción

Hacia adelante

La pistola se cierra y el movimiento sin aplicación de adhesivo es realizado.

Hacia atrás

La pistola se cierra y el movimiento hacia atrás es realizado sin aplicación de adhesivo.

Aplicación de adhesivo simulada

Se activa colocando la variable *gl_sim_glue* en TRUE. Ello tendrá por efecto la inhabilitación de las señales de apertura de la pistola y de flujo. (Véase Programas y Datos Predefinidos - *Módulo del Sistema GLUSER*).

Limitaciones

No se podrá rearrancar la secuencia de aplicación de adhesivo actual después de un paro. La aplicación de adhesivo será rearrancada con la instrucción siguiente de aplicación de adhesivo que contenga un argumento *|On*.

La velocidad máxima de aplicación de adhesivo depende de la distancia entre las posiciones programadas. Si se ha programado una velocidad demasiado elevada, aparecerá el mensaje de error 50082. Se podrá evitar este error realizando lo siguiente:

- cambiando las posiciones de forma que no estén tan juntas entre sí
- utilizando el argumento *|Conc* en las instrucciones de aplicación de adhesivo
- disminuyendo la velocidad.

Gestión de errores

Situaciones de error

Pueden ocurrir las siguientes situaciones de error:

- Error de un argumento de instrucción.
- Valores *ggndata* incorrectos.
- Arranque sin un argumento *|On*.
- Arranque con dos instrucciones con argumento *|On*.
- Final con argumento *|Off* sin haber arrancado con el argumento *|On*.
- Paro durante la ejecución de instrucciones de aplicación de adhesivo.

La instrucción o el dato defectuoso deberá ser cambiado y el conjunto de instrucciones de aplicación de adhesivo actuales deberá ser rearrancado desde el principio.

Sintaxis

GlueC

```
[[‘On’][‘Off’]
[‘Conc’,’]
[PuntoCir‘:=’]<expresión (IN) de robtarget>‘,’
[AlPunto‘:=’]<expresión (IN) de robtarget>‘,’
[Vel‘:=’]<expresión (IN) de speeddata>‘,’
[PistolaAdhesivo‘:=’]<persistente (PERS) de ggundata>
    [‘F1:=’<expresión (IN) de num>]
    [‘F2:=’<expresión (IN) de num>]
    [‘D:=’<expresión (IN) de num>],’
[Zona‘:=’]<expresión (IN) de zonedata>‘,’
[Herram‘:=’]<persistente (PERS) de tooldata>
[‘WObj:=’<persistente (PERS) de wobjdata>]’;
```

Información relacionada

	<u>Descripción en:</u>
Otras instrucciones de posicionamiento	Resumen RAPID - <i>Movimiento</i>
Definición de la velocidad	Tipos de datos - <i>speeddata</i>
Definición de los datos de zona	Tipos de datos - <i>zonedata</i>
Definición de la herramienta	Tipos de datos - <i>tooldata</i>
Definición de los objetos de trabajo	Tipos de datos - <i>wobjdata</i>
Definición de los datos de pistola de adhesivo	Tipos de datos - <i>ggundata</i>
Aplicación de adhesivo - generalidades	Resumen RAPID - <i>GlueWare</i>
Personalización de herramientas	Programas y Datos Predefinidos - <i>Módulo del Sistema GLUSER</i>
Configuración de E/S	Parámetros del sistema - <i>GlueWare</i>
Movimiento en general	Principios de Movimiento y de E/S

GlueL**Aplicación de adhesivo con un movimiento lineal**

GlueL (GlueLinear) se utiliza en aplicación de adhesivo para controlar el movimiento, la apertura de la pinza y el proceso de aplicación de adhesivo. *GlueL* mueve el TCP siguiendo una trayectoria lineal hacia la posición final.

Ejemplo 1

```
GlueL |On, p1, v250, ggun1 |F1:=100 |F2:=80, z30, tool7;
GlueL p2, v250, ggun1 |F1:=90 |F2:=70, z30, tool7;
GlueL |Off, p3, v250, ggun1, z30, tool7;
```

1. El TCP para *tool7* se mueve en una trayectoria lineal hacia la posición *p1* con la velocidad proporcionada en *v250*. Debido al argumento *|On* la pistola se abre y el flujo de adhesivo empieza según los datos proporcionados en *ggundata ggun1* con anticipación en su camino hacia *p1*. El flujo de adhesivo empieza con los valores de porcentajes proporcionados por los parámetros *|F1:=100* y *|F2:=80*.
2. El TCP se moverá a continuación de *p1* hacia *p2* con los valores de flujo proporcionados por la instrucción de adhesivo precedente. Antes de alcanzar *p2*, los valores de flujo pasan al 90% y al 70% respectivamente. El momento en que esto ocurre está especificado en *ggun1*.
3. El TCP se moverá a continuación de *p2* hacia *p3* con los valores de flujo proporcionados por la instrucción de adhesivo precedente. Debido al argumento *|Off* las salidas serán reinicializadas de acuerdo con los tiempos proporcionados en *ggun1* antes de alcanzar *p3*.

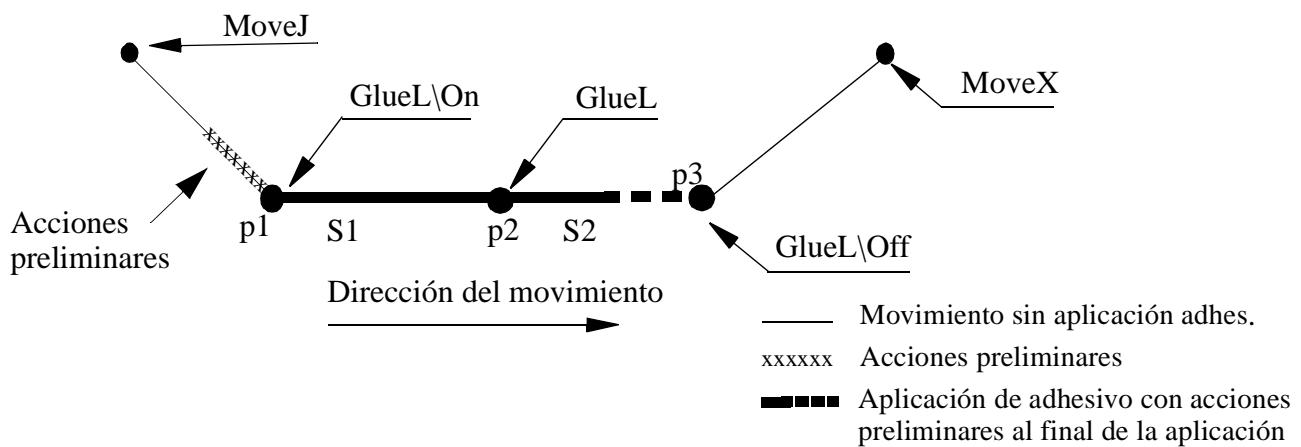


Figura 1 Ejemplo 1 de aplicación de adhesivo

S1: Flow1=100, Flow2=80, es decir, que los valores de adhesivo proporcionados por la primera instrucción están activos.

S2: Flow1=90, Flow2=70, es decir, que los valores de adhesivo proporcionados por la segunda instrucción están activos.

Argumentos

**GlueL [\\On]\\[Off] [\\Conc] AlPunto Vel PistolaAdhesivo [\\F1] [\\F2]
[\\D] Zona Herram [\\WObj]**

[\\On]

Tipo de dato: *switch*

El argumento *\\On* se usa en la primera instrucción de aplicación de adhesivo para arrancar el proceso de aplicación de adhesivo (véase la Figura 1).

El argumento sólo podrá utilizarse en la primera instrucción de aplicación de adhesivo para llevar a cabo la apertura necesaria de la pistola y la activación del flujo por anticipado. Si se ejecutan dos instrucciones consecutivas con el argumento *\\On* se producirá un mensaje de error.

Dado que la instrucción no puede contener a la vez los argumentos *\\On* y *\\Off* juntos, la trayectoria de adhesivo deberá tener por lo menos 2 instrucciones, una que contiene el argumento *\\On* y otra que contiene el argumento *\\Off*.

Las acciones preliminares, que se realizan en la trayectoria hacia la posición programada, sirven para activar las salidas de apertura de la pistola y de aplicación analógica de adhesivo.

[\\Off]

Tipo de dato: *switch*

El argumento *\\Off* se usa en la última instrucción de aplicación de adhesivo y sirve para terminar la aplicación una vez se ha alcanzado la posición programada. En su camino hacia la posición final, la salida para la apertura de la pistola así como las salidas de flujo, serán reinicializadas según los valores de tiempo especificados en los datos ggundata.

De esta forma es imposible terminar la aplicación de adhesivo sin utilizar el argumento *\\Off* durante los movimientos.

[\\Conc]

(Concurrente)

Tipo de dato: *switch*

Mientras el robot se está moviendo, se van ejecutando diversas instrucciones. Este argumento se utiliza en las mismas situaciones que el argumento correspondiente en otras instrucciones de Movimiento pero en la aplicación de adhesivo es muy útil para permitir velocidades más elevadas cuando existen instrucciones de aplicación de adhesivo consecutivas que están juntas unas a otras.

Con el argumento *\\Conc*, el número de instrucciones de movimiento sucesivas está limitado a 5. En una sección de programa que incluye la secuencia *Store-Path-Restopath*, no se permitirán instrucciones de movimiento con el argumento *\\Conc*.

Si se omite este argumento, la instrucción siguiente sólo será ejecutada después

de que el robot haya alcanzado la zona especificada.

AlPunto

Tipo de dato: *robtarget*

Es la posición de destino del robot y de los ejes externos. Suele estar definido como una posición con nombre o es almacenado directamente en la instrucción (si está marcado por un asterisco * en la instrucción).

Velocidad

Tipo de dato: *speeddata*

El dato de velocidad se aplica a los movimientos. Los datos de velocidad describen la velocidad del TCP, de la reorientación de la herramienta y la velocidad de los ejes externos.

PistolaAdhesivo

Tipo de dato: *ggundata*

Son los datos específicos de la pistola, por ejemplo el tiempo de reacción para activar un flujo, etc., para la pistola en uso (Véase Tipos de Datos - *Ggundata*).

[F1]

(*Flujo1*)

Tipo de dato: *num*

El argumento *F1* proporciona un valor de porcentaje para ajustar el flujo1 para la parte siguiente de la trayectoria. Si no se ha programado ningún valor, se utilizará el mismo valor que se utilizó en la instrucción precedente de aplicación de adhesivo. Si no se ha programado ningún valor en una instrucción de aplicación de adhesivo con el argumento *On*, se utilizará el valor 0.

[F2]

(*Flujo2*)

Tipo de dato: *num*

El argumento *F2* proporciona un valor de porcentaje para ajustar el flujo2 para la parte siguiente de la trayectoria. Si no se ha programado ningún valor, se utilizará el mismo valor que se utilizó en la instrucción precedente de aplicación de adhesivo. Si no se ha programado ningún valor en una instrucción de aplicación de adhesivo con el argumento *On*, se utilizará el valor 0.

[D]

(*Distancia*)

Tipo de dato: *num*

El argumento opcional *D* ofrece la posibilidad de llevar a cabo todas las acciones preliminares durante la instrucción a una distancia determinada (mm) con antelación a la posición programada.

Zona

Tipo de dato: *zonedata*

Son los datos de zona para el movimiento. Los datos de zona describen el tamaño de la trayectoria esquina generada.

Herram

Tipo de dato: *tooldata*

Es la herramienta utilizada en el movimiento del robot. El TCP de la herramienta es el punto que se mueve a la posición de destino especificada, y deberá corresponder con la posición del elemento final cuando la pistola está abierta.

[WObj]	<i>(Objeto de trabajo)</i>	Tipo de dato: <i>wobjdata</i>
---------------	----------------------------	-------------------------------

Es el objeto de trabajo (sistema de coordenadas) al que se refiere la posición del robot en la instrucción.

Cuando se omite este argumento, la posición del robot se referirá al sistema de coordenadas mundo. No obstante, deberá especificarse, si se utiliza un TCP estacionario o ejes externos coordinados para poder realizar un movimiento lineal respecto al objeto de trabajo.

Personalización de GlueWare

El paquete de programa GlueWare ofrece al usuario una multitud de oportunidades para personalizar y adaptar a cada instalación específica la instrucción *GlueL*:

- mediante los datos definidos por el usuario, que afectan el comportamiento interno en *GlueL*. (Véase Programas y Datos Predefinidos - *Módulo del Sistema GLUSER*).
- mediante el cambio de la configuración de E/S. (Véase Parámetros del Sistema- *Aplicación de adhesivo*).

Sin embargo, el cuerpo principal de esta instrucción *GlueL* está indicada en la configuración por defecto de la instalación.

Ejecución del programa

A continuación se indica la secuencia interna contenida en una instrucción *GlueL*:

- La pinza empieza a moverse hacia la posición.
- Si se está utilizando el argumento *|On*, la salida de apertura de la pistola *DO_Ggun1* o *DO_Ggun2* se activará en el momento específico antes de que la posición haya sido alcanzada. Si se está utilizando el argumento *|D* la salida se activará a la distancia especificada además de los tiempos especificados, antes de que la posición haya sido alcanzada.
- Los valores para flujo1 y flujo2 se activan en las salidas analógicas *AO_G1Flow1* y *AO_G1Flow2*, y *AO_G2Flow1* y *AO_G2Flow2* respectivamente en el tiempo específico antes de que la posición haya sido alcanzada. Si se está utilizando el argumento *|D*, las salidas se activarán a la distancia especificada además de los tiempos especificados, antes de que la posición haya sido alcanzada.
- Una vez que la posición programada ha sido alcanzada, la ejecución del programa continuará con la instrucción siguiente.

Cálculo de los valores de flujo de adhesivo

Los datos siguientes se utilizan cuando se calcula el valor lógico flujo1 (El valor lógico flujo2 se calcula de forma similar):

F1	parámetro de instrucción (0-100)
gl_fl1_ovr	ajuste global (Defecto: 100). (Véase Programas y Datos Predefinidos - <i>Módulo del Sistema GLUSER</i>)
gl_fl1_ref	valor máx. lógico (Defecto: 1000). (Véase Programas y Datos Predefinidos - <i>Módulo del Sistema GLUSER</i>)
current speed	velocidad actual del robot (mm/s).
fl1_refspeed	velocidad de referencia del adhesivo (mm/s). (Véase Tipos de Datos - <i>ggundata</i>)

Cálculo del flujo1 lógico cuando fl1_type = 1 (fijo):

$$\text{flujo1 lógico} = F1 * \text{gl_fl1_ovr} * \text{gl_fl1_ref} / 10000$$

Cálculo del flujo1 lógico cuando fl1_type = 2 (proporcional):

$$\text{flujo1 lógico} = (F1 * \text{gl_fl1_ovr} * \text{gl_fl1_ref} / 10000) * \text{current speed} / \text{fl1_refspeed}$$

Esto significa lo siguiente: Con los valores por defecto anteriores y con la configuración por defecto para los mín. y máx. lógicos de las salidas de adhesivo analógicas (Véase Parámetros del Sistema - *GlueWare*) obtenemos el siguiente resultado:

Si flow1_type = 1 (fijo): el valor máximo físico es activado si F1 = 100 (%) en la instrucción.

Si flow1_type = 2 (proporcional): el valor máximo físico es activado si F1 = 100 (%) en la instrucción y que la velocidad actual es la misma que fl1_refspeed en los datos actuales ggundata.

Ejemplo 2

```
GlueL \On, p1, v250, ggun1 \F1:=100 \F2:=80 \D:=50, z30, tool7;
GlueL p2, v250, ggun1 \F1:=90 \F2:=70 \D:=50, z30, tool7;
GlueL \Off, p3, v250, ggun1 \D:=50, z30, tool7;
```

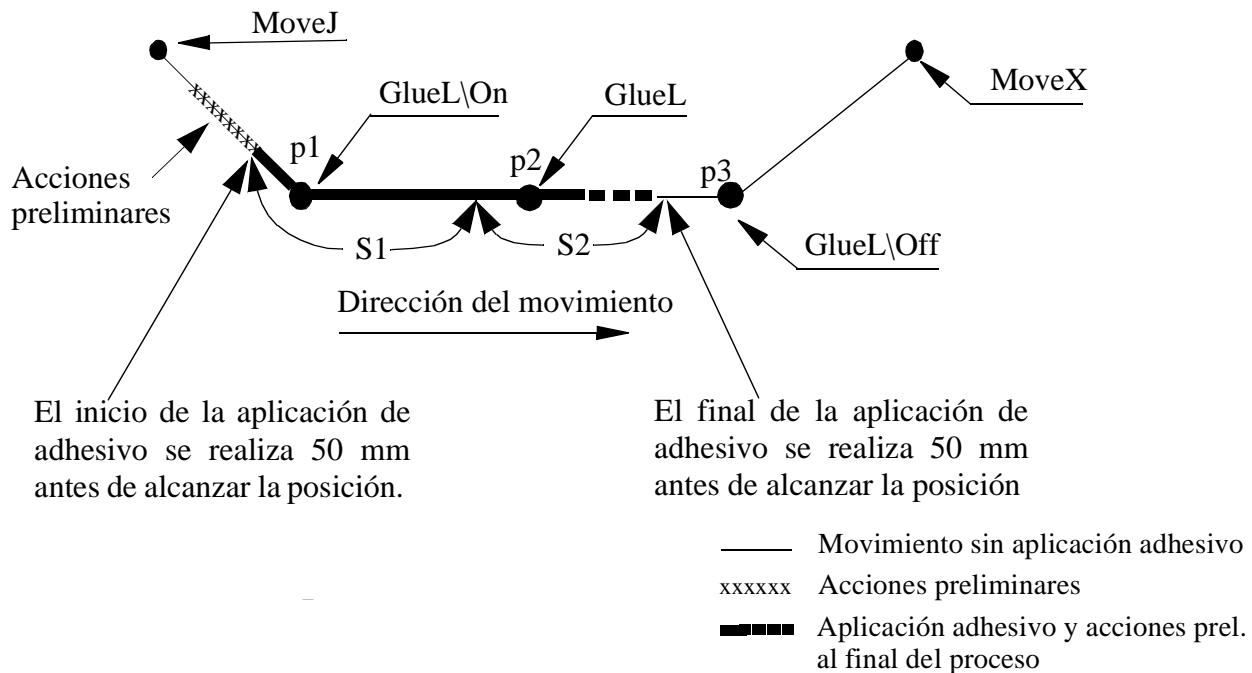


Figura 2 Ejemplo 2 de aplicación de adhesivo

En este ejemplo todo es lo mismo que en el ejemplo 1 excepto que todas las acciones específicas de aplicación de adhesivo se ejecutan a una distancia específica (50 mm) antes de alcanzar las posiciones programadas.

Imaginemos que el valor máximo lógico de las señales es 10. Cuando fl1_type=1 (fijo) y gl_fl1_our=100 (valor normal para el ajuste de flujo de adhesivo), se obtienen los valores 10 en el tramo S1 y 9 en el tramo S2 para flujo 1. Cuando se cambia gl_fl1_a a 50, se obtendrán los valores 500 y 450 respectivamente. Cuando fl2_type=2 (proporcional) y gl_fl2_our=100, se obtendrán los valores 800 en el tramo S1 y 700 en el tramo S2 para flujo 2, si la velocidad actual es la misma que la velocidad de referencia (fl2_refspeed).

Cuando la velocidad es reducida, por ejemplo, en una trayectoria esquina, el flujo será conformemente reducido. Los valores físicos de las señales están también determinados por la forma en que las señales analógicas han sido configuradas en los parámetros del sistema (relación entre valores físicos y lógicos).

Arranque y paro del programa

Paro durante el proceso de aplicación de adhesivo

Si ocurre un paro cuando el proceso de aplicación de adhesivo está activo, las salidas de adhesivo son eliminadas y aparecerá visualizado un mensaje de error. Cuando se ordena el rearranque, las demás instrucciones del conjunto actual de instrucciones de

adhesivo serán llevadas a cabo como instrucciones de posicionamiento normales. La aplicación de adhesivo será rearrancada con la siguiente instrucción de adhesivo que contenga un argumento *\On*.

Ejecución instrucción por instrucción

Hacia adelante

La pistola se cierra y el movimiento sin aplicación de adhesivo es realizado.

Hacia atrás

La pistola se cierra y el movimiento hacia atrás es realizado sin aplicación de adhesivo.

Aplicación de adhesivo simulada

Se activa colocando la variable *gl_sim_glue* en TRUE. Ello tendrá por efecto la inhabilitación de las señales de apertura de la pistola y de flujo. (Véase Programas y Datos Predefinidos - *Módulo del Sistema GLUSER*)

Limitaciones

No se podrá rearrancar la secuencia de aplicación de adhesivo actual después de un paro. La aplicación de adhesivo será rearrancada con la instrucción siguiente de aplicación de adhesivo que contenga un argumento *\On*.

La velocidad máxima de aplicación de adhesivo depende de la distancia entre las posiciones programadas. Si se ha programado una velocidad demasiado elevada, aparecerá el mensaje de error 50082. Se podrá evitar este error realizando lo siguiente:

- cambiando las posiciones de forma que no estén tan juntas entre sí
- utilizando el argumento *\Conc* en las instrucciones de aplicación de adhesivo
- disminuyendo la velocidad.

Gestión de errores

Situaciones de error

Pueden ocurrir las siguientes situaciones de error:

- Error de un argumento de instrucción.
- Valores *ggundata* incorrectos.
- Arranque sin un argumento \On.
- Arranque con dos instrucciones con argumento \On.
- Final con argumento \Off sin haber arrancado con el argumento \On.
- Paro durante la ejecución de instrucciones de aplicación de adhesivo.

La instrucción o el dato defectuoso deberá ser cambiado y el conjunto de instrucciones de aplicación de adhesivo actuales deberá ser rearrancado desde el principio.

Sintaxis

GlueL

```
[[\'On ]|[\'Off ]
[\' Conc]','
[AlPunto\' :=]<expresión (IN) de robtarget> ,
[Vel\' :=]<expresión (IN) de speeddata> ,
[PistolaAdhesivo\' :=]<persistente (PERS) de ggundata>
[\'F1\' :=<expresión (IN) de num>]
[\'F2\' :=<expresión (IN) de num>]
[\'D\' :=<expresión (IN) de num>]','
[Zona\' :=]<expresión (IN) de zonedata> ,
[Herram\' :=]<persistente (PERS) de tooldata>
[\'WObj\' :=<persistente (PERS) de wobjdata>];'
```

Información relacionada

Descripción en:

Otras instrucciones de posicionamiento	Resumen RAPID - <i>Movimiento</i>
Definición de la velocidad	Tipos de datos - <i>speeddata</i>
Definición de los datos de zona	Tipos de datos - <i>zonedata</i>
Definición de la herramienta	Tipos de datos - <i>tooldata</i>
Definición de los objetos de trabajo	Tipos de datos - <i>wobjdata</i>
Definición de los datos de pistola de adhesivo	Tipos de datos - <i>ggundata</i>
Aplicación de adhesivo - generalidades	Resumen RAPID - <i>GlueWare</i>
Personalización de herramientas	Programas y Datos Predefinidos - - <i>Módulo del Sistema GLUSER</i>
Configuración de E/S	Parámetros del sistema - <i>GlueWare</i>
Movimiento en general	Principios de Movimiento y de E/S

Módulo del Sistema GLUSER

El módulo del sistema GLUSER contiene datos y rutinas destinadas a personalizar el comportamiento de la aplicación GlueWare.

Los nombre son predefinidos y se utilizan de forma interna cuando se utiliza una instrucción *GlueL* o *GlueC*. Por esta razón, no se deberá cambiar los nombres.

Contenido

Datos

Los siguientes datos globales son predefinidos:

Nombre	Declaración	Descripción
gl_fl1_ovr	CONST num gl_fl1_ovr := 100	Ajuste global para la señal flujo1 Alcance: 0-200%
gl_fl2_ovr	CONST num gl_fl2_ovr := 100	Ajuste global para la señal flujo2 Alcance: 0-200%
gl_fl1_ref	CONST num gl_fl1_ref := 1000	Valor de referencia utilizado en el cálculo de flujo1 de adhesivo. Normalmente es el mismo valor que el Máx. lógico de la señal de salida analógica para flujo1.
gl_fl2_ref	CONST num gl_fl2_ref := 1000	Valor de referencia utilizado en el cálculo de flujo2 de adhesivo. Normalmente es el mismo valor que el Máx. lógico de la señal de salida analógica para flujo2.
gl_sim_glue	CONST bool gl_sim_glue := FALSE	Bandera que sirve para simular el proceso de aplicación de adhesivo. Si está en TRUE: ninguna señal de adhesivo será activada.
ggun1	PERS ggundata ggun1 := [1,0,0,2,0,0,0,0,1000,2,0,0,0,0,1000]	Datos ggundata predefinidos con valores defecto.

Rutinas

Existen algunas rutinas predefinidas instaladas en la aplicación. Estas rutinas no tienen funciones atribuidas por defecto, pero podrán ser cambiadas para personalizar el comportamiento de GlueWare.

rutina gl_err_actions

Esta rutina se ejecuta cuando ocurre un error de GlueWare.

routine gl_preglue

Esta rutina se ejecuta cuando se inicia un movimiento hacia una instrucción Glue\On (cuando la señal *DO_GL_Active* está activada en 1).

Nota: La rutina se ejecuta durante el movimiento del robot. Si las instrucciones dentro de esta rutina toman demasiado tiempo, ello reducirá la velocidad de aplicación de adhesivo posible.

routine gl_postglue

Esta rutina se ejecuta cuando la fase de aplicación de adhesivo ha terminado (cuando la señal *DO_GL_Actice* está activada en 0).

Nota: La rutina se ejecuta durante el movimiento del robot. Si las instrucciones dentro de esta rutina toman demasiado tiempo, ello reducirá la velocidad de aplicación de adhesivo posible.

rutina gl_power_on

Esta rutina es ejecutada cada vez que el sistema es puesto en marcha.

rutina gl_start

La rutina es ejecutada cada vez que se empieza la ejecución de un programa desde el principio. (Desde Main)

rutina gl_restart

La rutina es ejecutada cada vez que la ejecución de un programa parado es reanudada.

rutina gl_stop

Esta rutina es ejecutada cuando se realiza un paro normal durante la ejecución del programa.

rutina gl_qstop

Esta rutina es ejecutada cuando se realiza un paro de emergencia durante la ejecución del programa.

ÍNDICE

	Página
1 La Ventana de Movimiento	3
1.1 Ventana: Movimiento	3
1.1.1 Menú: Especial.....	3
2 La Ventana de Entradas/Salidas.....	4
2.1 Ventana: Entradas/Salidas	4
2.1.1 Menú: Archivo	4
2.1.2 Menú: Editar	5
2.1.3 Menú: Ver	5
3 La Ventana de Programa	6
3.1 Movimiento entre diferentes partes del programa	6
3.2 Menús generales	7
3.2.1 Menú: Archivo	7
3.2.2 Menú: Editar	8
3.2.3 Menú: Ver	9
3.3 Ventana: Instrucciones de Programa	10
3.3.1 Menú: IPL_1 (visualiza las diferentes listas de selección de instrucciones)	10
3.3.2 Menú: IPL_2 (visualiza las diferentes listas de selección de instrucciones)	11
3.4 Ventana: Rutinas del Programa	12
3.4.1 Menú: Rutina.....	13
3.4.2 Menú: Especial.....	13
3.5 Ventana: Datos del Programa	14
3.5.1 Menú: Datos	14
3.5.2 Menú: Especial.....	15
3.6 Ventana: Tipos de Datos del Programa	16
3.6.1 Menú: Tipos	16
3.7 Ventana: Test del Programa	17
3.7.1 Menú: Test	17
3.8 Ventana: Módulos del Programa	18
3.8.1 Menú: Módulo.....	18
4 La Ventana de Producción	19
4.1 Ventana: Funcionamiento en Producción.....	19
4.1.1 Menú: Archivo	19
4.1.2 Menú: Editar	19
4.1.3 Menú: Ver	20
5 El Administrador de Archivos	21
5.1 Ventana: Administrador de Archivos	21

Referência Rápida

5.1.1 Menú: Archivo	21
5.1.2 Menú: Editar	22
5.1.3 Menú: Ver	22
5.1.4 Menú: Opciones	22
6 La ventana de Servicio.....	23
6.1 Menús generales	23
6.1.1 Menú: Archivo	23
6.1.2 Menú: Editar	24
6.1.3 Menú: Ver	24
6.2 Ventana Registro de Servicio	25
6.2.1 Menú: Especial	25
6.3 Ventana de Calibración de Servicio	26
6.3.1 Menú: Calib	26
6.4 Ventana de Comutación de Servicio	27
6.4.1 Menú: Com	27
7 Los Parámetros del Sistema.....	28
7.1 Ventana: Parámetros del Sistema	28
7.1.1 Menú: Archivo	28
7.1.2 Menú: Editar	29
7.1.3 Menú: Temas	29
8 Ventana especiales ArcWare	30
8.1 Ventana: Producción	30
8.2 Ventana: Test del Programa.....	31
8.3 Ventana de ejecución	32

1 La Ventana de Movimiento

1.1 Ventana: Movimiento

Parámetros →
de movimiento
utilizados

Mvto. manual		Robot Pos:	
Unidad:	Robot	x:	1234.5 mm
Mvto:	Lineal	y:	-244.9 mm
Coord:	Base <input checked="" type="checkbox"/>	z:	12.8 mm
Herram:	herram0...	Q1:	0.7071
WObj:	Wobj0...	Q2:	0.0000
Bloqueo Joystick:	None	Q3:	0.0000
Incremento:	No <input type="checkbox"/>	Q4:	-0.7071
← ↕ ↑		x	y z
Mundo	Base	Herram	WObj

← Posición utilizada

← Movimiento resultante de diferentes deflexiones de la palanca de mando

1.1.1 Menú: Especial

Especial	
1	Alinear...
2	Incrementos...

Comando

Alinear

Incrementos

Sirve para:

Alinear la herramienta

Especificar los tamaños de los incrementos definidos por el usuario

2 La Ventana de Entradas/Salidas

2.1 Ventana: Entradas/Salidas

Nombre	Valor	Tipo
4 (64)		
di1	1	DI
di2	0	DI
pinzal	0	DO
pinza2	1	DO
pinza3	1	DO
pinza4	1	DO
noprog	13	GO
errorsold	0	DO
0	1	

2.1.1 Menú: Archivo

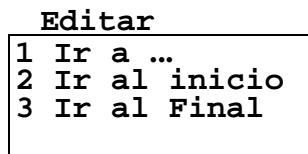
Archivo
1 Imprimir...
2 Preferencias...

Comando**Imprimir****Preferencias**Sirve para:

imprimir la lista de E/S utilizada

realizar una lista de preferencias en la ventana de Entradas/Salidas

2.1.2 Menú: Editar



Comando:

Ir a...

Ir al inicio

Ir al final

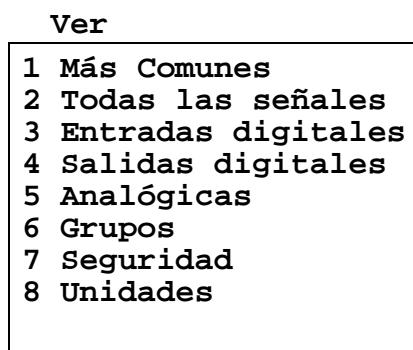
Sirve para:

ir a una línea específica de la lista

ir a la primera línea de la lista

ir a la última línea de la lista

2.1.3 Menú: Ver



Comando:

Más Comunes

Todas las señales

Entrada digital

Salida digital

Analógica

Grupos

Seguridad

Unidades

Sirve para visualizar:

la lista más común

todas las señales del usuario

todas las entradas digitales

todas las salidas digitales

todas las señales analógicas

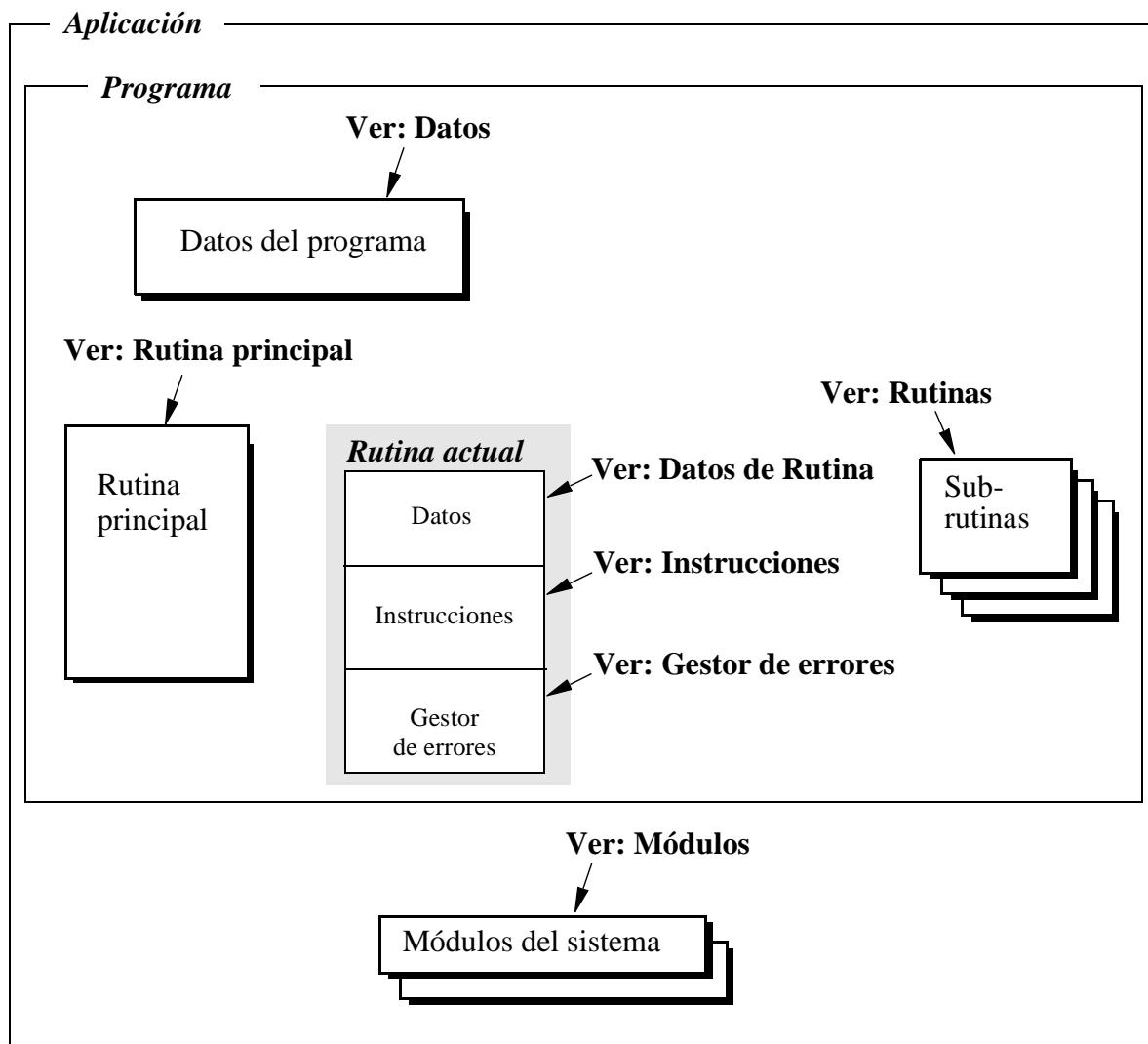
todos los grupos de señales digitales

todas las señales de seguridad

todas las unidades de E/S

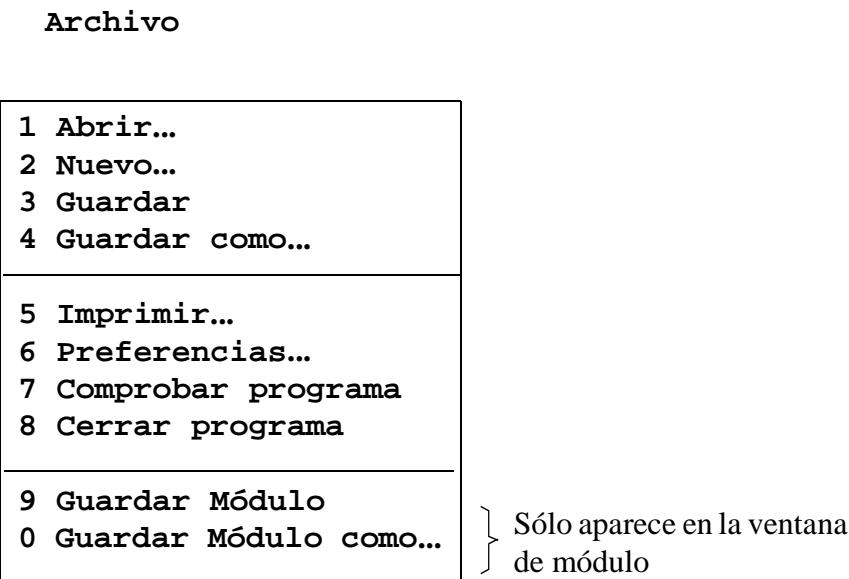
3 La Ventana de Programa

3.1 Movimiento entre diferentes partes del programa



3.2 Menús generales

3.2.1 Menú: Archivo



Comando:

Abrir
Nuevo
Guardar
Guardar como

Imprimir
Preferencias
Comprobar Programa
Cerrar Programa

Guardar Módulo
Guardar Módulo como

Sirve para:

leer programas de la memoria de masa
crear programas nuevos
guardar programas en la memoria de masa
guardar programas en la memoria de masa con
nombres nuevos

la impresión del programa
realizar preferencias en la ventana de Programa
la comprobación de que el programa es correcto
la eliminación de un programa de la memoria del
programa

guardar un módulo en la memoria de masa
guardar un módulo en la memoria de masa bajo un
nombre nuevo

3.2.2 Menú: Editar

Editar

```
Deshacer "Última acción"
1 Cortar
2 Copiar
3 Pegar
4 Ir al inicio
5 Ir al final
6 Marcar
7 Cambio seleccionado
8 Valor
9 ModPos
0 Buscar...
Ver/Ocultar IPL
```

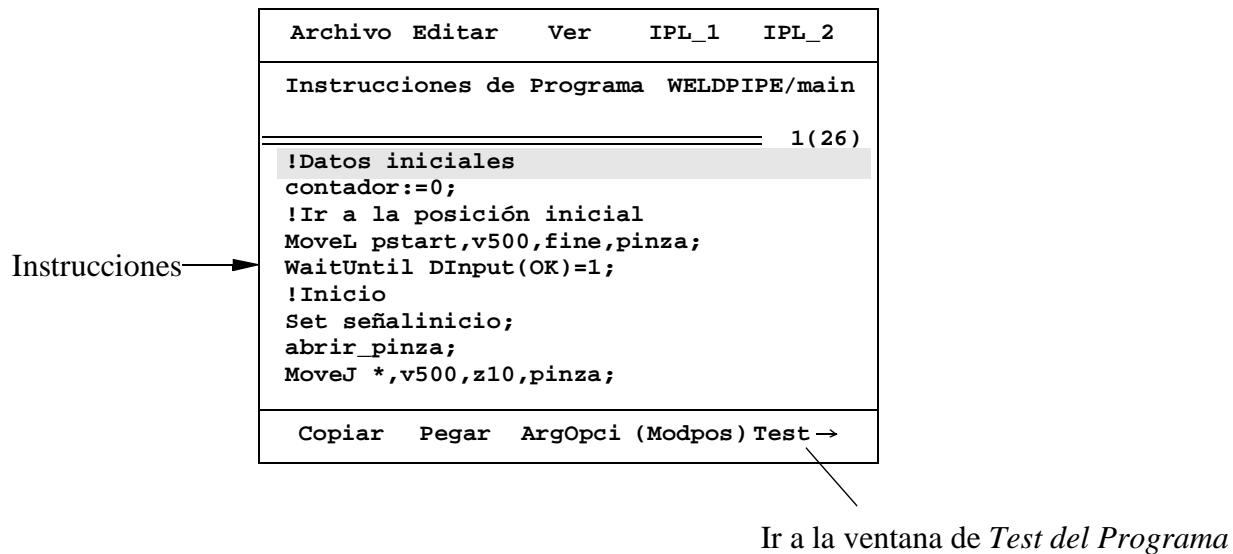
<u>Comando</u>	<u>Sirve para:</u>
Deshacer	llevar a cabo una acción de anulación de la última acción realizada y a la que se le puede aplicar el comando deshacer en la ventana seleccionada
Cortar	cortar las líneas seleccionadas al bloc de notas
Copiar	copiar las líneas seleccionadas al bloc de notas
Pegar	pegar el contenido del bloc de notas en un programa
Ir al inicio	ir a la primera línea
Ir al final	ir a la última línea
Marcar	seleccionar varias líneas
Cambio seleccionado	cambiar un argumento de instrucción
Valor	mostrar el valor utilizado (del argumento seleccionado)
Mod Pos	modificar una posición
Buscar	buscar/reemplazar un argumento específico
Ver/Ocultar IPL	Ver/ocultar una lista de selección de instrucciones

3.2.3 Menú: Ver

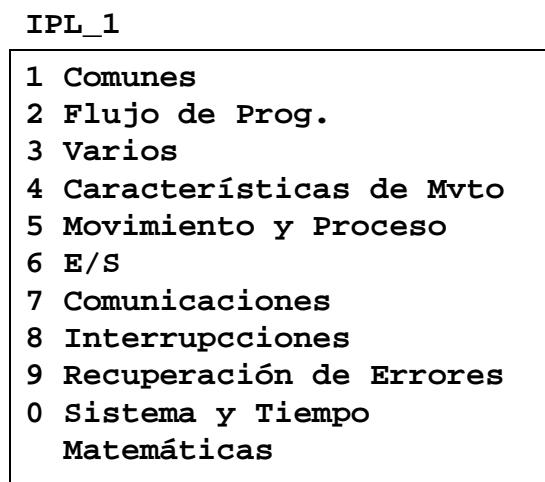
Ver
1 Instr. "<última rutina>"
2 Rutinas
3 Datos "<último tipo>"
4 Tipos de Datos
5 Test
6 Módulos
7 Rutina Main
8 Rutina seleccionada
9 Gestor de errores

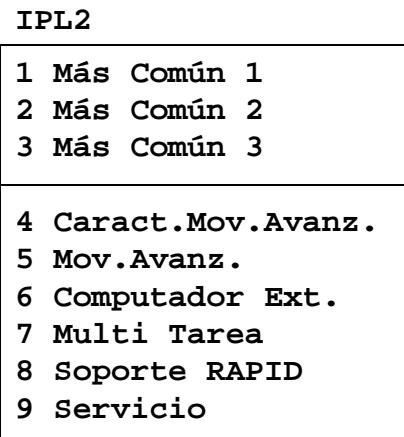
<u>Comando</u>	<u>Sirve para:</u>
Instr.	las instrucciones de la rutina utilizada
Rutinas	todas las rutinas
Datos	los datos del programa
Tipos de Datos	todo tipo de datos
Test	la ventana de <i>Test del Programa</i>
Módulos	todos los módulos
Rutina Main	las instrucciones de la rutina principal
Rutina Seleccionada	la instrucción de la rutina seleccionada
Gestor de errores	el gestor de errores de la rutina utilizada

3.3 Ventana: Instrucciones de Programa

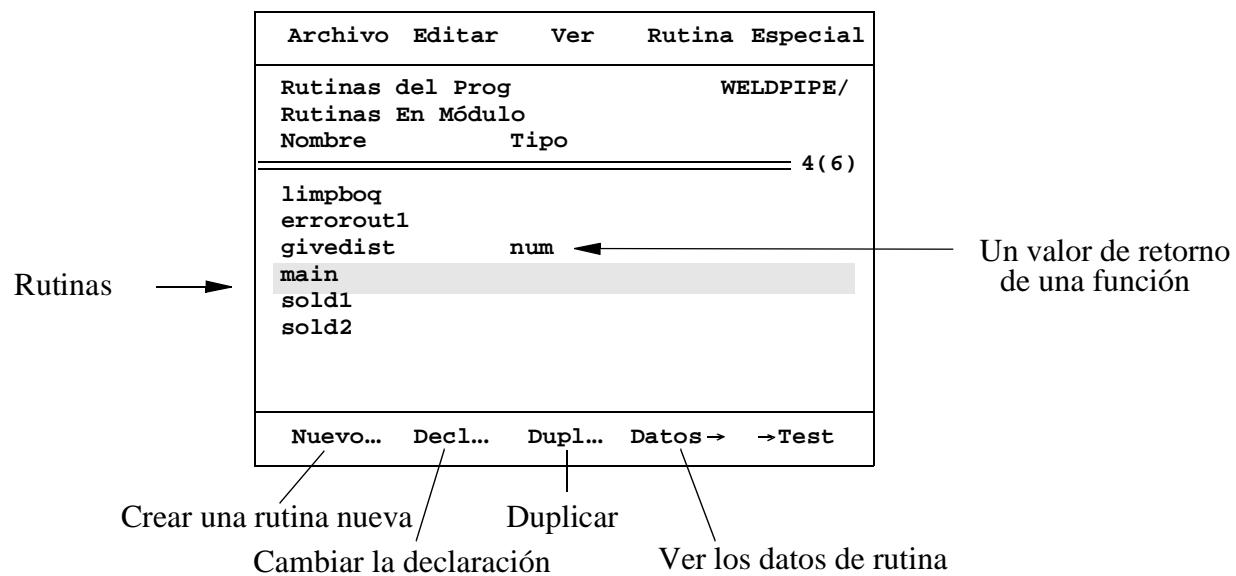


3.3.1 Menú: IPL_1 (visualiza las diferentes listas de selección de instrucciones)

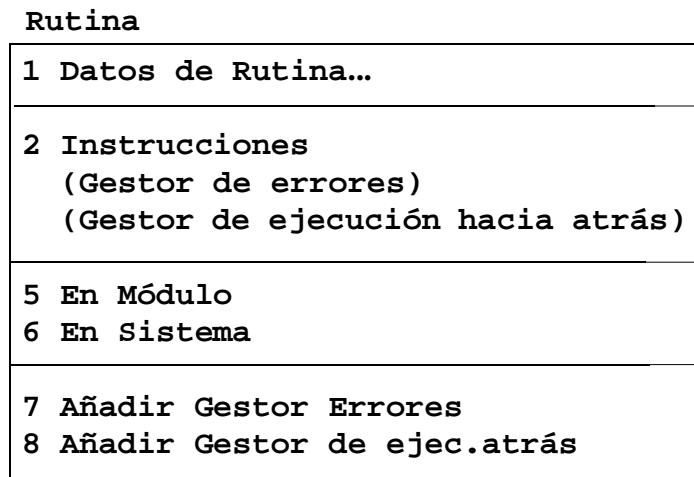


3.3.2 Menú: IPL_2 (visualiza las diferentes listas de selección de instrucciones)

3.4 Ventana: Rutinas del Programa

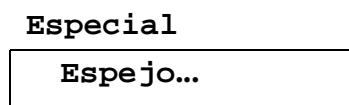


3.4.1 Menú: Rutina



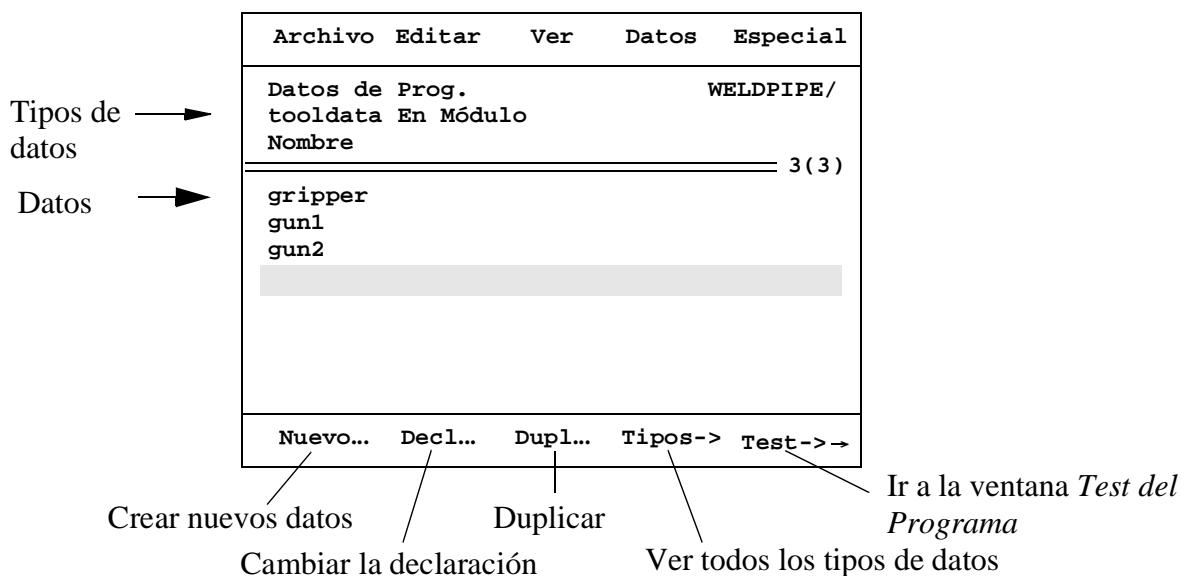
<u>Comando:</u>	<u>Sirve para:</u>
Datos de Rutina	crear una rutina nueva
Instrucciones	visualizar las instrucciones de la rutina seleccionada
Gestor de errores	visualizar el gestor de errores de la rutina seleccionada
Gestor de ejec. hacia atrás	visualizar el gestor de ejecución hacia atrás de la rutina seleccionada
En Módulo	visualizar únicamente las rutinas en el módulo utilizado
En Sistema	visualizar todas las rutinas de todos los módulos
Añadir/Borrar Gestor de Error	añadir/borrar un gestor de error a la rutina seleccionada
Añadir/Borrar Gestor de ejecución hacia atrás	añadir/borrar un gestor de ejecución hacia atrás a la rutina seleccionada

3.4.2 Menú: Especial

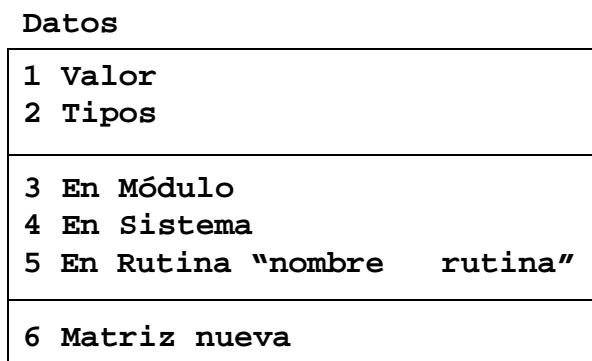


<u>Comando:</u>	<u>Sirve para:</u>
Espejo	crear una imagen espejo de una rutina o de un módulo

3.5 Ventana: Datos del Programa

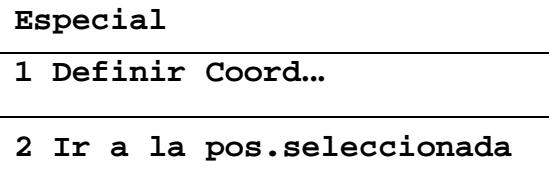


3.5.1 Menú: Datos



Comando:	Sirve para:
Valor	leer o cambiar el valor utilizado de los datos seleccionados
Tipos	llamar a la lista con todos los tipos de datos
En Módulo	llamar únicamente los datos de este módulo
En Sistema	crear datos nuevos
En Rutina	llamar todos los datos de rutina
Matriz nueva	declarar un dato de matriz nuevo

3.5.2 Menú: Especial



Comando:

Definir Coord

Ir a la pos. seleccionada

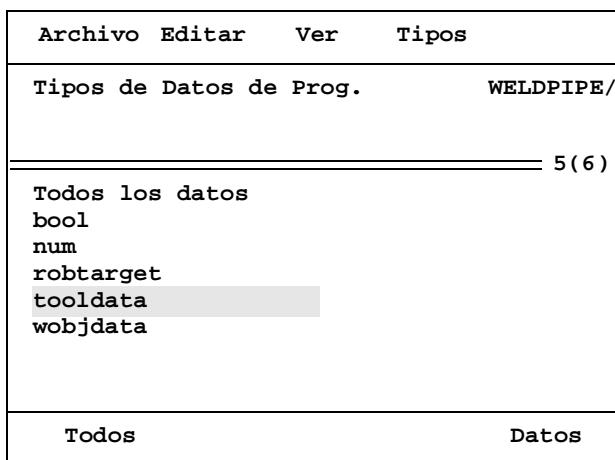
Sirve para:

la definición de una herramienta, de un objeto de trabajo o de un desplazamiento de programa)

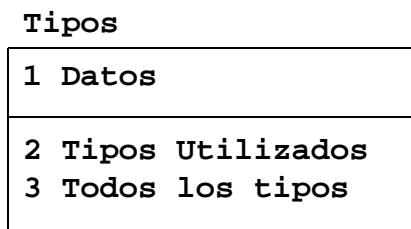
ir a la posición seleccionada

3.6 Ventana: Tipos de Datos del Programa

Tipos de datos →



3.6.1 Menú: Tipos



Comando:

Datos

Tipos Utilizados

Todos los tipos

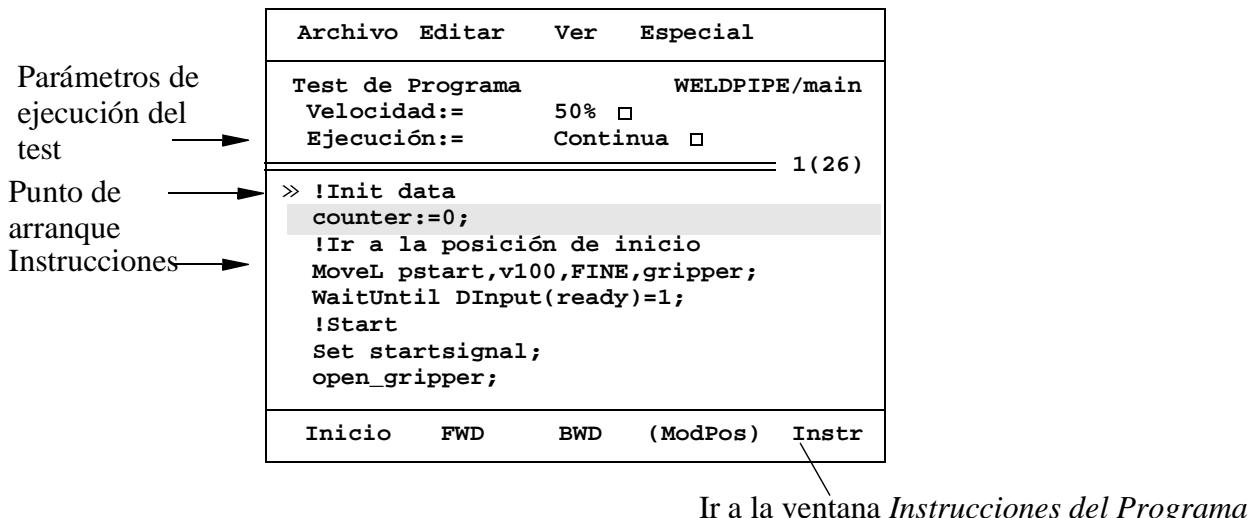
Sirve para:

llamar todos los datos de un tipo seleccionado

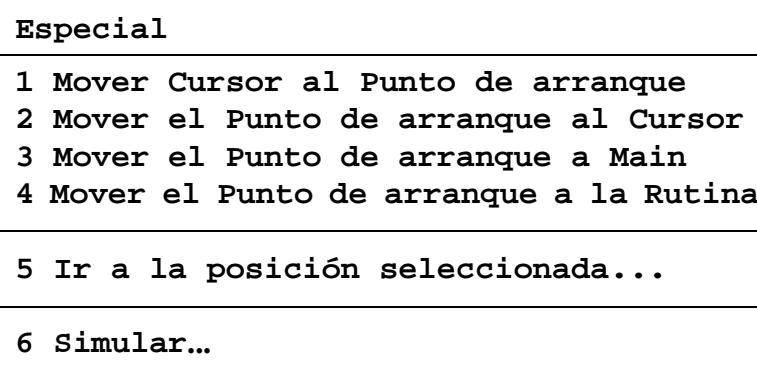
llamar únicamente aquellos tipos de datos que son utilizados

llamar todos los tipos de datos

3.7 Ventana: Test del Programa

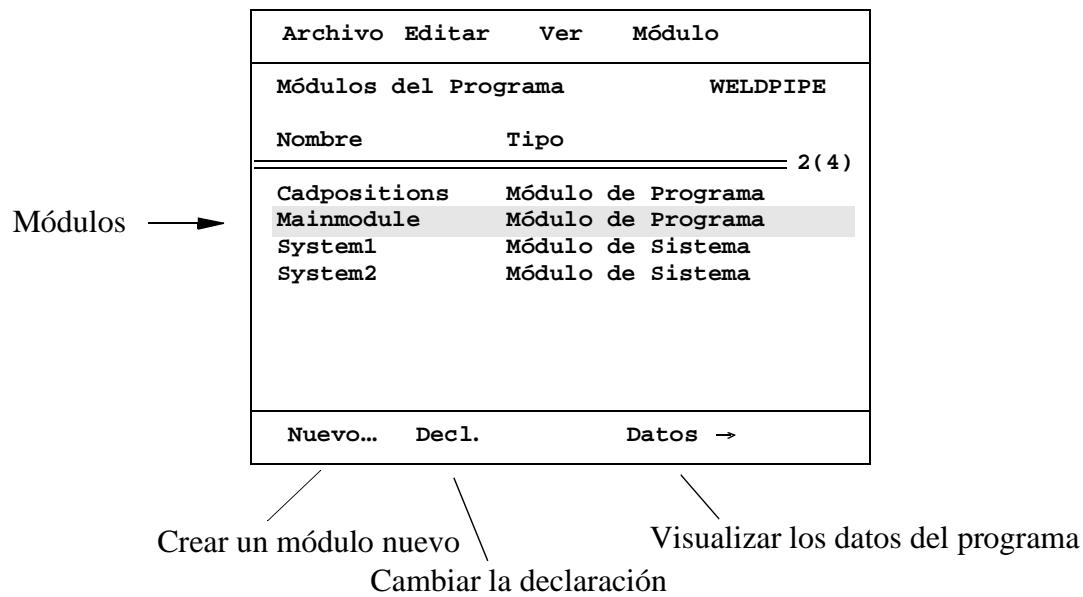


3.7.1 Menú: Test

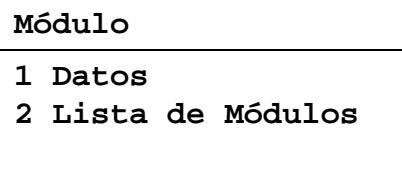


<u>Comando:</u>	<u>Sirve para:</u>
Mover el Cursor al Punto de arranque	arrancar a partir de la última instrucción parada
Mover el Punto de arranque al Cursor	arrancar a partir de la instrucción seleccionada
Mover el Punto de arranque a Main	arrancar a partir de la rutina principal main
Mover el Punto de arranque a la rutina	arrancar a partir de cualquier rutina
Ir a la posición seleccionada	ir a la posición seleccionada
Simular	permitir la ejecución del programa en el modo MOTORES OFF

3.8 Ventana: Módulos del Programa



3.8.1 Menú: Módulo



Comando:

Datos

Lista de Módulos

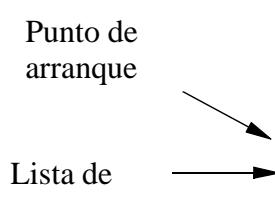
Sirve para:

visualizar los datos del programa

visualizar la lista completa de módulos

4 La Ventana de Producción

4.1 Ventana: Funcionamiento en Producción



Punto de arranque

Lista de programa

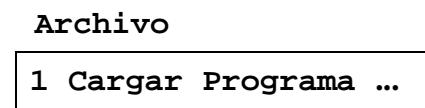
```

Archivo Editar Ver
Info Producción CAR_LIN1
Rutina : main
Estado : PARADO
Velocidad : 750 %
Modo Ejecución : Continua
=====
MoveL p1, v500, z20, tool1;
>> MoveL p2, v500, z20, tool1;
MoveL p3, v500, z20, tool1;
Set do1;
Set do2;

Iniciar FWD BWD

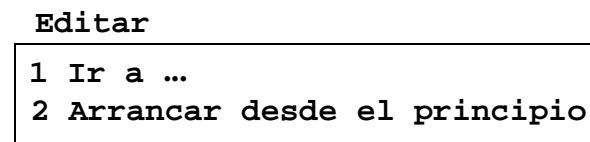
```

4.1.1 Menú: Archivo



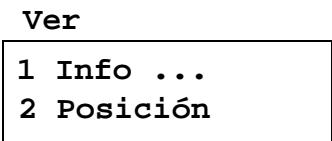
<u>Comando</u>	<u>Sirve para:</u>
Cargar Programa	cargar un programa

4.1.2 Menú: Editar



<u>Comando</u>	<u>Sirve para:</u>
Ir a	ir a una instrucción específica
Arrancar desde el principio	ir a la primera instrucción del programa

4.1.3 Menú: Ver



Comando

Info

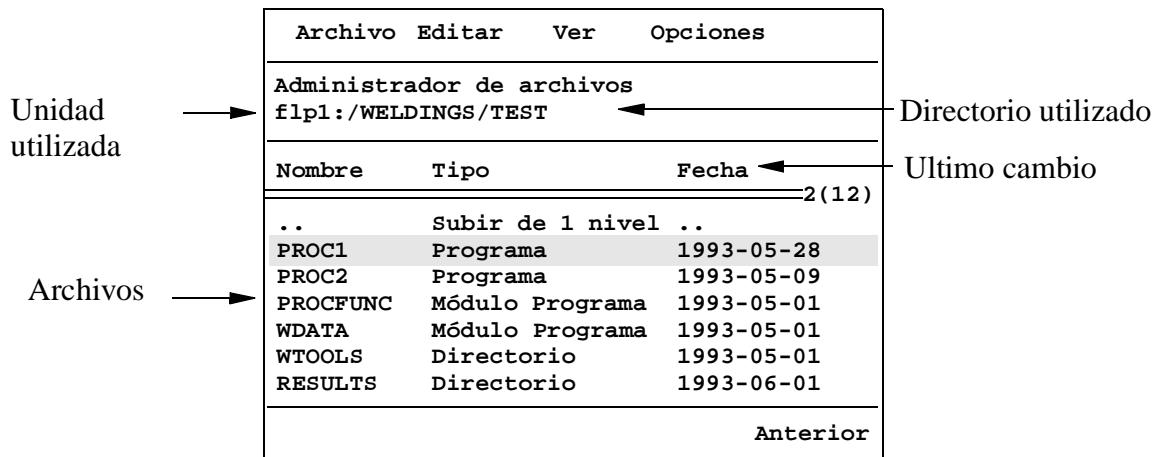
Posición

Sirve para visualizar:

el programa en la parte inferior de la ventana
ajustar una posición

5 El Administrador de Archivos

5.1 Ventana: Administrador de Archivos



5.1.1 Menú: Archivo

Archivo
1 Nuevo Directorio ...
2 Renombrar...
3 Copiar...
4 Mover...
5 Imprimir directorio

<u>Comando:</u>	<u>Sirve para:</u>
Nuevo Directorio	crear un directorio nuevo
Renombrar	cambiar el nombre de un archivo seleccionado
Copiar	copiar un archivo o un directorio seleccionado a otra memoria de masa o directorio
Mover	mover un archivo seleccionado o un directorio a otra memoria de masa o directorio
Imprimir directorio	imprimir una lista en una impresora

5.1.2 Menú: Editar

Editar

- 1 Ir a ...
- 2 Ir al inicio...
- 3 Ir al Final...

Comando:

Ir a

Ir al inicio

Ir al Final

Sirve para:

ir a una línea específica de la lista

ir a la primera línea de la lista

ir a la última línea de la lista

5.1.3 Menú: Ver

Ver

- 1 [ram1disk :]
- 2 [flop1:]Disc#12

Comando:

ram1disk:

flop1:

Sirve para ver:

los archivos contenidos en el disco RAM

los archivos contenidos en un disquete

5.1.4 Menú: Opciones

Opciones

- 1 Formatear...
- 2 Conversión rápida...

Comando:

Formatear

Conversión rápida

Sirve para:

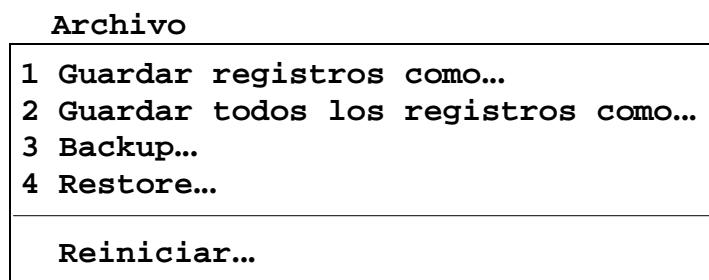
formatear un disquete

conversión de versiones antiguas de programa

6 La ventana de Servicio

6.1 Menús generales

6.1.1 Menú: Archivo

Comando**Guardar registros como**Sirve para:

guardar registros en un disquete o en otra memoria de mesa

Guardar todos los registros como

guardar todos los registros en un disquete o en otra memoria de mesa

Backup

Llevar a cabo una operación de backup

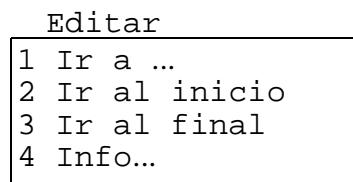
Restore

llevar a cabo una operación de restauración

Reiniciar

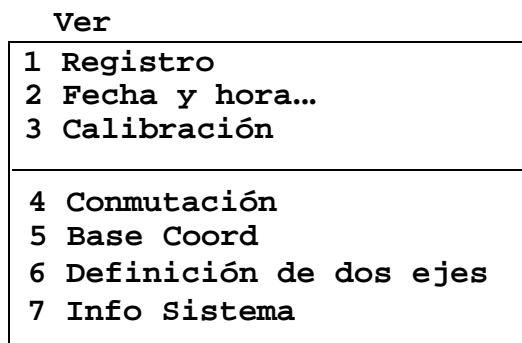
llevar a cabo un rearranque del robot

6.1.2 Menú: Editar



<u>Comando</u>	<u>Sirve para:</u>
Ir a	ir a una línea específica de la lista
Ir al inicio	ir a la primera línea de la lista
Ir al final	ir a la última línea de la lista
Info	muestra información referente a los registros de mensaje seleccionadas

6.1.3 Menú: Ver



<u>Comando</u>	<u>Sirve para:</u>
Registro	visualizar los diferentes registros
Fecha y hora	actualizar la fecha y hora
Calibración	calibrar el robot
Commutación	conmutar los motores
BaseCoord	calibrar el sistema de coordenadas de la base
Definición de dos ejes	calibrar el sistema de coordenadas de la base para un manipulador de dos ejes
Info del Sistema	visualizar la información del sistema

6.2 Ventana Registro de Servicio

The diagram shows a window titled "Registro Servicio". The menu bar at the top includes "Archivo", "Editar", "Ver", and "Especial". Below the menu is a table with columns: "Nombre", "Mensajes #", and "Ultimo". A tooltip "Lista registro" points to the left side of the table. Arrows point from the column headers "Mensajes #" and "Ultimo" to the right, with labels "No. de mensajes" and "Hora en que ocurrió el último mensaje". At the bottom of the table is a button labeled "Msjes->". An arrow points from this button to the text "Visualización de los mensajes de la lista seleccionada".

Nombre	Mensajes #	Ultimo
Común	10	0810 20:30.32
Estado	20	0810 20:25.14
Sistema	0	
Hardware	1	0810 20:30.32
Programa	0	
Movimiento	3	0810 19:15.12
Usuario	4	0809 12:30.00
Proceso	0	

6.2.1 Menú: Especial

The "Especial" menu contains three items:

- 1 Borrar Registro
- 2 Borrar todos los Registros
- 3 Renovar Registro por Evento

- | <u>Comando:</u> | <u>Sirve para:</u> |
|------------------------------------|---|
| Borrar Registro | borrar el contenido del registro seleccionado |
| Borrar todos los Registros | borrar el contenido de todos los registros |
| Renovar Registro por evento | actualizar el registro directamente cuando se produce el mensaje - el comando pasa a “ Renovar registro por comando “ cuando se selecciona, lo que significa que la lista no será actualizada hasta que se haya pulsado la tecla de función Renovar |

6.3 Ventana de Calibración de Servicio



Archivo	Editar	Ver	Calib
Servicio Calibración			
Unidad	Estado		1(4)
Robot	sincronizado		
Manip1	sincronizado		
Manip2	sincronizado		
Trackm	sincronizado		

6.3.1 Menú: Calib

Calib
1 Actualizar cont. rev...
2 Calibrar...

Comando:

Actualizar cont. rev.

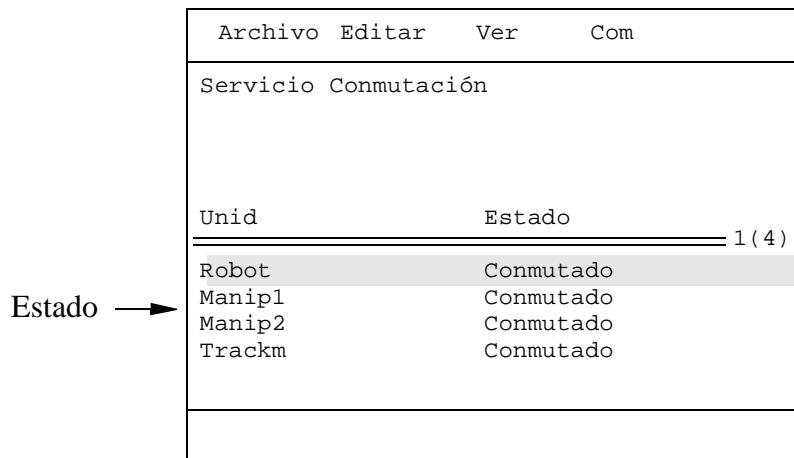
Calibrar

Sirve para:

actualizar el contador de vueltas)

calibrar utilizando el sistema de medida

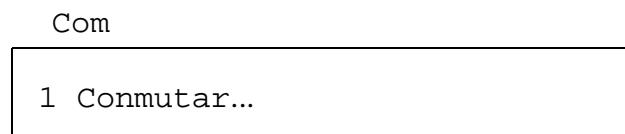
6.4 Ventana de Conmutación de Servicio



The screenshot shows a software interface titled "Servicio Conmutación". At the top is a menu bar with "Archivo", "Editar", "Ver", and "Com". Below the menu is a title "Servicio Conmutación". A table follows, with columns "Unid" and "Estado". The table has four rows, each representing a unit: "Robot", "Manip1", "Manip2", and "Trackm". All units are listed as "Conmutado". A horizontal arrow labeled "Estado" points to the "Estado" column header.

Unid	Estado
Robot	Conmutado
Manip1	Conmutado
Manip2	Conmutado
Trackm	Conmutado

6.4.1 Menú: Com



Comando:

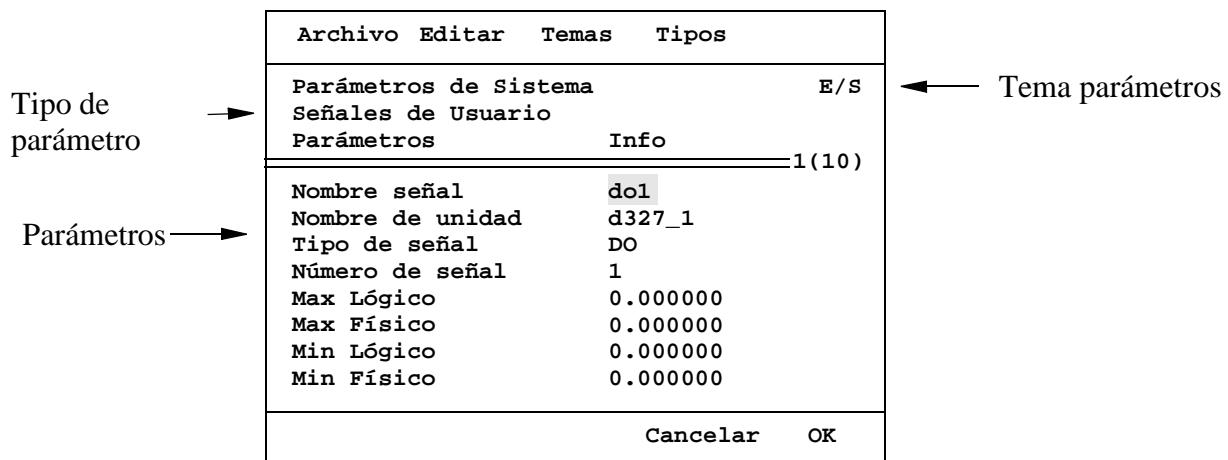
Conmutar

Sirve para:

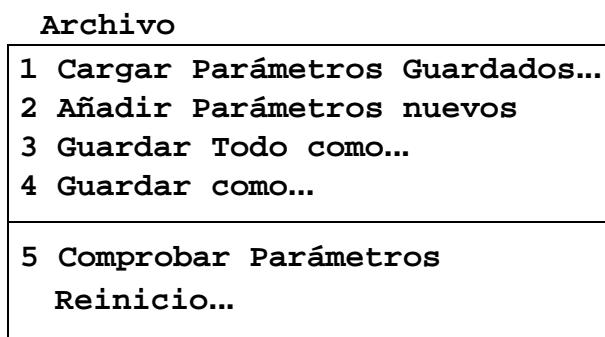
conmutar utilizando el sistema de medida

7 Los Parámetros del Sistema

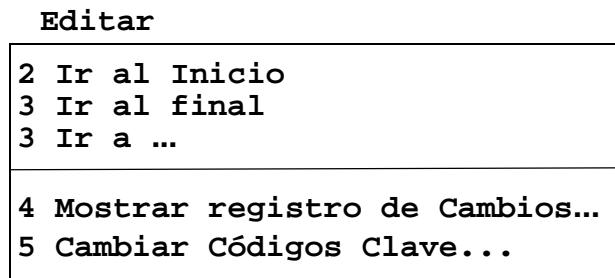
7.1 Ventana: Parámetros del Sistema



7.1.1 Menú: Archivo



<u>Comando:</u>	<u>Sirve para:</u>
Cargar Parámetros Guardados	cargar los parámetros a partir de un sistema de memoria de masa
Añadir Parámetros Nuevos	añadir los parámetros a partir de un sistema de memoria de masa
Guardar Todo como	guardar todos los parámetros en la memoria de masa
Guardar como	guardar los parámetros en la memoria de masa
Comprobar Parámetros	comprobar los parámetros antes de volver a iniciar
Reinicio	rearrancar el robot

7.1.2 Menú: EditarComando:**Ir al Inicio****Ir al final****Ir a****Mostrar registro de cambios****Cambiar Códigos Clave**Sirve para:

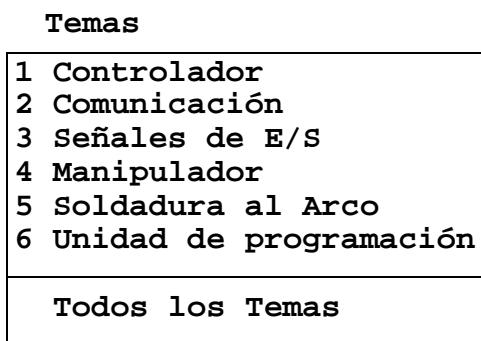
ir a la primera línea de un registro

ir a la última línea de un registro

ir a una línea específica de un registro

visualizar información referente a las modificaciones más recientes

cambiar los códigos clave

7.1.3 Menú: TemasComando:**Controlador****Comunicación****Señales de E/S****Manipulador****Soldadura al Arco****Unidad de Programación****Todos los Temas**Sirve para visualizar:

el parámetro del tema Controlador

el parámetro del tema Comunicación

los parámetros del tema E/S

los parámetros del tema del manipulador

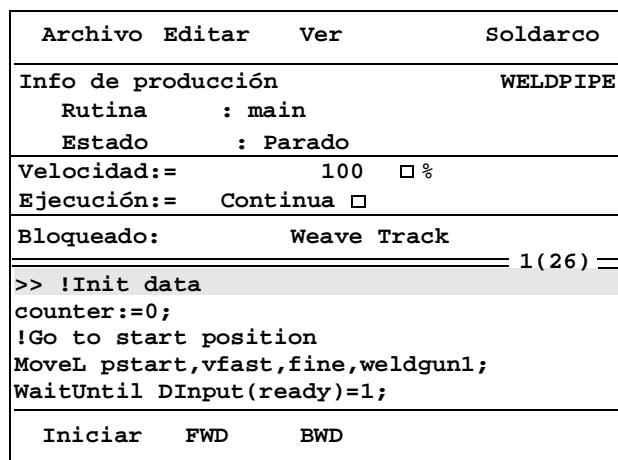
los parámetros del tema Soldadura al arco

los parámetros del tema Unidad de Programación

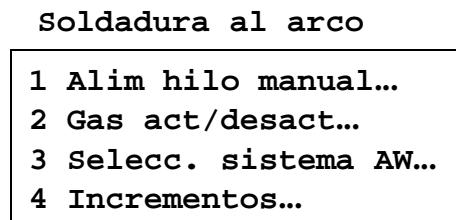
todos los temas

8 Ventana especiales ArcWare

8.1 Ventana: Producción



Menú: SoldaduraArco



<u>Comando</u>	<u>Sirve para :</u>
Alim hilo manual	alimentar el hilo hacia adelante o hacia atrás
Gas act./desact.	activar/desactivar el gas manualmente
Selecc. sistema de AW	seleccionar el sistema de soldadura al arco
Incrementos	cambiar los incrementos de ajuste

8.2 Ventana: Test del Programa

Archivo	Editar	Ver	Test	Soldarco
Test Programa		WELDPIPE/main		
Velocidad:		100%		
Ejecución:=		Continua		
Bloqueo:		Weave Track		
===== 1(26)				
>> !Init data				
counter:=0;				
Ir a la posición de inicio				
MoveL pstart,vfast,fine,weldgun1;				
Iniciar	(Modpos)	Instr >		

Menú: SoldaduraArco

Soldadura arco

- 1 Ajuste soldadura...
 - 2 Ajuste oscilación...
 - 3 Sensor...
 - 4 Bloqueo...
 - 5 Alim hilo manual...
 - 6 Gas act/desact...
 - 7 Selecc. sistema AW...
 - 8 Incrementos...

<u>Comando</u>	<u>Sirve para :</u>
Ajuste de soldadura	ajustar los datos de soldadura
Ajuste de oscilación	ajustar los datos de oscilación
Sensor	comunicar con el sensor de seguimiento de costura
Bloqueo	bloquear ciertas partes del proceso
Alim hilo manual	alimentar el hilo hacia adelante o hacia atrás
Gas act./desact.	activar/desactivar el gas manualmente
Selecc. sistema de AW	seleccionar el sistema de soldadura al arco
Incrementos	cambiar los incrementos de ajuste

8.3 Ventana de ejecución

Test	Soldaduraarco		
Ejecución Programa	WELDPIPE		
Velocidad:	100%		
Ejecución:=	Continua		
Bloqueo:	Weave Track		
Datos soldad:	welddata1		
Ajuste	Actual		1(3)
weld_speed	+ 5.5	105.5	mm/s
weld_wirefeed	- 0.15	2.30	m/min
weld_voltage	+ 3.50	33.50	Voltios
		Ajuste	

Menú: SoldaduraArco

Soldadura al arco

- 1 Ajuste soldadura...
- 2 Ajuste oscilación...
- 3 Valores de medida...

Comando:

- Ajuste de soldadura**
Ajuste de oscilación
Valores de medida

Sirve para:

- ajustar los datos de soldadura utilizados
ajustar los datos de oscilación utilizados
leer los valores de medida

INDICE

A

Abs 9-Abs-1
AccSet 8-AccSet-1
ACos 9-ACos-1
ActUnit 8-ActUnit-1
Adición 8-Add-1
agregado 5-18
alcance
 alcance de las rutinas 5-11
 alcance de los datos 5-20
AND 5-27
AOOutput 9-AOOutput-1
aplicación de adhesivo 15-GlueC-1, 15-GlueL-1
ArcC 13-ArcC-1
archivo
 cerrar 8-Close-1, 8-Rewind-1
 escritura 8-Write-1, 8-WriteBin-1
 leer 9-ReadBin-1, 9-ReadNum-1, 9-ReadStr-1
 reiniciar 8-Rewind-1
ArcL 13-ArcL-1
arco coseno 9-ACos-1
arco tangente 9-ATan-1, 9-ATan2-1
ArgName 9-ArgName-1
argument name 9-ArgName-1
aritmética 8-:=1
asignación 8-:=1
asignar un valor a un dato 3-8
ASin 9-ASin-1
ATan 9-ATan-1
ATan2 9-ATan2-1

B

backward handler 5-46
bool 7-bool-1
borrado del contenido del visualizador de la unidad de programación. 8-TPErase-1
Borrar 8-Clear-1
Break 8-Break-1
ByteToStr 9-ByteToStr-1

C

C_MOTSET 7-System data-1

C_PROGDISP 7-System data-1
cadena 5-4
cadena de texto 7-string-1
calentamiento 13-seamdata-5
CallByVar 8-CallByVar-1
canal serie
 abrir 8-Open-1
 archivo 8-WriteBin-1
 cerrar 8-Close-1
 escritura 8-Write-1
 leer 9-ReadBin-1, 9-ReadNum-1, 9-ReadStr-1
 reiniciar 8-Rewind-1
carga
 activar carga útil 8-GripLoad-1
carga útil 7-loaddata-1
 activar 8-GripLoad-1
CDate 9-CDate-1
Cerrar 8-Close-1
ClkRead 9-ClkRead-1
ClkStart 8-ClkStart-1
ClkStop 8-ClkStop-1
clock 7-clock-1
 stop 8-ClkStop-1
Comentarios 8-comment-1
comentarios 3-8, 5-5
comodines 5-5
Compact IF 8-Compact IF-1
componente de un registro 5-18
comunicación 3-39
condición 8-IF-1
conexiones enlazadas 6-41
confdata 7-confdata-1
configuración de los ejes 6-32
configuración del robot 6-32
ConfJ 8-ConfJ-1
ConfL -ConfL-1
CONNECT 8-CONNECT-1
CONST 5-22
constante 5-20
control de la configuración 8-ConfJ-1, -ConfL-1
convenciones tipográficas 2-4
CorrClear 8-CorrClear-1
CorrCon 8-CorrCon-1
CorrDiscon 8-CorrDiscon-1
CorrRead 9-CorrRead-1

CorrWrite 8-CorrWrit-1
 Cos 9--Cos-1
 CPos 9-CPos-1
 CRobT 9-CRobT-1
 cronómetro 7-clock-1, 8-ClkStart-1
 CTime 9-CTime-1
 cuaternion 7-orient-2

D

datos 5-20
 utilizados en expresiones 5-28
 datos de carga 7-loaddata-1
 datos de programa 5-20
 datos de rutina 5-20
 datos del sistema 7-System data-1
 DeactUnit 8-DeActUnit-1
 declaración
 dato constante 5-22
 dato persistente 5-22
 dato variable 5-21
 módulo 5-9
 rutina 5-13
 Decr 8-Decr-1
 desplazamiento
 posición 9-Offs-1
 desplazamiento de programa
 activar 8-PDispOn-1
 desactivar 8-PDispOff-1
 eliminar de la posición 9-ORobT-1
 desplazamiento de un programa 3-12
 Dim 9-Dim-1
 dionum 7-dionum-1
 disminución 8-Decr-1
 disminución de la velocidad 8-VelSet-1
 DIV 5-26
 DOutput 9-DOutput-1

E

ejecución concurrente 6-28
 ejecución hacia atrás 5-39
 ejes externos
 activar 8-ActUnit-1
 coordinados 6-7
 desactivar 8-DeActUnit-1
 ejes externos coordinados 6-7
 encabezado de archivo 5-6
 encendido 13-seamdata-3
 EOffsOff 8-EOffsOff-1

EOffsOn 8-EOffsOn-1
 EOffsSet 8-EOffsSet-1
 ERRNO 5-34, 7-System data-1
 errnum 7-errnum-1
 ErrWrite 8-ErrWrite-1
 escribir
 en la unidad de programación 8-TP-Write-1
 espera
 activación de una entrada digital 8-Wait-DI-1
 activación de una salida digital 8-Wait-DO-1
 esperar
 cualquier condición 8-WaitUntil-1
 hasta que el robot esté en posición 8-WaitTime-1
 un tiempo específico 8-WaitTime-1
 etiqueta 8-label-1
 EulerZYX 9-EulerZYX-1
 EXIT 8-EXIT-1
 ExitCycle 8-ExitCycle-1
 Exp 9-Exp-1
 exponential value 9-StrToByt-1
 expresión 5-26
 expresión de cadena 5-27
 expresiones aritméticas 5-26
 expresiones lógicas 5-27
 extjoint 7-extjoint-1

F

fase final 13-seamdata-6
 fecha 9-CDate-1
 file
 abrir 8-Open-1
 spystart 7-shape-1
 unload 8-UnLoad-1
 write 8-WriteStrBin-1
 fino 7--zonedata-1
 FOR 8-FOR-1
 función 5-11

G

Gestor de ejecución hacia atrás 11-5
 gestor de ejecución hacia atrás 5-43
 gestor de errores 5-34
 GetTime 9-GetTime-1
 global

dato 5-20
rutina 5-11
GlueC 15-GlueC-1
GlueL 15-GlueL-1
GlueWare 3-37
GOTO 8-GOTO-1
GOutput 9-GOutput-1
GripLoad 8-GripLoad-1
grupo de E/S 8-SetGO-1, 9-GOutput-1
gundata 14-gundata-1, 15-ggundata-1

H

hora 9-CTime-1, 9-GetTime-1

I

IDelete 8-IDelete-1
identificadores 5-3
IDisable 8-IDisable-1
IEnable 8-IEnable-1
IF 8-Compact IF-1, 8-IF-1
Incr 8-Incr-1
incremento 8-Incr-1
IndAMove 8-IndAMove-1
IndCMove 8-IndCMove-1
IndDMove 8-IndRMove-1
independent inpos 9-IndSpeed-1
independent motion 8-IndCMove-1, 8-In-
dRMove-1
IndInpos 9-IndInpos-1, 9-IndSpeed-1
IndReset 8-IndReset-1
IndRMove 8-IndRMove-1
IndSpeed 9-IndSpeed-1
inpos independiente 9-IndInpos-1
instrucciones de búsqueda 3-15
instrucciones de comprobación de la config-
uración 3-11
instrucciones de desplazamiento 3-12
instrucciones de entrada 3-19
instrucciones de espera 3-8
instrucciones de movimiento 3-15
instrucciones de posicionamiento 3-15
instrucciones de salida 3-19
instrucciones del flujo del programa 3-6
instrucciones matemáticas 3-29, 3-41
instrucciones para determinar las caracterís-
ticas de movimiento 3-10
instruções de tempo 3-28
interpolación 6-13

interpolación circular 6-15
interpolación de trayectoria esquina 6-16
interpolación lineal 6-14
interpolación lineal modificada 6-16
Interrupción
 borrar 8-IDelete-1
interrupción 3-24
 a partir de una entrada digital 8-ISignal-
 DI-1
 a partir de una salida digital 8-ISignal-
 DO-1
 activación 8-IWatch-1
 conexión 8-CONNECT-1
 desactivación 8-ISleep-1
 habilitación 8-IEnable-1
 identidad 7-intnum-1
 inabilitación 8-IDisable-1
 temporizada 8-ITimer-1
interrupciones 5-36
INTNO 7-System data-1
intnum 7-intnum-1
InvertDO 8-InvertDO-1
IO unit
 disable 8-IODisable-1
 enable 8-IOEnable-1
iodev 7-iodev-1
IODisable 8-IODisable-1
IOEnable 8-IOEnable-1
ISignalDI 8-ISignalDI-1
ISignalDO 8-ISignalDO-1
ISleep 8-ISleep-1
ITimer 8-ITimer-1
IvarValue 8-IVarVal-1
IWatch 8-IWatch-1

J

jinterpolación eje a eje 6-13
jointtarget 7-o_jointtarget-1

L

lectura
 tecla de función 8-TPReadFK-1
leer
 archivo 9-ReadBin-1, 9-ReadNum-1, 9-
 ReadStr-1
 canal serie 9-ReadBin-1, 9-ReadNum-1,
 9-ReadStr-1
 fecha actual 9-CDate-1

grupo de salidas 9-GOutput-1
hora actual 9-CTime-1, 9-GetTime-1
posición del robot utilizado 9-CRobT-1
reloj 9-ClkRead-1
salida digital 9-DOutput-1
llamada a función 5-29
llamada a otra rutina 3-6
llamada de un procedimiento nuevo 8-Proc-Call-1
llenado del cráter 13-seamdata-7
lmovimiento lineal 8-MoveL-1
Load 8-Load-1
local
 dato 5-20
 rutina 5-11

M

matriz 5-21
 obtener tamaño 9-Dim-1
mechanical unit 7-mecunit-1
mecunit 7-mecunit-1
MOD 5-26
módulo 5-8
 declaración 5-9
módulos del programa 5-8
módulos del sistema 5-9
motsetdata 7-motsetdata-1
MoveC 8-MoveC-1
MoveJ 8-MoveJ-1
MoveL 8-MoveL-1
movimiento
 círculo 8-MoveC-1
 eje a eje 8-MoveJ-1
 lineal 8-MoveL-1
movimiento circular 8-MoveC-1
movimiento eje a eje. 8-MoveJ-1
movimiento independiente 8-IndAMove-1
multitareas 5-42

N

NOT 5-27
num 7-num-1
número de error 5-34

O

objetos de trabajo 7-wobjdata-1
Offs 9-Offs-1

offset 9-Offs-1
Open 8-Open-1
operador
 prioridad 5-30
OR 5-27
orient 7-orient-1
OrientZYX 9-OrientZYX-1
ORobT 9-ORobT-1

P

palabras reservadas 5-3
parámetro opcional 5-12
parámetros 5-12
Paro 8-Stop-1
paro de la ejecución del programa 3-7
PathResol 8-PathResol-1
PDispOff 8-PDispOff-1
PDispOn 8-PDispOn-1
PERS 5-22
persistente 5-20
pos 7-pos-1
pose 7-pose-1
PoseInv 9-PoseInv-1
PoseMult 9-PoseMult-1
posición del robot 7-o_robtarget-1, 7-robtar-
 get-1
Pow 9-Pow-1
Presente 9-Present-1
principios de E/S 6-40
ProcCall 8-ProcCall-1
procedimiento 5-11
procedimiento nuevo
 llamar 8-ProcCall-1
programa 5-8
programación offline 11-3
programming 11-3
PulseDO 8-PulseDO-1
punto central de la herramienta 6-3
punto de paro 7--zonedata-1
punto de paso 7--zonedata-1

R

RAISE 8-RAISE-1
raíz cuadrada 9-Sqrt-1
ReadBin 9-ReadBin-1
ReadNum 9-ReadNum-1
ReadStr 9-ReadStr-1
recuperación de errores 5-34

reintento 8-RETRY-1, 8-TRYNEXT-1
 reducción de la aceleración 8-AccSet-1
 registro 5-18
 reglas de sintaxis 2-5
 Reiniciar 8-Rewind-1
 reloj
 arranque 8-ClkStart-1
 leer 9-ClkRead-1
 RelTool 9-ATan2-1, 9-DefFrame-1, 9-Op-
 Mode-1, 9-Pow-1, 9-RunMode-1
 Repetición 8-FOR-1
 repetición 8-WHILE-1
 Reset 8-Reset-1
 reset
 measuring system 8-IndReset-1
 resolución de trayectoria
 cambio 8-PathResol-1
 RestoPath 8-RestoPath-1
 RETRY 8-RETRY-1
 RETURN 8-RETURN-1
 Rewind 8-Rewind-1
 robot joint position 7-o_jointtarget-1
 robtarget 7-o_robtarget-1, 7-robtarget-1
 rutina 5-11
 declaración 5-13
 rutina de tratamiento de interrupciones 5-11
 rutina de tratamiento de interrupciones. 5-36
 rutina principal (main) 5-8

S

salida analógica
 activación 8-SetAO-1
 salida digital 9-DOutput-1
 activación 8-Set-1, 8-SetDO-1
 pulso 8-PulseDO-1
 reinicializar 8-Reset-1
 salto 8-GOTO-1
 seamdata (datos iniciales y finales) 13-seam-
 data-1
 SearchC 8-SearchC-1
 SearchL 8-SearchL-1
 serial channel
 file 8-WriteStrBin-1
 servo suave 3-12
 Set 8-Set-1
 SetAO 8-SetAO-1
 SetDO 8-SetDO-1
 SetGO 8-SetGO-1

signalai 7-signalxx-1
 signalao 7-signalxx-1
 signaldi 7-signalxx-1
 signaldo 7-signalxx-1
 signalgi 7-signalxx-1
 signalgo 7-signalxx-1
 simulated spot welding 15-GlueC-6, 15-
 GlueL-7
 Sin 9-Sin-1
 sincronización de E/S 6-27
 SingArea 8-SingArea-1
 sistema de coordenadas de desplazamiento
 6-6
 sistema de coordenadas de la base 6-3
 sistema de coordenadas de la herramienta 6-
 9
 sistema de coordenadas de la muñeca 6-9
 sistema de coordenadas del objeto 6-5, 7-
 wobjdata-1
 sistema de coordenadas del usuario 6-5, 7-
 wobjdata-1
 sistema de coordenadas mundo 6-4
 sistemas de coordenadas 6-3
 soldadura al arco 13-ArcC-1, 13-ArcL-1
 soldadura por puntos 3-31
 speeddata 7-speeddata-1
 spot weld gun data 14-gundata-1, 15-ggun-
 data-1
 spotdata 14-spotdata-1
 Sqrt 9-Sqrt-1
 StartMove 8-StartMove-1
 StopMove 8-StopMove-1
 StorePath 8-StorePath-1
 string 7-string-1
 StrToByte 9-StrToByte-1
 switch (interruptor) 5-12
 Symnum 7-symnum-1

T

Tan 9-Tan-1
 TCP 6-3
 estacionario 6-10
 TCP estacionario 6-10
 TEST 8-TEST-1
 TestDI 9-TestDI-1
 tipos de datos 5-18
 tipos de datos equivalente a 5-18
 tipos de datos igual a 5-18
 tipos de datos sin valor 5-18
 tooldata 7-tooldata-1

TPErase 8-TPErase-1
tpnum 7-tpnum-1
TPReadFK 8-TPReadFK-1
TPReadNum 8-TPReadNum-1
TPShow 8-tpshow-1
TPWrite 8-TPWrite-1
trayectoria esquina 7--zonedata-1
TriggC (Trig Circular) 8-TriggC-1
trigedata 7-trigedata-1
TriggEquip 8-TriggEquip-1
TriggInt 8-TriggInt-1
TriggIO 8-TriggIO-1
TriggJ (Trigg Joint) 8-TriggJ-1
TriggL (Trigg Linear) 8-TriggL-1
TRYNEXT 8-TRYNEXT-1
tunetype 7-tunetype-1

U

unidad de accionamiento común 8-ActUnit-1, 8-DeActUnit-1
unidad mecánica
 activar 8-ActUnit-1
 desactivar 8-DeActUnit-1
UnLoad 8-UnLoad-1
Usuario - módulo del sistema 10-3

V

valor absoluto 9-Abs-1
valor del arco seno 9-ASin-1
valor exponencial 9-Exp-1, 9-Pow-1
valor lógico 7-bool-1
valor numérico 7-num-1
valores lógicos 5-4
valores numéricos 5-4
VAR 5-21
variable 5-20
velocidad 7-speeddata-1
 disminución 8-VelSet-1
 máx. 8-VelSet-1
velocidad máxima 8-VelSet-1
VelSet 8-VelSet-1

W

WaitDI 8-WaitDI-1
WaitDO 8-WaitDO-1
WaitTime 8-WaitTime-1
WaitUntil 8-WaitUntil-1

weavedata (datos de oscilación) 13-weave-data-1
welldata (datos de soldadura) 13-welldata-1

WHILE 8-WHILE-1
wobjdata 7-wobjdata-1
Write 8-Write-1
write
 on the teach pendant 8-tpshow-1
WriteBin 8-WriteBin-1
WriteStrBin 8-WriteStrBin-1

X

XOR 5-27

Z

zonedata 7--zonedata-1

Glosario

Argumento	Son las partes de una instrucción que pueden ser cambiadas, es decir, todo excepto el nombre de la instrucción.
Modo automático	Es el modo utilizado cuando el selector de modo de operación está puesto en la posición  .
Componente	Es una parte de un registro.
Configuración	Es la posición de los ejes del robot en un lugar particular.
Constante	Son los datos que sólo pueden ser cambiados manualmente.
Trayectoria esquina	Es la trayectoria generada cuando el robot pasa de forma aproximada por un punto (punto de paso).
Declaración	Es la parte de una rutina o de un dato que define sus propiedades.
Diálogo/Ventana de diálogo	Cualquier ventana de diálogo que aparece en el visualizador de la unidad de programación deberá siempre ser cerrada (normalmente pulsando la tecla OK o Cancelar) antes de salir de ella.
Gestor de errores	Una parte separada de una rutina donde se puede tratar de solucionar un error. La ejecución normal podrá entonces ser rearrancada automáticamente.
Expresión	Es una secuencia de datos con sus operaciones asociadas; por ejemplo, <i>reg1+5</i> or <i>reg1>5</i> .
Punto de paso	Es un punto por cuya vecindad el robot pasa de forma aproximada -sin detenerse. La distancia a este punto depende del tamaño de la zona programada.
Función	Es una rutina que devuelve un valor.
Grupo de señales	Es un grupo de señales digitales que están agrupadas y que son manipuladas como una señal.
Interrupción	Es un acontecimiento que interrumpe temporalmente la ejecución del programa y que ejecuta una rutina de tratamiento de interrupción.
E/S	Entradas y Salidas eléctricas.
Rutina main	Es la rutina principal con la que el sistema suele arrancar cuando se pulsa la tecla Iniciar .
Modo manual	Es el modo utilizado cuando el selector de modo de operación está en la posición  .
Unidad mecánica	Es un grupo de ejes externos.
Módulo	Es un grupo de rutinas y datos, es decir, una parte del programa.
Motores ON/OFF	Indica el estado del robot, es decir, si se aplica o no tensión a los motores.
Panel de Control	Es el panel situado en la parte delantera del sistema de control.

Glosario

Orientación	Es la orientación del dispositivo final del manipulador.
Parámetro	Son los datos de entrada de una rutina enviados con la llamada de la rutina. Corresponde al argumento de una instrucción.
Persistente	Una variable cuyo valor es persistente.
Procedimiento	Es una rutina que no devuelve ningún valor.
Programa	Es el conjunto de instrucciones y de datos que define la tarea del robot. Los programas no contienen módulos del sistema.
Datos del programa	Son los datos a los que se puede acceder en un módulo completo o en el programa completo.
Módulo de programa	Es un subconjunto, definido por el usuario, de rutinas y/o datos que forman parte del programa robot.
Registro	Es un conjunto de tipos de datos.
Rutina	Es un subprograma.
Datos de rutina	Son datos locales que sólo pueden ser usados en una rutina.
Punto de arranque	Es la primera instrucción que se ejecutará al arrancar la ejecución del programa.
Punto de paro	Es el punto donde el robot se para antes de proseguir al punto siguiente.
Módulo de sistema	Es un módulo que siempre está presente en la memoria del programa. Cuando se lee un nuevo programa, los módulos del sistema permanecen en la memoria del programa.
Parámetros del sistema	Son los valores que definen el equipo del robot y sus propiedades; dicho de otra forma, son los datos de configuración del sistema.
Punto Central de la herramienta (TCP)	Es el punto, generalmente situado en el extremo de la herramienta, que sigue la trayectoria programada a la velocidad programada.
Rutina de tratamiento de interrupción (trap)	Es la rutina que define lo que hay que hacer cuando ocurre una interrupción específica.
Variable	Son los datos que pueden ser cambiados dentro de un programa, pero que pierden su valor (es decir regresa a su valor inicial) cuando se arranca un programa desde el principio.
Ventana	El robot puede ser programado y operado gracias a una serie de ventanas diferentes, como la ventana de Programa y la ventana de Servicio. Siempre se podrá salir de una ventana seleccionando otra ventana.
Zona	Es el espacio esférico que rodea un punto de paso. En cuanto el robot entra en esta zona, empieza a dirigirse hacia la posición siguiente.