

Manual de referencia técnica

Descripción general de RAPID

Power and productivity
for a better world™



Trace back information:
Workspace R15-2 version a8
Checked in 2015-10-01
Skribenta version 4.6.081

Manual de referencia técnica

Descripción general de RAPID

RobotWare 6.02

ID de documento: 3HAC050947-005

Revisión: B

La información de este manual puede cambiar sin previo aviso y no puede entenderse como un compromiso por parte de ABB. ABB no se hace responsable de ningún error que pueda aparecer en este manual.

Excepto en los casos en que se indica expresamente en este manual, ninguna parte del mismo debe entenderse como una garantía por parte de ABB por las pérdidas, lesiones, daños materiales, idoneidad para un fin determinado ni garantías similares.

ABB no será en ningún caso responsable de los daños accidentales o consecuentes que se produzcan como consecuencia del uso de este manual o de los productos descritos en el mismo.

Se prohíbe la reproducción o la copia de este manual o cualquiera de sus partes si no se cuenta con una autorización escrita de ABB.

Usted puede obtener copias adicionales de este manual a través de ABB.

El idioma original de esta publicación es el inglés. Cualquier otro idioma suministrado ha sido traducido del inglés.

© Copyright 2004-2015 ABB. Reservados todos los derechos.

ABB AB
Robotics Products
Se-721 68 Västerås
Suecia

Contenido

Descripción general de este manual	7
Cómo leer este manual	9
1 Programación básica en RAPID	11
1.1 Estructura del programa	11
1.1.1 Introducción	11
1.1.2 Elementos básicos	13
1.1.3 Módulos	17
1.1.4 Módulo de sistema <i>User</i>	20
1.1.5 Rutinas	21
1.2 Datos de programa	28
1.2.1 Tipos de datos	28
1.2.2 Declaraciones de datos	30
1.3 Expresiones	36
1.3.1 Tipos de expresiones	36
1.3.2 Utilización de datos en expresiones	39
1.3.3 Utilización de agregados en expresiones	40
1.3.4 Utilización de llamadas a funciones en las expresiones	41
1.3.5 Prioridad entre operadores	43
1.3.6 Sintaxis	44
1.4 Instrucciones	46
1.5 Control del flujo del programa	47
1.6 Otras instrucciones	49
1.7 Parámetros de movimiento	52
1.8 Movimiento	57
1.9 Señales de entrada y salida	66
1.10 Comunicaciones	69
1.11 Interrupciones	74
1.12 Recuperación en caso de error	79
1.13 UNDO	83
1.14 Sistema y tiempo	86
1.15 Matemáticas	88
1.16 Comunicación con un ordenador externo	91
1.17 Funciones para operaciones con archivos	92
1.18 Instrucciones de soporte de RAPID	93
1.19 Calibración y servicio	97
1.20 Funciones para cadenas de caracteres	99
1.21 Multitarea	101
1.22 Ejecución hacia atrás	108
2 Programación de movimiento y E/S	113
2.1 Sistemas de coordenadas	113
2.1.1 Punto central de la herramienta (TCP)	113
2.1.2 Sistemas de coordenadas utilizados para determinar la posición del TCP	114
2.1.3 Sistemas de coordenadas utilizados para determinar la dirección de la herramienta	121
2.2 Posicionamiento durante la ejecución del programa	124
2.2.1 Introducción	124
2.2.2 Interpolación de la posición y la orientación de la herramienta	126
2.2.3 Interpolación de trayectorias de esquina	130
2.2.4 Ejes independientes	137
2.2.5 Servo suave	140
2.2.6 Paro y reanudación	141
2.3 Sincronización con instrucciones lógicas	142
2.4 Configuración del robot	147
2.5 Modelos cinemáticos del robot	151

2.6	Supervisión del movimiento y detección de colisiones	156
2.7	Singularidades	160
2.8	Limitación optimizada de la aceleración	163
2.9	Zonas mundo	164
2.10	Principios de E/S	169
3	Glosario	173
<hr/>		
	Índice	175
<hr/>		

Descripción general de este manual

Acerca de este manual

Éste es un manual de referencia que contiene una explicación detallada del lenguaje de programación, así como de todos los tipos de datos, las instrucciones y las funciones. Este manual resulta especialmente útil a la hora de programar fuera de línea. Los usuarios sin experiencia deben empezar por *Manual del operador - IRC5 con FlexPendant*.

Utilización

Este manual debe leerse durante la fase de programación.

¿A quién va destinado este manual?

Este manual está destinado a personas que tengan cierta experiencia anterior con la programación, por ejemplo un programador de robots.

Requisitos previos

El lector debe tener cierta experiencia en programación y haber estudiado *Manual del operador - Introducción a RAPID*.

Organización de los capítulos

Este manual está organizado en los capítulos siguientes:

Capítulo	Contenido
Programación básica en RAPID	Responde a preguntas como “¿qué instrucción debo usar?” o “¿qué significa esta instrucción?”. Este capítulo describe brevemente todas las instrucciones, funciones y tipos de datos agrupados por las listas de selección de instrucciones que se usan durante la programación. También incluye un resumen de la sintaxis, lo que resulta especialmente útil a la hora de programar fuera de línea. También explica los detalles internos del lenguaje.
Programación de movimiento y E/S	Este capítulo describe los sistemas de coordenadas del robot, su velocidad y otras características de movimiento durante la ejecución.
Glosario	Un glosario para facilitar la comprensión de la información.

Referencias

Referencia	ID de documento
<i>Manual del operador - Introducción a RAPID</i>	3HAC029364-005
<i>Manual del operador - IRC5 con FlexPendant</i>	3HAC050941-005
<i>Manual de referencia técnica - Instrucciones, funciones y tipos de datos de RAPID</i>	3HAC050917-005
<i>Technical reference manual - RAPID kernel</i>	3HAC050946-001
<i>Manual de referencia técnica - Parámetros del sistema</i>	3HAC050948-005
<i>Application manual - Arc and Arc Sensor</i>	3HAC050988-001
<i>Application manual - Conveyor tracking</i>	3HAC050991-001
<i>Application manual - Controller software IRC5</i>	3HAC050798-001

Continúa en la página siguiente

Descripción general de este manual

Continuación

Referencia	ID de documento
Manual de aplicaciones - MultiMove	3HAC050961-005

Revisiones

Revisión	Descripción
-	Publicado con RobotWare 6.0.
A	Publicado con RobotWare 6.01. <ul style="list-style-type: none">• Añadida la instrucción TriggJIIOs; consulte Activación de salidas o interrupciones en posiciones concretas en la página 58.
B	Publicado con RobotWare 6.02. <ul style="list-style-type: none">• Añadidas funciones trigonométricas para el tipo de dato dnum; consulte Funciones aritméticas en la página 88.• Añadidos TriggDataCopy, TriggDataReset y TriggDataValid; consulte Activación de salidas o interrupciones en posiciones concretas en la página 58.• Añadida la instrucción SaveCfgData; consulte Guardado de datos de configuración en la página 94.

Cómo leer este manual

Convenciones tipográficas

Los programas de ejemplo se muestran siempre con el mismo formato en que se almacenan en un archivo o se imprimen. Esto difiere de lo que aparece en el FlexPendant en lo siguiente:

- Determinadas palabras de control que aparecen con una máscara en la pantalla del FlexPendant se imprimen, por ejemplo, las palabras que indican el principio y el final de una rutina.
- Las declaraciones de datos y rutinas se imprimen con el formato formal, por ejemplo, *VAR num reg1;*.

En las descripciones de este manual, todos los nombres de instrucciones, funciones y tipos de datos se escriben con fuente monoespaciada, por ejemplo: `TPWrite`. Los nombres de variables, parámetros del sistema y opciones se escriben en cursiva. Los comentarios del código de ejemplo no se traducen (aunque el manual esté traducido).

Reglas de sintaxis

Las instrucciones y funciones se describen utilizando tanto una sintaxis simplificada como una sintaxis formal. Si utiliza el FlexPendant para programar, normalmente sólo necesita conocer la sintaxis simplificada, dado que el robot asegura automáticamente que se utiliza la sintaxis correcta.

Ejemplo de sintaxis simplificada

Éste es un ejemplo de sintaxis simplificada con la instrucción `TPWrite`.

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

- Los argumentos obligatorios no se escriben entre paréntesis.
- Los argumentos opcionales se escriben entre corchetes `[]`. Estos argumentos pueden omitirse.
- Los argumentos excluyentes entre sí, es decir, los que no pueden existir a la vez en una misma instrucción, aparecen separados por una barra vertical `|`.
- Los argumentos que pueden repetirse un número arbitrario de veces aparecen entre llaves `{ }`.

El ejemplo anterior utiliza los siguientes argumentos:

- `String` es un argumento obligatorio.
- `Num`, `Bool`, `Pos`, `Orient` y `Dnum` son argumentos opcionales.
- `Num`, `Bool`, `Pos`, `Orient` y `Dnum` son excluyentes entre sí.

Ejemplo de sintaxis formal

```
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> ]
```

Continúa en la página siguiente

```
[ '\ ' Dnum' ::= ' <expression (IN) of dnum]';'
```

- Puede omitir el texto que aparece entre corchetes [].
- Los argumentos excluyentes entre sí, es decir, los que no pueden existir a la vez en una misma instrucción, aparecen separados por una barra vertical |.
- Los argumentos que pueden repetirse un número arbitrario de veces aparecen entre llaves { }.
- Los símbolos que se escriben para conseguir la sintaxis correcta aparecen entre comillas sencillas (apóstrofes) ' '.
- El tipo de dato del argumento y otras características aparecen entre símbolos de mayor y menor que: < >. Consulte la descripción de los parámetros de una rutina para obtener información más detallada.

Los elementos básicos del lenguaje y determinadas instrucciones se escriben con una sintaxis especial: EBNF. Ésta se basa en las mismas reglas, pero con algunos añadidos.

- El símbolo ::= significa *se define como*.
- El texto escrito entre símbolos de mayor y menor que < > se definen en una línea separada.

Ejemplo

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

1 Programación básica en RAPID

1.1 Estructura del programa

1.1.1 Introducción

Instrucciones

El programa está compuesto por un conjunto de instrucciones que describen la actividad del robot. Por tanto, existen instrucciones específicas para los distintos comandos, por ejemplo una para mover el robot, otra para seleccionar una salida, etc.

Por lo general, las instrucciones cuentan con un conjunto de argumentos asociados que definen qué debe ocurrir con una instrucción concreta. Por ejemplo, la instrucción utilizada para restablecer una salida contiene un argumento que define qué salida debe restablecerse, por ejemplo `Reset do5`. Estos argumentos pueden especificarse de una de las siguientes formas:

- Como un valor numérico, por ejemplo `5` ó `4.6`
- Como una referencia a un dato, por ejemplo `reg1`
- Como una expresión, por ejemplo `5+reg1*2`
- Como una llamada a una función, por ejemplo `Abs(reg1)`
- Como un valor de cadena, por ejemplo `"Producing part A"`

Rutinas

Existen tres tipos de rutinas: *procedimientos*, *funciones* y *rutinas TRAP*.

- Los procedimientos se utilizan como subprogramas.
- Las funciones devuelven un valor de un tipo concreto y se utilizan como argumento de una instrucción.
- Las rutinas TRAP proporcionan una forma de responder a las interrupciones. Una rutina TRAP puede asociarse a una interrupción determinada. Por ejemplo, al establecer una entrada, se ejecuta automáticamente si se produce dicha interrupción en concreto.

Datos

La información también puede almacenarse en datos, por ejemplo, datos de herramientas (que contienen toda la información sobre una herramienta, como su TCP y su peso) y datos numéricos (que pueden usarse, por ejemplo, para contar el número de piezas que deben procesarse). Los datos se agrupan en distintos tipos de datos que describen los distintos tipos de información, como herramientas, posiciones y cargas. Dado que es posible crear estos datos y asignarles nombres arbitrarios, no existe ningún límite en el número de datos (excepto el límite impuesto por la memoria disponible). Estos datos pueden existir de forma global en el programa o sólo localmente dentro de una rutina.

Continúa en la página siguiente

1 Programación básica en RAPID

1.1.1 Introducción

Continuación

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

- Las constantes representan valores fijos y la única forma de asignarles un nuevo valor es manualmente.
- La asignación de nuevos valores a las variables puede realizarse durante la ejecución del programa.
- Un valor persistente puede describirse como una variable “persistente”. Cuando se guarda un programa, el valor de inicialización corresponde al valor actual del valor persistente.

Otras características

Otras características del lenguaje son:

- Parámetros de rutinas
- Expresiones aritméticas y lógicas
- Gestión automática de errores
- Programas modulares
- Multitarea

En este lenguaje no se distingue entre mayúsculas y minúsculas. Por ejemplo, las letras mayúsculas y minúsculas se consideran iguales.

1.1.2 Elementos básicos

Identificadores

Los identificadores se utilizan para asignar nombres a módulos, rutinas, datos y etiquetas. Por ejemplo:

```
MODULE module_name
PROC routine_name( )
VAR pos data_name;
label_name:
```

El primer carácter de cualquier identificador debe ser una letra. Los demás caracteres pueden ser letras, dígitos o caracteres de subrayado (_).

La longitud máxima de cualquier identificador es de 32 caracteres, cada uno de los cuales es significativo. Se consideran iguales dos identificadores que se llaman igual pero que están escritos con distintas mayúsculas o minúsculas.

Palabras reservadas

Las palabras enumeradas a continuación están reservadas. Tienen un significado especial en el lenguaje RAPID y, por tanto, no pueden utilizarse como identificadores.

También existen varios nombres predefinidos para tipos de datos, datos de sistema, instrucciones y funciones, que tampoco pueden usarse como identificadores.

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	UNDO
VAR	VIEWONLY	WHILE	WITH
XOR			

Espacios y caracteres de salto de línea

El lenguaje de programación RAPID es un lenguaje con formato libre, lo que significa que es posible utilizar espacios en cualquier lugar del código, excepto dentro de:

- Identificadores
- Palabras reservadas

Continúa en la página siguiente

1 Programación básica en RAPID

1.1.2 Elementos básicos

Continuación

- Valores numéricos
- Marcadores de sustitución

Es posible utilizar caracteres de salto de línea, tabulador y salto de formulario en todos los lugares donde se permite el uso de espacios, excepto dentro de los comentarios.

Los identificadores, las palabras reservadas y los valores numéricos deben aparecer separados entre sí por un espacio, un carácter de salto de línea, un tabulador o un carácter de salto de formulario.

Valores numéricos

Los valores numéricos pueden expresarse como:

- Un entero, por ejemplo 3, -100, 3E2
- Un número con decimales, por ejemplo 3,5, -0,345, -245E-2

Los valores deben estar dentro del rango especificado por la norma ANSI IEEE 754, Norma para aritmética de coma flotante.

Valores lógicos

Los valores lógicos pueden expresarse como `TRUE` o `FALSE`.

Valores de cadena de caracteres

Los valores de cadena de caracteres son secuencias de caracteres (ISO 8859-1 (Latin-1)) y caracteres de control (caracteres no ISO 8859-1 (Latin-1) en el rango de códigos numéricos de 0 a 255). Es posible incluir códigos de caracteres, lo que permite incluir también caracteres no imprimibles (datos binarios) en las cadenas de caracteres. Las cadenas pueden contener un máximo de 80 caracteres.

Ejemplo:

```
"This is a string"  
"This string ends with the BEL control character \07"
```

Si se desea incluir una barra invertida (que se usa para indicar códigos de carácter) o un carácter de comillas, es necesario escribirlos dos veces.

Ejemplo:

```
"This string contains a "" character"  
"This string contains a \\ character"
```

Comentarios

Los comentarios se utilizan para facilitar la comprensión del programa. No afectan de ninguna forma al significado del programa.

Los comentarios comienzan con el signo de exclamación (!) y terminan con un carácter de salto de línea. Ocupan toda una línea y no pueden aparecer fuera de la declaración de un módulo.

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

Continúa en la página siguiente

Marcadores de sustitución

Los marcadores de sustitución pueden usarse para representar temporalmente partes del programa todavía no definidas. Los programas que contienen marcadores de sustitución son correctos sintácticamente y pueden cargarse en la memoria de programas.

Marcador de sustitución	Descripción
<TDN>	Definición de tipo de dato
<DDN>	Declaración de datos
<RDN>	Declaración de rutina
<PAR>	Parámetro alternativo opcional formal
<ALT>	Parámetro opcional formal
<DIM>	Dimensión de matriz formal (conformada)
<SMT>	Instrucción
<VAR>	Referencia a un objeto de datos (variable, variable persistente o parámetro)
<EIT>	Cláusula Else IF de una instrucción IF
<CSE>	Cláusula Case de una instrucción de test
<EXP>	Expresión
<ARG>	Argumento de llamada a procedimiento
<ID>	Identificador

Encabezado de archivo

Los archivos de programa pueden comenzar con el encabezado de archivo siguiente (aunque no es obligatorio):

```
%%%  
  VERSION:1  
  LANGUAGE:ENGLISH  
%%%
```

Sintaxis

Identificadores

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

Valores numéricos

```
<num literal> ::=  
  <integer> [ <exponent> ]  
  | <decimal integer> ) [ <exponent> ]  
  | <hex integer> | <octal integer>  
  | <binary integer>  
  | <integer> '.' [ <integer> ] [ <exponent> ]  
  | [ <integer> ] '.' <integer> [ <exponent> ]  
<integer> ::= <digit> {<digit>}  
<hex integer> ::= '0' ('X' | 'x')  
<hex digit> {<hex digit>}  
<octal integer> ::= '0' ('O' | 'o') <octal digit> {<octal digit>}
```

Continúa en la página siguiente

1 Programación básica en RAPID

1.1.2 Elementos básicos

Continuación

```
<binary integer> ::= '0' ('B' | 'b') <binary digit> {<binary digit>}  
<exponent> ::= ('E' | 'e') ['+' | '-'] <integer>  
  
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9  
<hex digit> ::= <digit> | A | B | C | D | E | F | a | b | c | d |  
e | f  
<octal digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7  
<binary digit> ::= 0 | 1
```

Valores lógicos

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

Valores de cadena de caracteres

```
<string literal> ::= '"' {<character> | <character code> } '"'  
<character code> ::= '\' <hex digit> <hex digit>  
<hex digit> ::= <digit> | A | B | C | D | E | F | a | b | c | d |  
e | f
```

Comentarios

```
<comment> ::= '!' {<character> | <tab>} <newline>
```

Caracteres

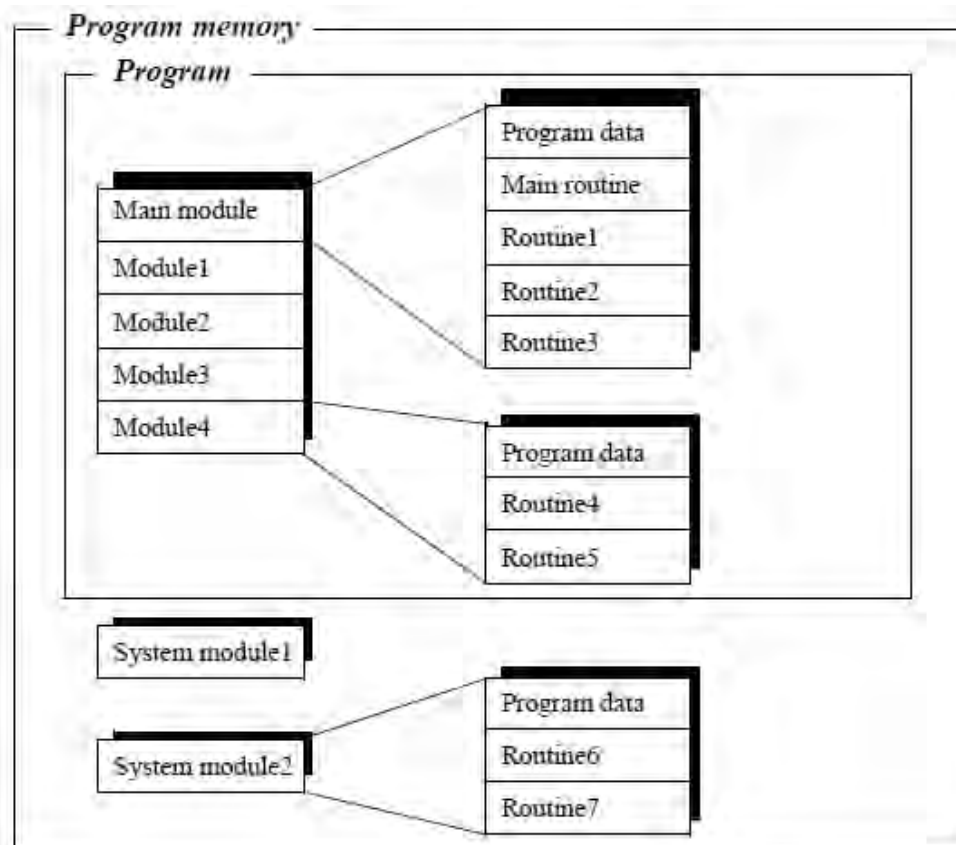
```
<character> ::= -- ISO 8859-1 (Latin-1)--  
<newline> ::= -- newline control character --  
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9  
<letter> ::= <upper case letter> | <lower case letter>  
<upper case letter> ::=  
A | B | C | D | E | F | G | H | I | J  
| K | L | M | N | O | P | Q | 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  
<lower case letter> ::=  
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) | ŷ
```

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

1.1.3 Módulos

Introducción

Los programas se dividen en *módulos de programa* y *módulos de sistema*.



xx1100000550

Módulos de programa

Un módulo de programa puede estar compuesto por datos y rutinas diferentes. Es posible copiar cada módulo, o el programa en su totalidad, a un disquete, a un disco RAM, etc., y viceversa.

Uno de los módulos contiene el procedimiento de entrada, un procedimiento global denominado *Main*. De hecho, la ejecución del programa implica la ejecución del procedimiento *Main*. El programa puede contener muchos módulos, pero sólo uno de ellos puede contener el procedimiento *main*.

Por ejemplo, un módulo puede definir la interacción con equipos externos o contener datos geométricos que se generan desde sistemas de CAD o se crean en línea mediante digitalización (programación con la unidad de programación).

Si bien las instalaciones pequeñas se suelen reflejar en un solo módulo, las instalaciones de mayor tamaño pueden tener un módulo *main* que hace referencia a rutinas y/o datos almacenados en uno o varios módulos adicionales.

Continúa en la página siguiente

1 Programación básica en RAPID

1.1.3 Módulos

Continuación

Módulos de sistema

Los módulos de sistema se utilizan para definir datos y rutinas comunes y específicos del sistema, como por ejemplo, herramientas. No se incluyen cuando se guarda un programa, lo que significa que cualquier cambio que se haga en un módulo de sistema afectará a todos los programas almacenados actualmente en la memoria de programas o que se carguen posteriormente en ella.

Declaraciones de módulos

La declaración de un módulo especifica el nombre y los atributos del módulo. Estos atributos sólo pueden añadirse fuera de línea, no a través del FlexPendant. A continuación aparecen algunos ejemplos de los atributos de un módulo:

Atributo	Si se especifica
SYSMODULE	El módulo es un módulo de sistema (si no se indica, es un módulo de programa)
NOSTEPIN	No se permite la activación del módulo durante la ejecución paso a paso
VIEWONLY	No se permite la modificación del módulo
READONLY	No se permite la modificación del módulo, pero sí la eliminación del atributo
NOVIEW	No se permite la visualización del módulo, sino solamente su ejecución. Las rutinas globales pueden utilizarse desde otros módulos y se ejecutan siempre como NOSTEPIN. Los valores de los datos globales pueden usarse en cada momento desde otros módulos o desde la ventana de datos del FlexPendant. El atributo NOVIEW sólo puede definirse fuera de línea desde un PC.

Por ejemplo:

```
MODULE module_name (SYSMODULE, VIEWONLY)
    !data type definition
    !data declarations
    !routine declarations
ENDMODULE
```

No se permite que un módulo tenga el mismo nombre que otro módulo o una rutina o un dato globales.

Estructura de archivos del programa

Como se indica arriba, todos los módulos de programa están contenidos en un programa con un nombre de programa concreto. Al guardar un programa en el disco flash o la memoria de almacenamiento, se crea un nuevo directorio con el nombre del programa. En este directorio se guardan todos los módulos del programa, con la extensión .mod, junto con un archivo de descripción que tiene el mismo nombre que el programa y la extensión .pgf. El archivo de descripción contendrá una lista con todos los módulos contenidos en el programa.

Sintaxis

Declaración de módulos

```
<module declaration> ::=
    MODULE <module name> [ <module attribute list> ]
```

Continúa en la página siguiente

```
<type definition list>
<data declaration list>
<routine declaration list>
ENDMODULE
<module name> ::= <identifier>
<module attribute list> ::= '(' <module attribute> { ',' <module
    attribute> } ')'
<module attribute> ::=

SYSMODULE
| NOVIEW
| NOSTEPIN
| VIEWONLY
| READONLY
```



Nota

Si se utilizan dos o más atributos, deben aparecer en el orden indicado anteriormente. El atributo **NOVIEW** sólo puede especificarse por sí solo o junto con el atributo **SYSMODULE**.

```
<type definition list> ::= { <type definition> }
<data declaration list> ::= { <data declaration> }
<routine declaration list> ::= { <routine declaration> }
```

1 Programación básica en RAPID

1.1.4 Módulo de sistema *User*

1.1.4 Módulo de sistema *User*

Introducción

Con el fin de simplificar la programación, el robot se suministra con datos predefinidos. No es necesario crear estos datos y, por tanto, pueden usarse directamente.

Si se utilizan estos datos, la programación inicial resulta más sencilla. Sin embargo, suele ser buena idea asignar sus propios nombres a los datos que use, con el fin de facilitar la lectura del programa.

Contenido

User comprende cinco datos numéricos (registros), un dato de objetos de trabajo, un reloj y dos valores simbólicos para señales digitales.

Nombre	Tipo de dato	Declaración
reg1	num	VAR num reg1:=0
reg2	.	.
reg3	.	.
reg4	.	.
reg5	num	VAR num reg5:=0
clock1	clock	VAR clock clock1

User es un módulo de sistema, lo que significa que siempre está presente en la memoria del robot, con independencia de qué programa esté cargado.

1.1.5 Rutinas

Introducción

Existen tres tipos de rutinas (subprogramas): procedimientos, funciones y rutinas TRAP.

- Los procedimientos no devuelven ningún valor y se utilizan en el contexto de las instrucciones.
- Las funciones devuelven un valor de un tipo concreto y se utilizan en el contexto de las expresiones.
- Las rutinas TRAP proporcionan una forma de responder a las interrupciones. Las rutinas TRAP pueden asociarse a una interrupción determinada. Cuando posteriormente se produce dicha interrupción, se ejecuta automáticamente la rutina TRAP correspondiente. Las rutinas TRAP no pueden ejecutarse explícitamente desde el programa.

Ámbito de una rutina

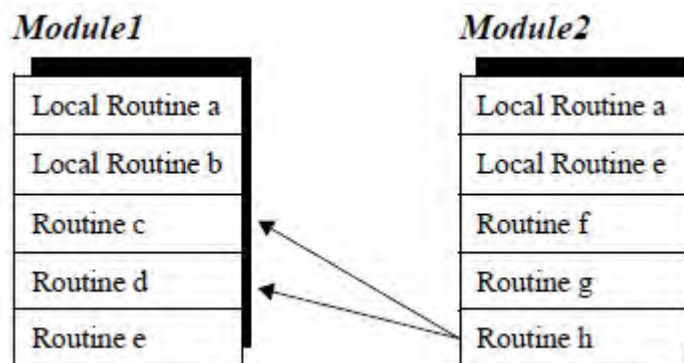
El ámbito de una rutina es el área dentro de la cual es visible la rutina. La inclusión de la directiva opcional `local` en una declaración de rutina hace que ésta sea local (dentro del módulo). De lo contrario, la rutina se considera global.

Ejemplo:

```
LOCAL PROC local_routine (...
PROC global_routine (...
```

A las rutinas se les aplican las siguientes reglas de ámbito:

- El ámbito de una rutina global puede incluir cualquier módulo de la tarea.
- El ámbito de una rutina local comprende el módulo al que pertenece.
- Dentro de su ámbito, las rutinas locales suponen la ocultación de rutinas globales o datos que tengan el mismo nombre.
- Dentro de su ámbito, las rutinas suponen la ocultación de instrucciones, rutinas predefinidas y datos que tengan el mismo nombre.



xx1100000551

En el ejemplo anterior, las rutinas siguientes están disponibles desde la rutina h:

- Módulo1: Rutina c, d.
- Módulo2: Todas las rutinas.

Continúa en la página siguiente

1 Programación básica en RAPID

1.1.5 Rutinas

Continuación

Una rutina no puede tener el mismo nombre que otra rutina, otro dato u otro tipo de dato del mismo módulo. No se permite que una rutina global tenga el mismo nombre que un módulo o una rutina global, un dato global o un tipo de dato global de otro módulo.

Parámetros

La lista de parámetros de la declaración de una rutina especifica los argumentos (los parámetros reales) que deben/pueden entregarse al llamar a la rutina.

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

- Normalmente, los parámetros se usan sólo como datos de entrada y se tratan como variables de la rutina. Cuando se modifica el contenido de esta variable, no cambia el argumento correspondiente.
- El parámetro `INOUT` especifica que el argumento correspondiente debe ser una variable (un entero, un elemento o un componente) o una variable persistente entera cuyo valor puede ser modificado por la rutina.
- El parámetro `VAR` especifica que el argumento correspondiente debe ser una variable (un entero, un elemento o un componente) cuyo valor puede ser modificado por la rutina.
- El parámetro `PERS` especifica que el argumento correspondiente debe ser una variable persistente entera cuyo valor puede ser modificado por la rutina.

Si se actualiza un parámetro de tipo `INOUT`, `VAR` o `PERS`, significa que, en realidad, se actualiza el argumento en sí, lo que posibilita el uso de argumentos para devolver valores a la rutina desde la que se hace la llamada.

Ejemplo:

```
PROC routine1 (num in_par, INOUT num inout_par,  
VAR num var_par, PERS num pers_par)
```

Un parámetro puede ser opcional y puede ser omitido de la lista de argumentos de la llamada a una rutina. Los parámetros opcionales se indican mediante una barra invertida (\) precediendo al parámetro.

Ejemplo:

```
PROC routine2 (num required_par \num optional_par)
```

No se permiten las referencias al valor de un parámetro opcional omitido en la llamada a una rutina. Esto significa que es necesario comprobar las llamadas a las rutinas para determinar la existencia de parámetros opcionales antes de utilizarlos.

Dos o más parámetros opcionales pueden ser excluyentes entre sí (es decir, que la declaración de uno de ellos excluye al otro), lo que significa que sólo uno de ellos puede estar presente en la llamada a una rutina. Esto se indica mediante una barra vertical (|) entre los parámetros afectados.

Ejemplo:

```
PROC routine3 (\num exclude1 | num exclude2)
```

El tipo especial, `switch`, (sólo) puede ser asignado a parámetros opcionales y proporciona una forma de usar argumentos modificadores, es decir, argumentos que sólo se especifican por nombre (no por valor). No es posible transmitir un

Continúa en la página siguiente

valor a un parámetro `switch`. La única forma de usar un parámetro `switch` es comprobar su presencia mediante la función predefinida `Present`.

Ejemplo:

```
PROC routine4 (\switch on | switch off)
...
IF Present (off ) THEN
...
ENDPROC
```

Es posible entregar matrices como argumentos. El grado de un argumento de matriz debe coincidir con el grado del parámetro formal correspondiente. La dimensión de un parámetro de matriz está conformada (marcada con *). Por tanto, la dimensión real depende de la dimensión del argumento correspondiente de la llamada a una rutina. Las rutinas pueden determinar la dimensión real de un parámetro mediante la función predefinida `Dim`.

Ejemplo:

```
PROC routine5 (VAR num pallet{*,*})
```

Finalización de una rutina

La ejecución de un procedimiento finaliza explícitamente con una instrucción `RETURN` o implícitamente cuando se alcanza el final (`ENDPROC`, `BACKWARD`, `ERROR` o `UNDO`) del procedimiento.

La evaluación de una función debe finalizar con una instrucción `RETURN`.

La ejecución de una rutina `TRAP` finaliza explícitamente con la instrucción `RETURN` o implícitamente cuando se alcanza el final (`ENDTRAP`, `ERROR` o `UNDO`) de la rutina `TRAP`. La ejecución continúa en el punto en el que se produjo la interrupción.

Continúa en la página siguiente

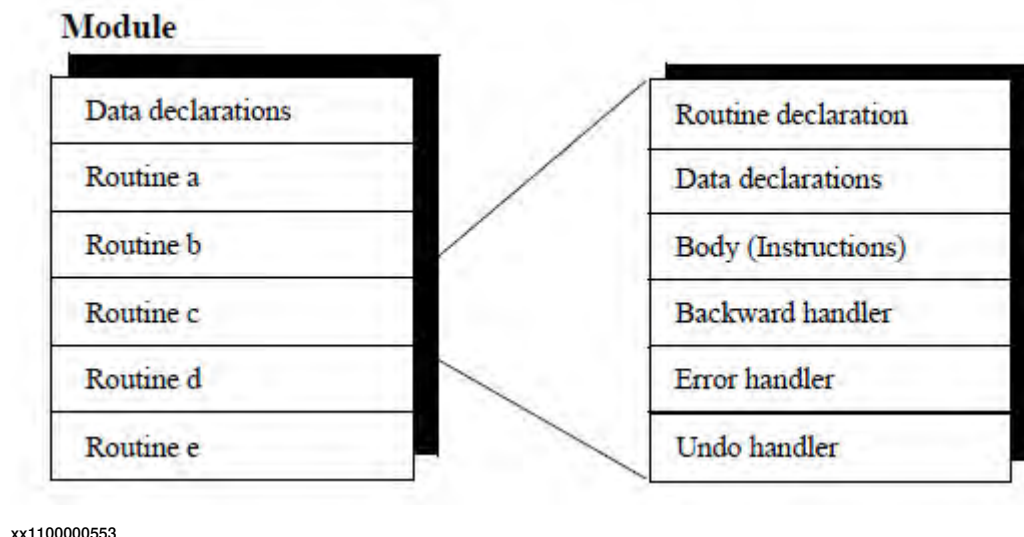
1 Programación básica en RAPID

1.1.5 Rutinas

Continuación

Declaraciones de rutinas

Las rutinas pueden contener declaraciones de rutinas (con sus parámetros), datos, un cuerpo principal, un gestor de ejecución hacia atrás (sólo en el caso de los procedimientos), un gestor de errores y un gestor de deshacer. No se permite el anidamiento de las declaraciones de rutinas, es decir, no se permite declarar una rutina dentro de una rutina.



xx1100000553

Declaración de procedimiento

Por ejemplo, multiplicar por un factor todos los elementos de una matriz de números:

```
PROC arrmul( VAR num array{*}, num factor)
  FOR index FROM 1 TO dim( array, 1 ) DO
    array{index} := array{index} * factor;
  ENDFOR
ENDPROC
```

Declaración de función

Las funciones pueden devolver valores de cualquier tipo de datos, excepto matrices.

Por 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 rutina TRAP

Por ejemplo, responder a la interrupción de alimentador vacío:

```
TRAP feeder_empty
  wait_feeder;
  RETURN;
ENDTRAP
```

Continúa en la página siguiente

Llamada a procedimiento

Cuando se realiza una llamada a un procedimiento, se utilizarán los argumentos que corresponden a los parámetros del procedimiento:

- Es necesario especificar los parámetros obligatorios. Además, es necesario especificarlos en el orden correcto.
- Puede omitir los argumentos opcionales.
- Es posible usar argumentos opcionales para transferir parámetros de una rutina a otra.

Consulte [Utilización de llamadas a funciones en las expresiones en la página 41](#).

El nombre del procedimiento puede especificarse de forma estática mediante un identificador (enlazamiento en tiempo de compilación) o especificarse en tiempo de ejecución a partir de una expresión de cadena de caracteres (enlazamiento en tiempo de ejecución). Si bien el enlazamiento en tiempo de compilación debe considerarse la forma normal de llamar a procedimientos, hay ocasiones en las que el enlazamiento en tiempo de ejecución permite crear un código muy eficiente y compacto. El enlazamiento en tiempo de ejecución se define mediante la inclusión de signos de porcentaje antes y después de la cadena de caracteres que se usa como nombre del procedimiento.

Ejemplo:

```
! early binding
TEST products_id
CASE 1:
  proc1 x, y, z;
CASE 2:
  proc2 x, y, z;
CASE 3:
  ...
! same example using late binding
% "proc" + NumToStr(product_id, 0) % x, y, z;
...
! same example again using another variant of late binding
VAR string procname {3} :=["proc1", "proc2", "proc3"];
...
% procname{product_id} % x, y, z;
...
```

Tenga en cuenta que el enlazamiento en tiempo de ejecución sólo está disponible para llamadas a procedimientos, no para llamadas a funciones. Si se hace referencia a un procedimiento desconocido mediante enlazamiento en tiempo de ejecución, la variable de sistema `ERRNO` cambia al valor `ERR_REFUNKPRC`. Si se hace referencia a un error de llamada a procedimiento (de sintaxis, no de procedimiento) mediante enlazamiento en tiempo de ejecución, la variable `ERRNO` cambia al valor `ERR_CALLPROC`.

Continúa en la página siguiente

1 Programación básica en RAPID

1.1.5 Rutinas

Continuación

Sintaxis

Declaración de rutina

```
<routine declaration> ::=
  [LOCAL] ( <procedure declaration>
    | <function declaration>
    | <trap declaration> )
  | <comment>
  | <RDN>
```

Parámetros

```
<parameter list> ::=
  <first parameter declaration> { <next parameter declaration> }
<first parameter declaration> ::=
  <parameter declaration>
  | <optional parameter declaration>
  | <PAR>
<next parameter declaration> ::=
  ',' <parameter declaration>
  | <optional parameter declaration>
  | ',' <optional parameter declaration>
  | ',' <PAR>
<optional parameter declaration> ::=
  '\' ( <parameter declaration> | <ALT> )
  { '|' ( <parameter declaration> | <ALT> ) }
<parameter declaration> ::=
  [ VAR | PERS | INOUT ] <data type>
  <identifier> [ '{' ( '*' { ',' '*' } ) | <DIM> ] '{'
  | 'switch' <identifier>
```

Declaración de procedimiento

```
<procedure declaration> ::=
  PROC <procedure name>
  '(' [ <parameter list> ] ')'
  <data declaration list>
  <instruction list>
  [ BACKWARD <instruction list> ]
  [ ERROR <instruction list> ]
  [ UNDO <instruction list> ]
  ENDPROC
<procedure name> ::= <identifier>
<data declaration list> ::= {<data declaration>}
```

Declaración de función

```
<function declaration> ::=
  FUNC <value data type>
  <function name>
  '(' [ <parameter list> ] ')'
  <data declaration list>
  <instruction list>
  [ ERROR <instruction list> ]
  [ UNDO <instruction list> ]
```

Continúa en la página siguiente

ENDFUNC

<function name> ::= <identifier>

Declaración de rutina TRAP

```
<trap declaration> ::=  
  TRAP <trap name>  
  <data declaration list>  
  <instruction list>  
  [ ERROR <instruction list> ]  
  [ UNDO <instruction list> ]  
  ENDTRAP  
<trap name> ::= <identifier>
```

Llamada a procedimiento

```
<procedure call> ::= <procedure> [ <procedure argument list> ] ';'   
<procedure> ::=  
  <identifier>  
  | '%' <expression> '%'   
<procedure argument list> ::= <first procedure argument> {  
  <procedure argument> }   
<first procedure argument> ::=  
  <required procedure argument>  
  | <optional procedure argument>  
  | <conditional procedure argument>  
  | <ARG>   
<procedure argument> ::=  
  ',' <required procedure argument>  
  | <optional procedure argument>  
  | ',' <optional procedure argument>  
  | <conditional procedure argument>  
  | ',' <conditional procedure argument>  
  | ',' <ARG>   
<required procedure argument> ::= [ <identifier> ':= ' ] <expression>  
<optional procedure argument> ::= '\ ' <identifier> [ ':= '  
  <expression> ]   
<conditional procedure argument> ::= '\ ' <identifier> '?' (  
  <parameter> | <VAR> )
```

1.2 Datos de programa

1.2.1 Tipos de datos

Introducción

Existen tres clases diferentes de tipos de datos:

- Los tipos *atómicos* son atómicos en cuanto a que no están definidos partiendo de ningún otro tipo y que no pueden ser divididos en distintas partes o distintos componentes. Un ejemplo es el tipo `num`.
- Los tipos de datos de *registro* son tipos compuestos con componentes con nombre y ordenados. Un ejemplo es `pos`. Los componentes pueden ser de tipo atómico o de registro.

Los valores de registro pueden expresarse usando una representación agregada. Por ejemplo: `[300, 500, profundidad]` valor agregado de registro `pos`.

El acceso a un componente concreto de un dato de registro puede realizarse a través del nombre del componente. Por ejemplo: `pos1.x := 300`; asignación del componente `x` de `pos1`.

- Un tipo de dato de *alias* es por definición equivalente a otro tipo. Los tipos de *Alias* permiten clasificar los objetos de datos.

Tipos de datos sin valor

Cada tipo de datos disponible es un tipo de datos *con valor* o un tipo de dato *sin valor*. En otras palabras, un tipo de dato con valor representa alguna forma de valor. Los datos sin valor no pueden usarse en operaciones basadas en valores:

- Inicialización
- Asignación (`:=`)
- Comprobaciones de igualdad (`=`) y de diferencia (`<>`)
- Instrucciones `TEST`
- Parámetros `IN` (modo de acceso) en llamadas a rutinas
- Tipos de datos de función (`return`)

Los tipos de datos de entrada (*signalai*, *signaldi*, *signalgi*) son del tipo de datos de *semivalor*. Estos datos pueden usarse en operaciones basadas en valores, excepto la inicialización y la asignación.

En la descripción de un tipo de datos, sólo se especifica cuando es un tipo de datos de semivalor o sin valor.

Tipos de datos de igualdad (alias)

Un tipo de dato de *alias* se define como equivalente a otro tipo de dato. Los datos con los mismos tipos de datos pueden usarse unos en sustitución de otros.

Ejemplo:

```
VAR num level;  
VAR dionum high:=1;  
level:= high;
```

Continúa en la página siguiente

Esto es correcto dado que `dionum` es un tipo de dato de *alias* de `num`.

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>
```

1.2.2 Declaraciones de datos

Introducción

Existen tres tipos de datos:

- La asignación de nuevos valores a las *variables* puede realizarse durante la ejecución del programa.
- Un valor *persistente* puede describirse como una variable persistente. Esto se consigue haciendo que una actualización del valor de una variable persistente provoque automáticamente la actualización del valor de inicialización de la declaración de la variable persistente. (Cuando se guarda un programa, el valor de inicialización de cualquier declaración de variable persistente refleja el valor actual de la variable persistente).
- Las *constantes* representan valores fijos y no es posible asignarles nuevos valores.

Las declaraciones de datos introducen los datos mediante la asociación de un nombre (un identificador) a un tipo de dato. Excepto en el caso de los datos predefinidos y las variables de bucle, es necesario declarar todos los datos utilizados.

Ámbito de un dato

El ámbito de un dato es el área dentro de la cual es visible el dato. La inclusión de la directiva opcional *local* en una declaración de datos hace que ésta sea local (dentro del módulo). De lo contrario, el dato se considera global. Recuerde que la directiva *local* sólo puede usarse en el nivel de módulo, no dentro de una rutina.

Ejemplo

```
LOCAL VAR num local_variable;  
VAR num global_variable;
```

Datos de programa

Los datos declarados fuera de una rutina se conocen como *datos de programa*. A los datos de programa se les aplican las siguientes reglas de ámbito:

- El ámbito de un dato de programa predefinido o global puede incluir cualquier módulo.
- El ámbito de un dato de programa local comprende el módulo al que pertenece.
- Dentro de su ámbito, los datos de programa locales suponen la ocultación de cualquier dato o rutina globales que tengan el mismo nombre (incluidas las instrucciones, las rutinas predefinidas y los datos).

No se permite que los datos de programa tengan el mismo nombre que otros datos o rutinas del mismo módulo. No se permite que los datos de programa globales tengan el mismo nombre que otros datos globales o rutinas de otros módulos.

Datos de rutina

Los datos declarados dentro de una rutina se conocen como *datos de rutina*. Recuerde que los parámetros de una rutina también se manejan como datos de rutina. A los datos de rutina se les aplican las siguientes reglas de ámbito:

- El ámbito de un dato de rutina comprende la rutina a la que pertenece.
- Dentro de su ámbito, los datos de rutina suponen la ocultación de las rutinas o los datos que tengan el mismo nombre.

No se permite que los datos de rutina tengan el mismo nombre que otros datos o etiquetas de la misma rutina.

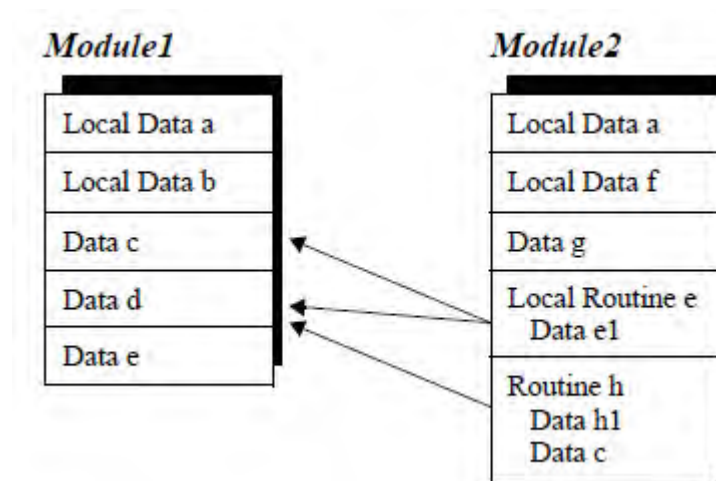
Ejemplo

En este ejemplo, los datos siguientes están disponibles desde la rutina e:

- Módulo1: Datos c, d.
- Módulo2: Datos a, f, g, e1.

Los datos siguientes están disponibles desde la rutina h:

- Módulo1: Dato d.
- Módulo2: Datos a, f, g, h1, c.



xx1100000554

Declaración de variables

Una variable es introducida mediante una declaración de variable y puede declararse como global del sistema, global de la tarea o local.

Ejemplo:

```
VAR num globalvar := 123;  
TASK VAR num taskvar := 456;  
LOCAL VAR num localvar := 789;
```

Es posible asignar un formato de matriz (de grado 1, 2 ó 3) a variables de cualquier tipo, mediante la especificación de información de dimensiones en la declaración. Las dimensiones son valores enteros mayores que 0.

Ejemplo:

```
VAR pos pallet{14, 18};
```

Continúa en la página siguiente

1 Programación básica en RAPID

1.2.2 Declaraciones de datos

Continuación

Es posible inicializar las variables con tipos de valores (es decir, indicar su valor inicial). La expresión utilizada para inicializar una variable de programa debe ser constante. Recuerde que se permite utilizar el valor de una variable no inicializada, pero ésta no está definida, es decir, tiene el valor cero.

Ejemplo:

```
VAR string author_name := "John Smith";
VAR pos start := [100, 100, 50];
VAR num maxno{10} := [1, 2, 3, 9, 8, 7, 6, 5, 4, 3];
```

El valor de inicialización se establece en las situaciones siguientes:

- Al abrir el programa
- Cuando se ejecuta el programa desde el principio

Declaración de variables persistentes

Las variables persistentes sólo pueden ser declaradas en el nivel de módulo, no dentro de una rutina. Estas variables persistentes pueden declararse como globales del sistema, globales de la tarea o locales.

Ejemplo:

```
PERS num globalpers := 123;
TASK PERS num taskpers := 456;
LOCAL PERS num localpers := 789;
```

Todas las variables persistentes globales del sistema que tengan el mismo nombre comparten el valor actual. Las variables persistentes globales de la tarea y locales no comparten el valor actual con otras variables persistentes.

Las variables persistentes locales y globales de la tarea deben recibir un valor de inicialización. En el caso de las variables persistentes globales del sistema, el valor inicial puede omitirse. El valor de inicialización debe ser un solo valor (sin referencias a datos ni operandos) o un solo agregado con miembros que, a su vez, sean valores simples o agregados simples.

Ejemplo:

```
PERS pos refpnt := [100.23, 778.55, 1183.98];
```

Es posible asignar un formato de matriz (de grado 1, 2 ó 3) a variables persistentes de cualquier tipo, mediante la especificación de información de dimensiones en la declaración. Las dimensiones son valores enteros mayores que 0.

Ejemplo:

```
PERS pos pallet{14, 18} := [...];
```

Recuerde que si cambia el valor actual de una variable persistente, el hacerlo provoca la actualización del valor de inicialización (si no se había omitido) de la declaración de la variable persistente. Sin embargo, debido a aspectos de rendimiento, esta actualización no tiene lugar durante la ejecución del programa. El valor inicial se actualiza al guardar el módulo (Copia de seguridad, Guardar módulo, Guardar programa). También se actualiza al editar el programa. La ventana de datos de programa del FlexPendant siempre muestra el valor actual de la variable persistente.

Ejemplo:

```
PERS num reg1 := 0;
...
```

Continúa en la página siguiente


```

reg1 := 5;
After module save, the saved module looks like this:
PERS num reg1 := 5;
...
reg1 := 5;

```

Declaración de constantes

Las constantes se introducen mediante declaraciones de constante. No es posible modificar el valor de las constantes.

Ejemplo:

```
CONST num pi := 3.141592654;
```

Es posible asignar un formato de matriz (de grado 1, 2 ó 3) a constantes de cualquier tipo, mediante la especificación de información de dimensiones en la declaración. Las dimensiones son valores enteros mayores que 0.

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

Inicialización de datos

El valor de inicialización de una constante o una variable puede ser una expresión constante.

El valor de inicialización de una variable persistente sólo puede ser una expresión literal.

Ejemplo:

```

CONST num a := 2;
CONST num b := 3;
!Correct syntax
CONST num ab := a + b;
VAR num a_b := a + b;
PERS num a__b := 5; !
!Faulty syntax
PERS num a__b := a + b;

```

En la tabla siguiente puede ver qué ocurre durante distintas actividades, como el reinicio, un nuevo programa, inicio del programa, etc.

El evento de sistema afecta a	Encendido (reinicio)	Abrir, cerrar o nuevo programa	Iniciar programa (mover puntero a main)	Iniciar programa (mover puntero a rutina)	Iniciar programa (mover puntero a cursor)	Iniciar programa (llamada a rutina)	Iniciar programa (después de ciclo)	Iniciar programa (después de paro)
Constante	Sin cambios	Inicialización	Inicialización	Inicialización	Sin cambios	Sin cambios	Sin cambios	Sin cambios
Variable	Sin cambios	Inicialización	Inicialización	Inicialización	Sin cambios	Sin cambios	Sin cambios	Sin cambios
Persistente	Sin cambios	Inicialización i / Sin cambios	Sin cambios	Sin cambios	Sin cambios	Sin cambios	Sin cambios	Sin cambios

Continúa en la página siguiente

1 Programación básica en RAPID

1.2.2 Declaraciones de datos

Continuación

El evento de sistema afecta a	Encendido (reinicio)	Abrir, cerrar o nuevo programa	Iniciar programa (mover puntero a main)	Iniciar programa (mover puntero a rutina)	Iniciar programa (mover puntero a cursor)	Iniciar programa (llamada a rutina)	Iniciar programa (después de ciclo)	Iniciar programa (después de paro)
Interrupciones con comando	Pedidas de nuevo	Desaparece	Desaparece	Desaparece	Sin cambios	Sin cambios	Sin cambios	Sin cambios
Rutina de puesta en marcha SYS_RESET (con parámetros de movimiento)	No se ejecuta	Se ejecuta ii	Se ejecuta	No se ejecuta	No se ejecuta	No se ejecuta	No se ejecuta	No se ejecuta
Archivos	Se cierran	Se cierran	Se cierran	Se cierran	Sin cambios	Sin cambios	Sin cambios	Sin cambios
Trayectoria	Creada de nuevo al arrancar	Desaparece	Desaparece	Desaparece	Desaparece	Sin cambios	Sin cambios	Sin cambios

i Las variables persistentes que no tienen ningún valor inicial sólo se inicializan si no se han declarado previamente.

ii Genera un error cuando hay un error semántico en el programa de la tarea actual.

Clase de almacenamiento

La *clase de almacenamiento* de un objeto de datos determina cuándo asigna y libera el sistema la memoria del objeto de datos. La clase de almacenamiento de un objeto de datos se determina por el tipo de objeto de datos y el contexto de su declaración. Puede ser *estática* o *volátil*.

Las constantes, las variables persistentes y las variables de módulo son estáticas, es decir, que tienen el mismo tipo de almacenamiento durante toda la vida de una tarea. Esto significa que cualquier valor asignado a una variable persistente o a una variable de módulo permanece siempre sin cambios hasta la siguiente asignación.

Las variables de rutina son volátiles. La memoria necesaria para almacenar el valor de una variable volátil se asigna en primer lugar con una llamada a la rutina que alberga la declaración en la que se encuentra la variable. Más tarde, la memoria se libera en el punto de retorno al lugar del programa desde el que se llama a la rutina. Esto significa que el valor de una variable de rutina está siempre sin definir antes de la llamada a la rutina y siempre se pierde (queda sin definición) al final de la ejecución de la rutina.

En una cadena de llamadas recursivas a rutinas (una rutina que se llama a sí misma de forma directa o indirecta), cada instancia de la rutina recibe una posición propia en la memoria para la misma variable de rutina (se crean varias *instancias* de la misma variable).

Continúa en la página siguiente

Sintaxis

Declaración de datos

```
<data declaration> ::=  
  [LOCAL] ( <variable declaration>  
    | <persistent declaration>  
    | <constant declaration> )  
  | TASK <persistent declaration>  
  | <comment>  
  | <DDN>
```

Declaración de variables

```
<variable declaration> ::=  
  VAR <data type> <variable definition> ';'   
<variable definition> ::=  
  <identifier> [ '{' <dim> { ',' <dim> } '}' ]  
    [ ':' <constant expression> ]  
<dim> ::= <constant expression>
```

Declaración de variables persistentes

```
<persistent declaration> ::=  
  PERS <data type> <persistent definition> ';'   
<persistent definition> ::=  
  <identifier> [ '{' <dim> { ',' <dim> } '}' ]  
    [ ':' <literal expression> ]
```



Nota

La expresión literal sólo puede omitirse en el caso de las variables persistentes globales del sistema.

Declaración de constantes

```
<constant declaration> ::=  
  CONST <data type> <constant definition> ';'   
<constant definition> ::=  
  <identifier> [ '{' <dim> { ',' <dim> } '}' ]  
    ':' <constant expression>  
<dim> ::= <constant expression>
```

1 Programación básica en RAPID

1.3.1 Tipos de expresiones

1.3 Expresiones

1.3.1 Tipos de expresiones

Descripción

Una expresión especifica la evaluación de un valor. Por ejemplo, puede usarla en las situaciones siguientes:

En instrucciones de asignación	Por ejemplo: <code>a:=3*b/c;</code>
Como una condición de una instrucción IF	Por ejemplo: <code>IF a>=3 THEN ...</code>
Como un argumento en una instrucción	Por ejemplo: <code>WaitTime time;</code>
Como un argumento en la llamada a una función	Por ejemplo: <code>a:=Abs(3*b);</code>

Expresiones aritméticas

Las expresiones aritméticas se utilizan para evaluar valores numéricos.

Ejemplo:

`2*pi*radius`

Operador	Funcionamiento	Tipos de operandos	Tipo de resultado
+	suma	num + num	num ⁱ
+	suma	dnum + num	dnum ⁱ
+	suma unitaria; conservar el signo	+num o +dnum o +pos	igual ⁱⁱ , ⁱ
+	suma de vectores	pos + pos	pos
-	resta	num - num	num ⁱ
-	resta	dnum - dnum	dnum ⁱ
-	resta unitaria; cambio de signo	-num o -pos	igual ⁱⁱ , ⁱ
-	resta unitaria; cambio de signo	-num o -dnum o -pos	igual ⁱⁱ , ⁱ
-	resta de vectores	pos - pos	pos
*	multiplicación	num * num	num ⁱ
*	multiplicación	dnum * dnum	dnum ⁱ
*	multiplicación de vectores escalares	num * pos o pos * num	pos
*	producto de vectores	pos * pos	pos
*	vinculación de rotaciones	orient * orient	orient
/	división	num / num	num
/	división	dnum / dnum	dnum
DIV ⁱⁱⁱ	división entera	num DIV num	num
DIV ⁱⁱⁱ	división entera	dnum DIV dnum	dnum

Continúa en la página siguiente

Operador	Funcionamiento	Tipos de operandos	Tipo de resultado
MOD ⁱⁱⁱ	módulo de entero; resto	num MOD num	num
MOD ⁱⁱⁱ	módulo de entero; resto	dnum MOD dnum	dnum

- ⁱ Conserva la representación de entero (exacto) siempre y cuando los operandos y el resultado se mantengan dentro del subdominio de enteros de tipo numérico.
- ⁱⁱ El resultado recibe el mismo tipo que el operando. Si el operando tiene un tipo de dato de alias, el resultado recibe el tipo "básico" del alias (num, dnum o pos).
- ⁱⁱⁱ Operaciones enteras, por ejemplo 14 DIV 4=3, 14 MOD 4=2. (No se permite el uso de operandos no enteros.)

Expresiones lógicas

Las expresiones lógicas se utilizan para evaluar valores lógicos (TRUE/FALSE).

Ejemplo:

a>5 AND b=3

Operador	Funcionamiento	Tipos de operandos	Tipo de resultado
<	menor que	num < num	bool
<	menor que	dnum < dnum	bool
<=	menor que o igual a	num <= num	bool
<=	menor que o igual a	dnum <= dnum	bool
=	igual a	cualquiera ⁱ = cualquiera	bool
>=	mayor que o igual a	num >= num	bool
>=	mayor que o igual a	dnum >= dnum	bool
>	mayor que	num > num	bool
>	mayor que o igual a	dnum > dnum	bool
<>	distinto de	cualquiera <> cualquiera	bool
AND	y	bool AND bool	bool
XOR	OR exclusivo	bool XOR bool	bool
OR	O bien	bool OR bool	bool
NOT	no unitario; negación	NOT bool	bool

- ⁱ Sólo tipos de datos con valor. Los operandos deben ser del mismo tipo.

Continúa en la página siguiente

1 Programación básica en RAPID

1.3.1 Tipos de expresiones

Continuación

a AND b			a XOR b		
a \ b	True	False	a \ b	True	False
True	True	False	True	False	True
False	False	False	False	True	False

a OR b			NOT b	
a \ b	True	False	b	
True	True	True	True	False
False	True	False	False	True

xx1100000555

Expresiones de cadena de caracteres

Las expresiones de cadena de caracteres se utilizan para realizar operaciones con cadenas de caracteres.

Ejemplo: "IN" + "PUT" genera el resultado "INPUT"

Operador	Funcionamiento	Tipos de operandos	Tipo de resultado
+	concatenación de cadenas de caracteres	string + string	string

1.3.2 Utilización de datos en expresiones

Introducción

Es posible usar variables enteras, variables persistentes o constantes como parte de una expresión.

Ejemplo:

```
2*pi*radius
```

Matrices

Es posible hacer referencia a una variable, una variable persistente o una constante declarada como matriz, ya sea como un todo o en sus distintos elementos.

Para hacer referencia a un elemento de matriz se utiliza el número de índice del elemento. El índice es un valor entero mayor que 0 y no debe ir más allá de la dimensión declarada. El valor de índice 1 se utiliza para seleccionar el primer elemento. El número de elementos de la lista de índices debe corresponder al grado declarado (1, 2 ó 3) de la matriz.

Ejemplo:

```
VAR num row{3};
VAR num column{3};
VAR num value;

! get one element from the array
value := column{3};

! get all elements in the array
row := column;
```

Registros

Es posible hacer referencia a una variable, una variable persistente o una constante declarada como registro, ya sea como un todo o en sus distintos componentes.

Para hacer referencia a un componente de un registro, se utiliza el nombre del componente.

Ejemplo:

```
VAR pos home;
VAR pos pos1;
VAR num yvalue;
..
! get the Y component only
yvalue := home.y;

! get the whole position
pos1 := home;
```

1.3.3 Utilización de agregados en expresiones

Introducción

Los agregados se utilizan con valores de registro o de matriz.

Ejemplo:

```
! pos record aggregate
pos := [x, y, 2*x];

! pos array aggregate
posarr := [[0, 0, 100], [0,0,z]];
```

Requisitos previos

Debe ser posible determinar el tipo de dato de un agregado por su contexto. El tipo de dato de cada miembro del agregado debe ser igual al tipo del miembro correspondiente del tipo determinado.

Ejemplo (tipo de agregado pos - determinado por p1):

```
VAR pos p1;
p1 :=[1, -100, 12];
```

Ejemplo de qué **no** se permite (no permitido ya que no es posible determinar el tipo de dato de ninguno de los agregados por su contexto):

```
VAR pos p1;
IF [1, -100, 12] = [a,b,b,] THEN
```


1.3.4 Utilización de llamadas a funciones en las expresiones

Introducción

La llamada a una función inicia la evaluación de una función determinada y recibe el valor devuelto por la función.

Ejemplo:

```
Sin(angle)
```

Argumentos

Los argumentos de la llamada a una función se utilizan para transferir datos a (y posiblemente de) la función a la que se llama. El tipo de dato de un argumento debe ser igual al tipo del parámetro correspondiente de la función. Se permite la omisión de los argumentos opcionales, pero el orden de los argumentos (presentes) debe ser el mismo que el orden de los parámetros formales. Además, es posible declarar dos o más argumentos opcionales que se excluyen entre sí, en cuyo caso sólo puede estar presente uno de ellos en la lista de argumentos.

Los argumentos obligatorios se separan entre sí mediante una coma “,”. El nombre formal del parámetro puede incluirse u omitirse.

Ejemplo	Descripción
<pre>Polar(3.937, 0.785398) Polar(Dist:=3.937, Angle:=0.785398)</pre>	Dos argumentos obligatorios, con o sin el nombre del parámetro.
<pre>Cosine(45) Cosine(0.785398\Rad)</pre>	Un argumento obligatorio, con o sin un modificador.
<pre>Dist(p2) Dist(\distance:=pos1, p2)</pre>	Un argumento obligatorio, con o sin un argumento opcional.

Los argumentos opcionales deben ir precedidos de una barra invertida “\” y el nombre formal del parámetro. Los argumentos de parámetro modificador son un caso algo especial. No pueden incluir expresiones de argumento. En su lugar, estos argumentos sólo pueden estar “presentes” o “ausentes”.

Los argumentos condicionales se utilizan para la propagación sin problemas de los argumentos opcionales a través de cadenas de llamadas a rutinas. Se considera que un argumento condicional está “presente” si el parámetro opcional especificado (de la función desde la que se hace la llamada) está presente. De lo contrario, se considera sencillamente como omitido. Recuerde que el parámetro especificado debe ser opcional.

Ejemplo:

```
PROC Read_from_file (iodev File \num Maxtime)
..
character:=ReadBin (File \Time?Maxtime);
! Max. time is only used if specified when calling the routine
! Read_from_file
..
ENDPROC
```

Continúa en la página siguiente

1 Programación básica en RAPID

1.3.4 Utilización de llamadas a funciones en las expresiones

Continuación

Parámetros

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

- Los parámetros **IN** (los predeterminados) permiten usar cualquier expresión como argumento. La función a la que se llama considera el parámetro como una constante.
- Los parámetros **INOUT** requieren que el argumento correspondiente sea una variable (un elemento, un elemento de una matriz o un componente de un registro) o una variable persistente. La función a la que se llama obtiene un acceso completo (lectura y escritura) al argumento.
- Los parámetros **VAR** requieren que el argumento correspondiente sea una variable (un elemento, un elemento de una matriz o un componente de un registro) o una variable persistente. La función a la que se llama obtiene un acceso completo (lectura y escritura) al argumento.
- Los parámetros **PERS** requieren que el argumento correspondiente sea una variable persistente entera. La función a la que se llama obtiene un acceso completo (lectura y actualización) al argumento.

1.3.5 Prioridad entre operadores

Reglas de prioridad

La prioridad relativa de los operadores determina el orden en el que se evalúan. Los paréntesis proporcionan una forma de redefinir la prioridad de los operadores. Las reglas siguientes implican la siguiente prioridad de los operadores:

Prioridad	Operadores
Máximo	* / DIV MOD
	+ -
	< > <> <= >= =
	AND
Mínimo	XOR OR NOT

Los operadores con alta prioridad se evalúan antes que los operadores con prioridad baja. Los operadores con la misma prioridad se evalúan de izquierda a derecha.

Ejemplo de expresión	Orden de evaluación	Comentario
a + b + c	(a + b) + c	Regla de izquierda a derecha
a + b * c	a + (b * c)	* mayor que +
a OR b OR c	(a OR b) OR c	Regla de izquierda a derecha
a AND b OR c AND d	(a AND b) OR (c AND d)	AND mayor que OR
a < b AND c < d	(a < b) AND (c < d)	< mayor que AND

1.3.6 Sintaxis

Expresiones

```
<expression> ::= <expr> | <EXP>
<expr> ::= [ NOT ] <logical term> { ( OR | XOR ) <logical term> }
<logical term> ::= <relation> { AND <relation> }
<relation> ::= <simple expr> [ <relop> <simple expr> ]
<simple expr> ::= [ <addop> ] <term> { <addop> <term> }
<term> ::= <primary> { <mulop> <primary> }
<primary> ::=
    <literal>
    | <variable>
    | <persistent>
    | <constant>
    | <parameter>
    | <function call>
    | <aggregate>
    | '(' <expr> ')'
```

Operadores

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

Valores constantes

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

Datos

```
<variable> ::=
    <entire variable>
    | <variable element>
    | <variable component>
<entire variable> ::= <ident>
<variable element> ::= <entire variable> '{' <index list> '}'
<index list> ::= <expr> { ',' <expr> }
<variable component> ::= <variable> '.' <component name>
<component name> ::= <ident>
<persistent> ::=
    <entire persistent>
    | <persistent element>
    | <persistent component>
<constant> ::=
    <entire constant>
    | <constant element>
    | <constant component>
```

Agregados

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

Continúa en la página siguiente

Llamadas a funciones

```
<function call> ::= <function> '(' [ <function argument list> ]  
                    ')'  
<function> ::= <ident>  
<function argument list> ::= <first function argument> { <function  
                    argument> }  
<first function argument> ::=  
    <required function argument>  
    | <optional function argument>  
    | <conditional function argument>  
<function argument> ::=  
    ',' <required function argument>  
    | <optional function argument>  
    | ',' <optional function argument>  
    | <conditional function argument>  
    | ',' <conditional function argument>  
<required function argument> ::= [ <ident> ':' ] <expr>  
<optional function argument> ::= '\<ident> [ ':' ] <expr> ]  
<conditional function argument> ::= '\<ident> '?' <parameter>
```

Expresiones especiales

```
<constant expression> ::= <expression>  
<literal expression> ::= <expression>  
<conditional expression> ::= <expression>
```

Parámetros

```
<parameter> ::=  
    <entire parameter>  
    | <parameter element>  
    | <parameter component>
```

1 Programación básica en RAPID

1.4 Instrucciones

1.4 Instrucciones

Descripción

Las instrucciones se ejecutan una tras otra a no ser que una instrucción de flujo del programa, una interrupción o un error hagan que la ejecución continúe en algún otro lugar.

La mayoría de las instrucciones terminan en punto y coma “;”. Las etiquetas terminan en dos puntos “:”. Algunas instrucciones pueden contener otras instrucciones y terminan con palabras clave determinadas:

Instrucción	Palabra de terminación
IF	ENDIF
FOR	ENDFOR
WHILE	ENDWHILE
TEST	ENDTEST

Ejemplo:

```
WHILE index < 100 DO
    .
    index := index + 1;
ENDWHILE
```

Listas de selección

Todas las instrucciones están reunidas en grupos específicos que se describen en las secciones siguientes. Estas agrupaciones son las mismas que aparecen en las listas de selección que se utilizan al añadir nuevas instrucciones a un programa desde el editor de programas del FlexPendant.

Sintaxis

```
<instruction list> ::= { <instruction> }
<instruction> ::=
    [<instruction according to separate chapter in this manual>
    | <SMT>
```

1.5 Control del flujo del programa

Introducción

El programa se ejecuta secuencialmente como una regla, es decir, una instrucción tras otra. En ocasiones, se requieren instrucciones que interrumpen esta ejecución secuencial y que llaman a otra instrucción, para enfrentarse a las distintas situaciones que pueden darse durante la ejecución.

Principios de programación

El flujo del programa puede controlarse acorde con cinco principios diferentes:

- Llamar a otra rutina (procedimiento) y, una vez ejecutada dicha rutina, continuar la ejecución con la instrucción que sigue a la llamada a la rutina.
- Ejecutar instrucciones diferentes en función de si se cumple o no una condición determinada.
- Repetir una secuencia de instrucciones un número determinado de veces o hasta que se cumple una condición determinada.
- Ir a una etiqueta dentro de la misma rutina.
- Detener la ejecución del programa.

Llamada a otra rutina

Instrucción	Se usa para:
ProcCall	Llamar (saltar a) otra rutina
CallByVar	Llamar a procedimientos que tienen nombres concretos
RETURN	Volver a la rutina original

Control del programa dentro de la rutina

Instrucción	Se usa para:
Compact IF	Ejecutar una instrucción sólo si se cumple una condición
IF	Ejecutar una secuencia de instrucciones diferentes en función de si se cumple una condición
FOR	Repetir una sección del programa un número de veces
WHILE	Repetir una secuencia de instrucciones mientras siga cumpliéndose una condición
TEST	Ejecutar instrucciones diferentes en función del valor de una expresión
GOTO	Saltar a una etiqueta
label	Especificar una etiqueta (un nombre de línea)

Detención de la ejecución del programa

Instrucción	Se usa para:
Stop	Detiene la ejecución del programa
EXIT	Detener la ejecución del programa de forma que no se permita reiniciarlo

Continúa en la página siguiente

1 Programación básica en RAPID

1.5 Control del flujo del programa

Continuación

Instrucción	Se usa para:
Break	Detener temporalmente la ejecución del programa con fines de depuración
SystemStopAction	Detener la ejecución del programa y el movimiento del robot

Detención del ciclo actual

Instrucción	Se usa para:
ExitCycle	Detener el ciclo actual y trasladar el puntero del programa a la primera instrucción de la rutina principal. Cuando se selecciona el modo de ejecución <code>CONT</code> , la ejecución continúa en el siguiente ciclo de programa.

1.6 Otras instrucciones

Introducción

Existe un conjunto de instrucciones adicionales que se utilizan para:

- Asignar valores a los datos
- Esperar una determinada cantidad de tiempo o esperar hasta que se cumpla una condición
- Insertar un comentario en el programa
- Cargar módulos de programa

Asignación de un valor a un dato

Es posible asignar cualquier valor a un dato determinado. Por ejemplo, el valor puede ser inicializado con un valor constante, por ejemplo 5, o actualizado mediante una expresión aritmética, por ejemplo `reg1+5*reg3`.

Instrucción	Se usa para:
<code>:=</code>	Asignar un valor a un dato

Espera

Es posible programar el robot de forma que espere un tiempo determinado o que espere hasta que se cumpla la condición que desee, por ejemplo, esperar hasta que se active una entrada.

Instrucción	Se usa para:
<code>WaitTime</code>	Esperar una cantidad determinada de tiempo o esperar hasta que el robot deje de moverse
<code>WaitUntil</code>	Esperar hasta que se cumpla una condición
<code>WaitDI</code>	Esperar hasta que se active una entrada digital
<code>WaitDO</code>	Esperar hasta que se active una salida digital

Comentarios

La única finalidad de insertar comentarios en el programa es facilitar su lectura. Los comentarios no afectan a la ejecución del programa.

Instrucción	Se usa para:
<code>!</code>	Hacer comentarios sobre el programa. Las líneas que comienzan con <code>!</code> son comentarios y son descartados de la ejecución del programa.

Carga de módulos de programa

Los módulos de programa pueden cargarse desde la memoria de almacenamiento o eliminarse de la memoria de programas. Esto permite manejar programas grandes con sólo una memoria pequeña.

Instrucción	Se usa para:
<code>Load</code>	Cargar un módulo de programa en la memoria de programas
<code>UnLoad</code>	Descargar un módulo de programa de la memoria de programas

Continúa en la página siguiente

1 Programación básica en RAPID

1.6 Otras instrucciones

Continuación

Instrucción	Se usa para:
StartLoad	Cargar un módulo de programa en la memoria de programas durante la ejecución
WaitLoad	Conectar el módulo, si se ha cargado con StartLoad, a la tarea de programa
CancelLoad	Cancelar la carga de un módulo que se está cargando o que se ha cargado con la instrucción StartLoad
CheckProgRef	Comprobar referencias de programa
Save	Guardar un módulo de programa
EraseModule	Borrar un módulo de la memoria de programas

Tipo de dato	Se usa para:
loadsession	Programar una sesión de carga

Otras funciones

Instrucción	Se usa para:
TryInt	Comprobar si un objeto de dato es un entero válido

Función	Se usa para:
OpMode	Leer el modo actual de funcionamiento del robot
RunMode	Leer el modo actual de ejecución de programas del robot
NonMotionMode	Leer el modo sin ejecución de movimientos actual de la tarea del programa
Dim	Obtener las dimensiones de una matriz
Present	Determinar si está presente un parámetro opcional cuando se hace una llamada a una rutina
Type	Devuelve el nombre del tipo de dato de una variable especificada
IsPers	Comprobar si un parámetro es de tipo persistente
IsVar	Comprobar si un parámetro es una variable

Datos básicos

Tipo de dato	Para definir
bool	Datos lógicos (con los valores verdadero o falso)
num	Valores numéricos (decimales o enteros)
dnum	Valores numéricos (decimales o enteros). Un tipo de dato con un rango mayor que el de num.
string	Cadenas de caracteres
switch	Parámetros de rutina sin valor

Continúa en la página siguiente

Funciones de conversión

Función	Se usa para:
StrToByte	Convertir un byte en un dato de cadena de caracteres con un formato definido
ByteToStr	Convertir una cadena de caracteres con un formato definido de datos de byte a un dato de byte

1 Programación básica en RAPID

1.7 Parámetros de movimiento

1.7 Parámetros de movimiento

Introducción

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

- Velocidad máxima de TCP
- Velocidad máxima y ajuste de velocidad
- Aceleración
- Gestión de distintas configuraciones de robot
- Carga útil
- Comportamiento cerca de puntos singulares
- Desplazamiento del programa
- Servo suave
- Valores de ajuste
- Activación y desactivación de búfer de eventos

Principios de programación

Las características básicas del movimiento del robot dependen de los datos especificados con cada instrucción de posicionamiento. Sin embargo, algunos datos se especifican en instrucciones separadas que se aplican a todos los movimientos hasta que cambia dicha información.

Los parámetros generales de movimiento se especifican utilizando varias instrucciones, pero también pueden leerse a través de la variable de sistema `C_MOTSET` o `C_PROGDISP`.

Los valores predeterminados se establecen automáticamente (mediante la ejecución de la rutina `SYS_RESET` del módulo `BASE_SHARED` del sistema) en los casos siguientes:

- cuando se utiliza el modo de reinicio **Restablecer sistema**,
- Cuando se carga un nuevo programa
- Cuando se inicia el programa desde el principio

Función de velocidad máxima de TCP

Función	Se usa para:
<code>MaxRobSpeed</code>	Devolver la velocidad de TCP máxima del tipo de robot utilizado

Definición de la velocidad

La velocidad absoluta se programa como un argumento de la instrucción de posicionamiento. Además, es posible definir la velocidad máxima y el ajuste de velocidad (un porcentaje de la velocidad programada).

También es posible establecer una limitación de la velocidad y activarla al activar una señal de entrada de sistema.

Instrucción	Para definir
<code>VelSet</code>	La velocidad máxima y el ajuste de velocidad

Continúa en la página siguiente

Instrucción	Para definir
SpeedRefresh	El ajuste de velocidad para el movimiento en curso
SpeedLimAxis	Establecer la limitación de velocidad de un eje. Se aplicará posteriormente a través de una señal de entrada de sistema.
SpeedLimCheckPoint	Establecer la limitación de velocidad de los puntos de control. Se aplicará posteriormente a través de una señal de entrada de sistema.

Definición de la aceleración

En algunos casos, por ejemplo cuando se manejan piezas frágiles, es posible reducir la aceleración en una parte del programa.

Instrucción	Se usa para:
AccSet	Definir la aceleración máxima.
WorldAccLim	Limitar la aceleración/deceleración de la herramienta (y la carga de la pinza) dentro del sistema de coordenadas mundo.
PathAccLim	Establecer o restablecer limitaciones de aceleración y/o deceleración de TCP a lo largo de la trayectoria de movimiento.

Definición del control de configuración

Normalmente, la configuración del robot se comprueba durante los movimientos. Si se utiliza el movimiento de ejes (eje por eje), se conseguirá la configuración correcta. Si se utiliza un movimiento lineal o circular, el robot siempre se desplaza hacia la configuración más cercana, pero se realiza una comprobación si es la misma que la programada. Sin embargo, es posible cambiar este comportamiento.

Instrucción	Se usa para:
ConfJ	Control de configuración activado/desactivado durante el movimiento de ejes
ConfL	Control de configuración activado/desactivado durante el movimiento lineal

Definición de la carga útil

Para alcanzar el rendimiento óptimo del robot, es necesario definir la carga útil correcta.

Instrucción	Para definir
GripLoad	La carga útil de la pinza

Definición del comportamiento cerca de puntos singulares

Es posible programar el robot para evitar puntos singulares, mediante la modificación automática de la orientación de la herramienta.

Instrucción	Para definir
SingArea	El método de interpolación utilizado al pasar por puntos singulares

Continúa en la página siguiente

1 Programación básica en RAPID

1.7 Parámetros de movimiento

Continuación

Activación y desactivación de búfer de eventos

Para conseguir un rendimiento óptimo en el robot y un buen comportamiento de las aplicaciones al combinar una aplicación con puntos finos y una aplicación continua en la cual es necesario activar señales de antemano debido al uso de equipos de proceso lentos, es posible activar y desactivar el búfer de eventos.

Instrucción	Para definir
ActEventBuffer	Activar el búfer de eventos configurado.
DeactEventBuffer	Desactivar el uso del búfer de eventos.

Desplazamiento de programa

Cuando es necesario desplazar una parte del programa, por ejemplo a continuación de una búsqueda, es posible añadir un desplazamiento de programa.

Instrucción	Se usa para:
PDispOn	Activar el desplazamiento de programa
PDispSet	Activar el desplazamiento de programa mediante la especificación de un valor
PDispOff	Desactivar el desplazamiento de programa
EOffsOn	Activar un offset de eje adicional
EOffsSet	Activar un offset de eje adicional mediante la especificación de un valor
EOffsOff	Desactivar un offset de eje adicional

Función	Se usa para:
DefDFrame	Calcular un desplazamiento de programa desde tres posiciones
DefFrame	Calcular un desplazamiento de programa desde seis posiciones
ORobT	Eliminar el desplazamiento de programa desde una posición
DefAccFrame	Definir una base de coordenadas desde las posiciones originales y las desplazadas

Servo suave

Uno o varios de los ejes del robot pueden funcionar en el modo “suave”. Con esta función, el robot puede adaptarse a nuevas situaciones y puede sustituir, por ejemplo, a una herramienta con resorte.

Instrucción	Se usa para:
SoftAct	Activar el servo suave para uno o varios ejes
SoftDeact	Desactivar el servo suave
DitherAct ⁱ	Permite aplicar una función de oscilación al servo suave
DitherDeact	Desactivar una función de oscilación al servo suave

ⁱ Sólo en el sistema IRB 7600

Continúa en la página siguiente

Ajustar los valores de ajuste del robot

En general, el funcionamiento del robot incorpora una optimización automática. Sin embargo, en casos extremos pueden producirse situaciones como rebasamientos. Puede modificar los valores de ajuste del robot para conseguir el funcionamiento requerido.

Instrucción	Se usa para:
TuneServo	Ajustar los valores de ajuste del robot
TuneReset	Restablecer el ajuste al modo normal
PathResol	Ajustar la resolución de trayectorias geométricas
CirPathMode	Seleccionar la forma en que la herramienta cambia de orientación durante una interpolación circular
Tipo de dato	Se usa para:
tunetype	Representar el tipo de ajuste como una constante simbólica

Zonas mundo

Es posible definir hasta 10 volúmenes diferentes dentro del área de trabajo del robot. Puede usarlos para:

- Indicar que el TCP del robot es una parte definida del área de trabajo.
- Delimitar el área de trabajo del robot e impedir colisiones con la herramienta.
- Crear un área de trabajo común para dos robots. En este caso, el área de trabajo está disponible sólo para un robot cada vez.

Las instrucciones de la siguiente tabla sólo están disponibles si el robot está equipado con la opción *World Zones*.

Instrucción	Se usa para:
WZBoxDef	Definir una zona mundo cuadrada
WZCylDef	Definir una zona mundo cilíndrica
WZSphDef	Definir una zona mundo esférica
WZHomeJointDef	Definir una zona mundo en coordenadas de ejes
WZLimJointDef	Definir una zona mundo en coordenadas de ejes para la delimitación del área de trabajo
WZLimSup	Activar la supervisión de límites de una zona mundo
WZDOSet	Activar la zona mundo para el establecimiento de salidas digitales
WZDisable	Desactivar la supervisión de una zona mundo temporal
WZEnable	Activar la supervisión de una zona mundo temporal
WZFree	Eliminar la supervisión de una zona mundo temporal
wztemporary	Identificar una zona mundo temporal
wzstationary	Identificar una zona mundo estacionaria
shapedata	Describir la geometría de una zona mundo

Continúa en la página siguiente

1 Programación básica en RAPID

1.7 Parámetros de movimiento

Continuación

Varios para parámetros de movimiento

Instrucción	Se usa para:
WaitRob	Esperar hasta que el robot y el eje adicional han llegado al punto de paro o tienen una velocidad cero.

Tipo de dato	Se usa para:
motsetdata	Parámetros de movimiento, excepto desplazamiento de programa
progdisp	Desplazamiento del programa

1.8 Movimiento

Principio de movimiento del robot

Los movimientos del robot se programan como movimientos de posición a posición, es decir, “moverse de la posición actual a una nueva posición”. La trayectoria entre estas dos posiciones es calculada automáticamente por el robot.

Principios de programación

Las características básicas del movimiento, como el tipo de trayectoria, se especifican mediante la selección de la instrucción de posicionamiento adecuada.

Las demás características de movimiento se especifican mediante la definición de datos que se usan como argumentos de la instrucción:

- Datos de posición (posición final del robot y los ejes adicionales)
- Datos de velocidad (velocidad deseada)
- Datos de zona (exactitud de la posición)
- Datos de la herramienta (por ejemplo, la posición del TCP)
- Datos de objeto tratado (por ejemplo, el sistema de coordenadas actual)

Algunas de las características de movimiento del robot se determinan mediante instrucciones lógicas que se aplican a todos los movimientos (consulte [Parámetros de movimiento en la página 52](#)):

- Velocidad máxima y ajuste de velocidad
- Aceleración
- Gestión de distintas configuraciones de robot
- Carga útil
- Comportamiento cerca de puntos singulares
- Desplazamiento del programa
- Servo suave
- Valores de ajuste
- Activación y desactivación de búfer de eventos

Para el posicionamiento tanto del robot como de los ejes adicionales se usan las mismas instrucciones. Los ejes adicionales se mueven a una velocidad constante y alcanzan la posición final al mismo tiempo que el robot.

Instrucciones de posicionamiento

Instrucción	Tipo de movimiento
MoveC	Mover el TCP a lo largo de una trayectoria circular.
MoveJ	Movimiento de ejes.
MoveL	Mover el TCP a lo largo de una trayectoria lineal.
MoveAbsJ	Movimiento absoluto de ejes.
MoveExtJ	Mover un eje adicional lineal o giratorio sin TCP.
MoveCAO	Mueve el robot en una trayectoria circular y establece una salida analógica en la esquina

Continúa en la página siguiente

1 Programación básica en RAPID

1.8 Movimiento

Continuación

Instrucción	Tipo de movimiento
MoveCDO	Mover el robot en una trayectoria circular y establecer una salida digital a medio camino de la trayectoria de esquina
MoveCGO	Mueve el robot en una trayectoria circular y establece una señal de salida de grupo en la esquina
MoveJAO	Mueve el robot mediante el movimiento de los ejes y activa una salida analógica en la esquina
MoveJDO	Mover el robot con un movimiento de ejes y establecer una salida digital a medio camino de la trayectoria de esquina
MoveJGO	Mueve el robot mediante un movimiento de ejes y establece una señal de salida de grupo en la esquina
MoveLAO	Mueve el robot siguiendo una trayectoria lineal y establece una salida analógica en la esquina
MoveLDO	Mover el robot en una trayectoria lineal y establecer una salida digital a medio camino de la trayectoria de esquina
MoveLGO	Mueve el robot linealmente y establece una señal de salida de grupo en la esquina
MoveCSync	Mover el robot en una trayectoria circular y ejecutar un procedimiento de RAPID.
MoveJSync	Mover el robot con un movimiento de ejes y ejecutar un procedimiento de RAPID.
MoveLSync	Mover el robot de forma lineal y ejecutar un procedimiento de RAPID.

Búsqueda

Durante el movimiento, el robot puede buscar la posición del objeto de trabajo, por ejemplo. La posición buscada (indicada por una señal de sensor) se almacena y puede usarse más tarde para posicionar el robot o para calcular un desplazamiento de programa.

Instrucción	Tipo de movimiento
SearchC	TCP a lo largo de una trayectoria circular.
SearchL	TCP a lo largo de una trayectoria lineal.
SearchExtJ	Movimiento de ejes de la unidad mecánica sin TCP.

Activación de salidas o interrupciones en posiciones concretas

Normalmente, las instrucciones lógicas se ejecutan en la transición de una instrucción de posicionamiento a otra. Sin embargo, si se utilizan instrucciones especiales de movimiento, podrían ejecutarse éstas en su lugar cuando el robot se encuentra en una posición determinada.

Instrucción	Se usa para:
TriggC	Mover el robot (el TCP) en círculo con una condición de disparo activada.
TriggCheckIO	Definir una comprobación de E/S en una posición determinada
TriggEquip	Definir una condición de disparo para establecer una salida en una posición determinada, con la posibilidad de incluir una compensación de tiempo por el retardo de los equipos externos.

Continúa en la página siguiente

Instrucción	Se usa para:
TriggDataCopy	Copiar el contenido de una variable de tipo trigndata
TriggDataReset	Restablecer el contenido en una variable de tipo trigndata
TriggRampAO	Definir una condición de disparo para aumentar o disminuir en rampa una señal de salida analógica en una posición determinada, con la posibilidad de incluir una compensación de tiempo por el retardo de los equipos externos.
TriggJ	Mover el robot eje por eje con una condición de disparo activada.
TriggJIos	Mover el robot (el TCP) eje por eje con una condición de disparo de E/S activada.
TriggInt	Definir una condición de disparo para ejecutar una rutina TRAP en una posición determinada
TriggIO	Definir una condición de disparo para establecer una salida en una posición determinada
TriggL	Mover el robot (el TCP) linealmente con una condición de disparo activada.
TriggLIos	Mover el robot (el TCP) linealmente con una condición de disparo de E/S activada.
StepBwdPath	Retroceder en su trayectoria en una rutina de evento RESTART.
TriggStopProc	Crear un proceso interno de supervisión en el sistema para la puesta a cero de las señales de proceso especificadas y la generación de datos de reinicio en una variable persistente especificada, cada vez que se detiene el programa (STOP) o se produce un paro de emergencia (QSTOP) en el sistema.
Funciones	Se usa para:
TriggDataValid	Comprobar si el contenido de una variable de tipo trigndata es válido
Tipos de datos	Se usa para:
trigndata	Condiciones de disparo
aiotrigg	Condiciones de disparo con E/S analógica
restartdata	Datos para TriggStopProc
triggios	Condiciones de disparo para TriggJIos y TriggLIos
triggstrgo	Condiciones de disparo para TriggJIos y TriggLIos
triggiosdnum	Condiciones de disparo para TriggJIos y TriggLIos

Control de una señal analógica de salida de forma proporcional al TCP real

Instrucción	Se usa para:
TriggSpeed	Definir condiciones y acciones para el control de una señal analógica de salida cuyo valor de salida es proporcional a la velocidad real del TCP.

Continúa en la página siguiente

1 Programación básica en RAPID

1.8 Movimiento

Continuación

Control del movimiento si se produce un error o una interrupción

Para poder corregir un error o una interrupción, es posible detener temporalmente el movimiento y reiniciarlo posteriormente.

Instrucción	Se usa para:
StopMove	Definir condiciones y acciones para el control de una señal analógica de salida cuyo valor de salida es proporcional a la velocidad real del TCP.
StartMove	Reiniciar los movimientos del robot
StartMoveRetry	Reiniciar los movimientos del robot y hacer un reintento en una secuencia indivisible
StopMoveReset	Restablecer el estado de detención de movimiento, pero sin iniciar los movimientos del robot
StorePath	Almacenar la última trayectoria generada
RestoPath	Restaurar una trayectoria almacenada anteriormente
ClearPath	Eliminar toda la trayectoria de movimiento del nivel actual de trayectorias de movimiento.
PathLevel	Obtener el nivel de trayectoria actual.
SyncMoveSuspend ⁱ	Suspender el movimiento coordinado sincronizado en el nivel StorePath.
SyncMoveResume ⁱ	Reanudar el movimiento coordinado sincronizado en el nivel StorePath.

ⁱ Sólo si el robot está equipado con la opción *MultiMove Coordinated*.

Función	Se usa para:
IsStopMoveAct	Obtener el estado de los indicadores de movimiento de paro.

Obtención de la información de robot en un sistema MultiMove

Se utiliza para obtener un nombre o una referencia al robot en la tarea de programa actual.

Función	Se usa para:
RobName	Obtener el nombre del robot controlado en la tarea de programa actual, si lo hay.

Datos	Se usa para:
ROB_ID	Obtener datos que contienen una referencia al robot controlado en la tarea de programa actual, si lo hay.

Control de ejes adicionales

Para el posicionamiento del robot y de los ejes adicionales se usan las mismas instrucciones. Sin embargo, algunas instrucciones sólo afectan a los movimientos de los ejes adicionales.

Instrucción	Se usa para:
DeactUnit	Desactivar una unidad mecánica externa
ActUnit	Activar una unidad mecánica externa
MechUnitLoad	Definir una carga útil para una unidad mecánica

Continúa en la página siguiente

Función	Se usa para:
GetMotorTorque	Lee el par actual de los motores del robot y de los ejes externos; puede utilizarse para detectar si una pinza servo sostiene o no una carga.
GetNextMechUnit	Obtener el nombre de las unidades mecánicas del sistema del robot
IsMechUnitActive	Comprobar si una unidad mecánica está activada o no

Ejes independientes

El eje 6 del robot (o el 4 en el caso del sistema IRB 1600, 2600 y 4600, excepto las versiones de ID) o un eje adicional pueden moverse independientemente del resto de movimientos. También es posible restablecer el área de trabajo de un eje, con lo que se reducen los tiempos de ciclo.

Las instrucciones de la siguiente tabla sólo están disponibles si el robot está equipado con la opción *Independent Axis*.

Instrucción	Se usa para:
IndAMove	Poner un eje en el modo independiente y mover el eje hasta una posición absoluta.
IndCMove	Poner un eje en el modo independiente e iniciar un movimiento continuo del eje.
IndDMove	Poner un eje en el modo independiente y mover el eje una distancia delta.
IndRMove	Poner un eje en el modo independiente y mover el eje hasta una posición relativa (dentro de la revolución del eje).
IndReset	Poner un eje en el modo dependiente o/y restablecer el área de trabajo.
HollowWristReset ⁱ	Restablecer la posición de los ejes de muñeca en los manipuladores de muñeca huecos, por ejemplo, en los sistemas IRB 5402 e IRB 5403.

ⁱ Sólo puede usarse en los robots IRB 5402 e IRB 5403.

Las funciones de la siguiente tabla sólo están disponibles si el robot está equipado con la opción *Independent Axis*.

Función	Se usa para:
IndInpos	Comprobar si un eje independiente está en posición.
IndSpeed	Comprobar si un eje independiente ha alcanzado la velocidad programada.

Corrección de trayectoria

Las instrucciones, funciones y tipos de datos de las siguientes tablas sólo están disponibles si el robot está equipado con las opciones *Path offset* o *RobotWare-Arc sensor*.

Instrucción	Se usa para:
CorrCon	Comprobar si un eje independiente está en posición
CorrWrite	Comprobar si un eje independiente ha alcanzado la velocidad programada

Continúa en la página siguiente

1 Programación básica en RAPID

1.8 Movimiento

Continuación

Instrucción	Se usa para:
CorrDiscon	Desconectarse de un generador de conexión con el que se ha conectado anteriormente
CorrClear	Eliminar todos los generadores de corrección conectados

Función	Se usa para:
CorrRead	Leer las correcciones totales suministradas por todos los generadores de correcciones conectados.

Tipo de dato	Se usa para:
corrdescr	Añadir offsets geométricos al sistema de coordenadas de trayectoria

Grabadora de trayectorias

Las instrucciones, funciones y tipos de datos de las siguientes tablas sólo están disponibles si el robot está equipado con la opción *Path Recovery*.

Instrucción	Se usa para:
PathRecStart	Iniciar la grabación de la trayectoria del robot
PathRecStop	Detener la grabación de la trayectoria del robot
PathRecMoveBwd	Mover el robot hacia atrás por una trayectoria grabada
PathRecMoveFwd	Mover el robot de nuevo a la posición en la que se ejecutó PathRecMoveBwd

Función	Se usa para:
PathRecValidBwd	Comprobar si la grabadora de trayectorias está activa y si está disponible una trayectoria hacia atrás grabada
PathRecValidFwd	Comprobar si la grabadora de trayectorias puede utilizarse para moverse hacia adelante

Tipo de dato	Se usa para:
pathrecid	Identificar un punto de ruptura para la grabadora de trayectorias

Seguimiento de transportadores

Las instrucciones de la siguiente tabla sólo están disponibles si el robot está equipado con la opción *Conveyor tracking*.

Instrucción	Se usa para:
WaitWObj	Esperar a un objeto de trabajo en un transportador
DropWObj	Soltar el objeto de trabajo sobre el transportador

Seguimiento servo para transportador de indexación

Las instrucciones de la siguiente tabla sólo están disponibles si el robot está equipado con la opción *Conveyor tracking*.

Instrucción	Se usa para:
IndCnvAddObject	Se utiliza para añadir manualmente un objeto a la cola de objetos.

Continúa en la página siguiente

Instrucción	Se usa para:
IndCnvEnable	Se inicia la escucha de la entrada digital y se ejecuta un movimiento de indexación al dispararse.
IndCnvDisable	El sistema detiene la escucha de la entrada digital.
IndCnvInit	Configura la funcionalidad del transportador de indexación.
IndCnvReset	Para poder realizar un movimiento manual o ejecutar una instrucción de movimiento para el transportador de indexación, es necesario cambiar el sistema al modo Normal, lo que ocurre con esta instrucción o al mover el PP a Main.
indcnvdata	Se utiliza para configurar el comportamiento de la funcionalidad del transportador de indexación.

Sensor Synchronization

Sensor Synchronization es la función por la cual la velocidad del robot depende de un sensor que puede montarse sobre un transportador móvil o el eje de un motor de empuje.

Las instrucciones de la siguiente tabla sólo están disponibles si el robot está equipado con la opción *Sensor Synchronization*.

Instrucción	Se usa para:
WaitSensor	Conectarse con un objeto de la ventana inicial de una unidad mecánica con sensor.
SyncToSensor	Iniciar o detener la sincronización de los movimientos del robot con el movimiento de los sensores.
DropSensor	Desconectarse del objeto actual.

Identificación de carga y detección de colisiones

Instrucción	Se usa para:
MotionSup ⁱ	Desactiva/activa la supervisión del movimiento
ParIdPosValid	Posición de robot válida para la identificación de parámetros
ParIdRobValid	Tipo de robot válido para la identificación de parámetros
LoadId	Identificación de carga de la herramienta o la carga útil
ManLoadId	Identificación de carga de manipuladores externos

ⁱ Sólo si el robot está equipado con la opción *Collision Detection*.

Tipo de dato	Se usa para:
loadidnum	Representar un entero con una constante simbólica
paridnum	Representar un entero con una constante simbólica
paridvalidnum	Representar un entero con una constante simbólica

Funciones de posición

Función	Se usa para:
Offs	Añadir un offset a una posición del robot, expresada en relación con el objeto de trabajo

Continúa en la página siguiente

1 Programación básica en RAPID

1.8 Movimiento

Continuación

Función	Se usa para:
RelTool	Añadir un offset, expresado en el sistema de coordenadas de la herramienta
CalcRobT	Calcular el valor de <code>robtarget</code> a partir del valor de <code>jointtarget</code>
CPos	Leer la posición actual (sólo los valores x, y, z del robot)
CRobT	Leer la posición actual (<code>robtarget</code> completo)
CJointT	Leer los ángulos actuales de los ejes
ReadMotor	Leer los ángulos actuales de los motores
CTool	Leer el valor actual de <code>tooldata</code>
CWobj	Leer el valor actual de <code>wobjdata</code>
ORobT	Eliminar el desplazamiento de programa desde una posición
MirPos	Calcular la posición especular de una posición
CalcJointT	Calcular los ángulos de las articulaciones a partir del valor de <code>robtarget</code>
Distance	La distancia entre dos posiciones

Comprobación de una trayectoria interrumpida después de una caída de alimentación

Función	Se usa para:
PFRestart	Comprobar si la trayectoria fue interrumpida como consecuencia de la caída de alimentación.

Funciones de estado

Función	Se usa para:
CSpeedOverride	Leer el ajuste de velocidad establecido por el usuario en el Editor de programas o la ventana de producción.

Datos de movimiento

Los datos de movimiento se utilizan como argumentos de las instrucciones de posicionamiento.

Tipo de dato	Para definir
<code>robtarget</code>	La posición final
<code>jointtarget</code>	La posición final de una instrucción <code>MoveAbsJ</code> o <code>MoveExtJ</code>
<code>speeddata</code>	La velocidad
<code>zonedata</code>	La exactitud de la posición (punto de paro o punto de paso)
<code>tooldata</code>	El sistema de coordenadas de la herramienta y la carga de la herramienta
<code>wobjdata</code>	El sistema de coordenadas del objeto de trabajo
<code>stoppointdata</code>	La finalización de la posición
<code>identno</code>	Un número utilizado para controlar la sincronización de dos o más movimientos sincronizados y coordinados entre sí

Continúa en la página siguiente

Datos básicos de los movimientos

Tipo de dato	Para definir
pos	Una posición (x, y, z)
orient	Una orientación
pose	Un sistema de coordenadas (posición y orientación)
confdata	La configuración de los ejes del robot
extjoint	La posición del eje adicional
robjoint	La posición de los ejes del robot
loaddata	Una carga
mecunit	Una unidad mecánica externa

Información relacionada

Opciones	Se describe en
<i>Collision Detection</i> <i>Sensor Synchronization</i> <i>Independent Axis</i> <i>Path Offset</i> <i>Path Recovery</i>	<i>Application manual - Controller software IRC5</i>
<i>Conveyor tracking</i>	<i>Application manual - Conveyor tracking</i>
<i>MultiMove</i>	<i>Manual de aplicaciones - MultiMove</i>
<i>RobotWare-Arc</i>	<i>Application manual - Arc and Arc Sensor</i>

1 Programación básica en RAPID

1.9 Señales de entrada y salida

1.9 Señales de entrada y salida

Señales

El robot puede contar con varias señales de usuario digitales y analógicas que pueden leerse y modificarse desde el propio programa.

Principios de programación

Los nombres de las señales se definen en los parámetros del sistema. Estos nombres están siempre disponibles en el programa para la lectura o el establecimiento de operaciones de E/S.

El valor de una señal analógica o de un grupo de señales digitales se especifica como un valor numérico.

Modificación del valor de una señal

Instrucción	Para definir
InvertDO	Invertir el valor de una señal digital de salida
PulseDO	Generar un pulso en una señal digital de salida
Reset	Restablecer una señal digital de salida (ponerla a 0)
Set	Activar una señal digital de salida (cambiarla a 1)
SetAO	Cambiar el valor de una señal analógica de salida
SetDO	Cambiar el valor de una señal digital de salida (su valor simbólico, por ejemplo <i>high/low</i>)
SetGO	Cambiar el valor de un grupo de señales digitales de salida

Lectura del valor de una señal de entrada

El valor de una señal de entrada puede leerse directamente desde el programa, como en los ejemplos siguientes:

```
! Digital input
IF dil = 1 THEN ...
! Digital group input
IF gil = 5 THEN ...
! Analog input
IF ail > 5.2 THEN ...
```

Pueden generarse los errores recuperables siguientes. El error puede ser gestionado en un gestor de errores. La variable de sistema ERRNO cambia a:

ERR_NORUNUNIT Sin contacto con la unidad de E/S.

Lectura del valor de una señal de salida

Función	Para definir
AOutput	Leer el valor actual de una señal analógica de salida
DOutput	Leer el valor actual de una señal digital de salida
GOutput	Leer el valor actual de un grupo de señales digitales de salida

Continúa en la página siguiente

Función	Para definir
GOutputDnum	Leer el valor actual de un grupo de señales digitales de salida. Puede gestionar señales digitales de grupo de hasta 32 bits. El valor leído se devuelve con el tipo de dato <code>dnum</code> .
GInputDnum	Leer el valor actual de un grupo de señales digitales de entrada. Puede gestionar señales digitales de grupo de hasta 32 bits. El valor leído se devuelve con el tipo de dato <code>dnum</code> .

Comprobación de señales de entrada o salida

Instrucción	Para definir
WaitDI	Esperar hasta que se activa o se pone a cero una entrada digital
WaitDO	Esperar hasta que se activa o se pone a cero una salida digital
WaitGI	Esperar hasta que un grupo de entradas digitales cambie a un valor
WaitGO	Esperar hasta que un grupo de salidas digitales cambie a un valor
WaitAI	Esperar hasta que una entrada digital sea inferior o superior a un valor
WaitAO	Esperar hasta que una salida digital sea inferior o superior a un valor

Función	Para definir:
TestDI	Comprobar si una entrada digital está activada
ValidIO	Señal de E/S válida para su uso
GetSignalOrigin	Obtención de información acerca del origen de una señal de E/S

Tipo de dato	Para definir
signalorigin	Describe el origen de la señal de E/S

Desactivación y activación de módulos de E/S

Los módulos de E/S están activados automáticamente en el momento de la puesta en marcha del sistema, pero pueden desactivarse durante la ejecución del programa y reactivarse más tarde.

Instrucción	Para definir
IODisable	Desactivar un módulo de E/S
IOEnable	Activar un módulo de E/S

Definición de señales de entrada y salida

Instrucción	Para definir
AliasIO	Definir una señal con un nombre de alias

Tipo de dato	Para definir
dionum	El valor simbólico de una señal digital
signalai	El nombre de una señal analógica de entrada

Continúa en la página siguiente

1 Programación básica en RAPID

1.9 Señales de entrada y salida

Continuación

Tipo de dato	Para definir
signalao	El nombre de una señal analógica de salida
signaldi	El nombre de una señal digital de entrada
signaldo	El nombre de una señal digital de salida
signalgi	El nombre de un grupo de señales digitales de entrada
signalgo	El nombre de un grupo de señales digitales de salida
signalorigin	Describe el origen de la señal de E/S

Obtención del estado del bus y la unidad de E/S

Instrucción	Para definir
IOBusState	Obtener el estado actual del bus de E/S.

Función	Para definir
IOUnitState	Devuelve el estado actual de la unidad de E/S.

Tipo de dato	Para definir
iounit_state	El estado de la unidad de E/S
bustate	El estado del bus de E/S

Inicio de un bus de E/S

Instrucción	Para definir
IOBusStart	Iniciar un bus de E/S.

1.10 Comunicaciones

Comunicación a través de canales serie

Existen cuatro formas posibles de comunicarse a través de canales serie:

- Es posible enviar mensajes a la pantalla del FlexPendant y el usuario puede responder a las preguntas, por ejemplo, sobre el número de piezas a procesar.
- Es posible almacenar información alfanumérica en archivos de texto de la memoria de almacenamiento o leerse de éstos. Por ejemplo, es posible almacenar de esta forma las estadísticas de producción y procesarlas más adelante en un PC. También es posible enviar directamente la información a una impresora conectada al robot.
- Es posible transferir información binaria entre el robot y un sensor, por ejemplo.
- También es posible transferir información binaria entre el robot y otro ordenador, por ejemplo, a través de un protocolo de comunicaciones.

Principios de programación

La decisión de utilizar información alfanumérica o binaria depende de cómo manejan esta información los equipos con los que se comunica el robot. Por ejemplo, un archivo puede contener datos almacenados en formato alfanumérico o binario.

Si se requiere una comunicación en ambos sentidos de forma simultánea, debe utilizarse la transmisión binaria.

En primer lugar es necesario abrir el canal serie o el archivo. Al hacerlo, se asigna al canal o archivo un descriptor que se utiliza posteriormente como referencia al leer o escribir. El FlexPendant está disponible en todo momento y no es necesario abrirlo.

Es posible imprimir tanto textos como los valores de determinados tipos de datos.

Comunicación a través del FlexPendant, grupo de funciones TP

Instrucción	Se usa para:
TPERase	Vaciar la pantalla de usuario del FlexPendant
TPWrite	Escribir un texto en la pantalla de usuario del FlexPendant
ErrWrite	Escribir un texto en la pantalla del FlexPendant y almacenarlo a la vez en el registro de errores del programa.
TPReadFK	Asignar etiquetas a las teclas de función y leer qué tecla se ha pulsado
TPReadDnum	Leer un valor numérico del FlexPendant
TPReadNum	Leer un valor numérico del FlexPendant
TPShow	Seleccionar una ventana del FlexPendant desde RAPID
tpnum	Representar la ventana del FlexPendant con una constante simbólica

Continúa en la página siguiente

1 Programación básica en RAPID

1.10 Comunicaciones

Continuación

Comunicación a través del FlexPendant, grupo de funciones UI

Instrucción	Se usa para:
UIMsgBox	Escribir un mensaje en el FlexPendant Leer el botón presionado en el FlexPendant Tipo básico
UIShow	Abrir una aplicación en el FlexPendant desde RAPID

Función	Se usa para:
UIMessageBox	Escribir un mensaje en el FlexPendant Leer el botón presionado en el FlexPendant Tipo avanzado
UIDNumEntry	Leer un valor numérico del FlexPendant
UIDNumTune	Ajustar un valor numérico del FlexPendant
UINumEntry	Leer un valor numérico del FlexPendant
UINumTune	Ajustar un valor numérico del FlexPendant
UIAlphaEntry	Leer un texto del FlexPendant
UIListView	Seleccionar un elemento de una lista del FlexPendant
UIClientExist	Obtener una indicación de si el FlexPendant está conectado al sistema

Tipo de dato	Se usa para:
icondata	Representar un icono con una constante simbólica
buttondata	Representar un botón con una constante simbólica
listitem	Definir elementos de listas de menú
btnres	Representar un botón seleccionado con una constante simbólica
uishownum	ID de instancia para UIShow

Lectura o escritura a través de un canal serie o un archivo alfanumérico

Instrucción	Se usa para:
Open	Abrir un canal o un archivo para lectura o escritura
Write	Escribir un texto en el canal o el archivo
Close	Cerrar el canal o el archivo

Función	Se usa para:
ReadNum	Leer un valor numérico
ReadStr	Leer una cadena de caracteres

Comunicación a través de canales serie o archivos o buses de campo binarios

Instrucción	Se usa para:
Open	Abrir un canal serie o un archivo para una transferencia binaria de datos
WriteBin	Escribir en un canal serie o un archivo binario

Continúa en la página siguiente

Instrucción	Se usa para:
WriteAnyBin	Escribir en cualquier canal serie o archivo binario
WriteStrBin	Escribir una cadena de caracteres en cualquier canal serie o archivo binario
Rewind	Seleccionar el principio del archivo como posición actual
Close	Cerrar el canal o el archivo
ClearIOBuff	Vaciar el búfer de entrada de un canal serie
ReadAnyBin	Leer de cualquier canal serie binario
WriteRawBytes	Escribir los datos de tipo rawbytes en un canal serie o un archivo o un bus de campo binario
ReadRawBytes	Leer los datos de tipo rawbytes de un canal serie o un archivo o un bus de campo binario
Función	Se usa para:
ReadBin	Leer de un canal serie binario
ReadStrBin	Leer una cadena de caracteres de un canal serie o un archivo binario

Comunicación a través de datos rawbytes

Las instrucciones y funciones siguientes se usan en apoyo de las instrucciones de comunicación `WriteRawBytes` y `ReadRawBytes`.

Instrucción	Se usa para:
ClearRawBytes	Cambiar la variable rawbytes a cero
CopyRawBytes	Copiar de una variable rawbytes a otra
PackRawBytes	Empaquetar el contenido de una variable en un “contenedor” de tipo rawbytes
UnPackRawBytes	Desempaquetar el contenido de un “contenedor” de tipo rawbytes en una variable
PackDNHeader	Empaquetar el título del mensaje DeviceNet en un “contenedor” de rawbytes
Función	Se usa para:
RawBytesLen	Obtener la longitud actual de los bytes válidos en una variable rawbyte

Datos para canales, archivos y buses de campo en serie

Tipo de dato	Para definir
iodev	Una referencia a un canal serie o un archivo, para usarla posteriormente en operaciones de lectura y escritura
rawbytes	Un “contenedor” general de datos, que se utiliza para la comunicación con los dispositivos de E/S

Comunicación mediante zócalos

Instrucción	Se usa para:
SocketCreate	Creación de un nuevo zócalo

Continúa en la página siguiente

1 Programación básica en RAPID

1.10 Comunicaciones

Continuación

Instrucción	Se usa para:
SocketConnect	Conexión a un ordenador remoto (sólo aplicaciones cliente)
SocketSend	Envío de datos a un ordenador remoto
SocketSendTo	Envío de datos a un ordenador remoto
SocketReceive	Recepción de datos desde un ordenador remoto
SocketReceiveFrom	Recepción de datos desde un ordenador remoto
SocketClose	Cierre del zócalo
SocketBind	Enlazar un zócalo a un puerto (sólo aplicaciones de servidor)
SocketListen	Permanecer a la escucha de conexiones (sólo aplicaciones de servidor)
SocketAccept	Aceptación de conexiones (sólo aplicaciones de servidor)

Función	Se usa para:
SocketGetStatus	Obtención del estado actual del zócalo
SocketPeek	Prueba para comprobar la presencia de datos en un zócalo

Tipo de dato	Para definir
socketdev	Dispositivo de zócalo
socketstatus	Estado del zócalo

Comunicación mediante RAPID Message Queues

Tipo de dato	Para definir
rmqheader ⁱ	El componente rmqheader es una parte del tipo de dato rmq-message y se usa para describir el mensaje.
rmqmessage	Un contenedor general de datos utilizado para comunicarse con la funcionalidad de RAPID Message Queue.
rmqslot	El número de identidad de una tarea de RAPID o un cliente de Robot Application Builder.
IRMQMessage	Ordenar y habilitar interrupciones para un tipo de dato en concreto
RMQFindSlot	Buscar el número de identidad de la cola configurada para una tarea de RAPID o un cliente de Robot Application Builder.
RMQGetMessage	Obtener el primer mensaje de la cola de esta tarea.
RMQGetMsgData	Extraer los datos de un mensaje
RMQGetMsgHeader	Extraer la información de encabezado de un mensaje
RMQSendMessage	Enviar datos a la cola configurada para una tarea de RAPID o un cliente de SDK
RMQSendWait	Enviar un mensaje y esperar una respuesta
RMQEmptyQueue	Vaciar la cola de RMQ conectada a la instrucción de ejecución de la tarea.
RMQReadWait	Esperar hasta que llegue un mensaje, o en caso contrario tiempo límite agotado.

ⁱ Sólo si el robot está equipado al menos con una de las opciones *FlexPendant Interface*, *PC Interface* o *Multitasking*.

Continúa en la página siguiente

Función	Se usa para:
RMQGetSlotName ⁱ	Obtener el nombre de un cliente de RAPID Message Queue a partir de un número de identidad determinado, es decir, un <code>rmqslot</code> determinado

ⁱ Sólo si el robot está equipado al menos con una de las opciones *FlexPendant Interface*, *PC Interface* o *Multitasking*.

1.11 Interrupciones

Introducción

Las interrupciones son eventos definidos por el programa e identificados por *números de interrupción*. Las interrupciones se producen siempre que una *condición de interrupción* resulta verdadera. Al contrario que los errores, el punto en el que se produce una interrupción no está relacionado directamente (sincronizado con) ninguna posición específica del código. La aparición de una interrupción provoca la suspensión de la ejecución normal del programa y su control se entrega a una *rutina TRAP*.

A pesar de que el robot detecta inmediatamente la aparición de una interrupción (sólo retrasada por la velocidad del hardware), su respuesta (mediante una llamada a la rutina TRAP correspondiente) sólo puede tener lugar en posiciones concretas del programa. Se trata de las siguientes:

- Cuando se entra en la siguiente instrucción
- En cualquier momento durante la ejecución de una instrucción de espera, es decir, `WaitUntil`.
- En cualquier momento durante la ejecución de una instrucción de movimiento, es decir, `MoveL`.

Normalmente, esto tiene como resultado un retardo de entre 2 y 30 ms entre la detección de la interrupción y la respuesta, en función del tipo de movimiento que se esté realizando en el momento de la interrupción.

La elevación de interrupciones puede estar *desactivada* o *activada*. Si las interrupciones están desactivadas, las interrupciones que se produzcan se almacenan en una cola y no se elevan hasta que se activen de nuevo las interrupciones. Recuerde que la cola de interrupciones puede contener más de una interrupción en espera. Las interrupciones de la cola se elevan en orden *FIFO* (primero en entrar, primero en salir). Las interrupciones siempre están desactivadas durante la ejecución de una rutina TRAP.

Las interrupciones no se manejan si el programa está en el modo paso a paso o si está detenido. Las interrupciones presentes en la cola en el momento de paro son desechadas y las interrupciones generadas durante el paro no se procesan, excepto en el caso de las interrupciones seguras. Consulte [Safe Interrupt en la página 76](#).

El número máximo de definiciones de interrupción en un momento dado está limitado a 100 por cada tarea de programa.

Principios de programación

Cada interrupción tiene asignada una identidad de interrupción. Obtiene su identidad mediante la creación de una variable (del tipo de datos `intnum`) y conectándola a una rutina TRAP.

A partir de ese momento, la identidad de interrupción (variable) se utiliza para solicitar una interrupción, es decir, para especificar el motivo de la interrupción. Puede tratarse de uno de los eventos siguientes:

- Cambio del valor de una entrada o una salida a uno o a cero.

Continúa en la página siguiente

- Transcurre un intervalo determinado después de la petición de una interrupción.
- Se alcanza una posición determinada.

Cuando se solicita una interrupción, ésta queda activada automáticamente, pero es posible desactivarla temporalmente. Esto puede realizarse de dos formas:

- Es posible desactivar todas las interrupciones. Todas las interrupciones que se produzcan durante este intervalo se sitúan en una cola y se generan automáticamente cuando se activen de nuevo las interrupciones.
- Es posible desactivar interrupciones concretas. Las interrupciones que se produzcan durante este intervalo se descartan.

Conexión de interrupciones a rutinas TRAP

Instrucción	Se usa para:
CONNECT	Conectar una variable (una identidad de interrupción) a una rutina TRAP

Petición de interrupciones

Instrucción	Para solicitar
ISignalDI	Una interrupción desde una señal digital de entrada
ISignalDO	Una interrupción desde una señal digital de salida
ISignalGI	Una interrupción de un grupo de señales digitales de entrada
ISignalGO	Una interrupción de un grupo de señales digitales de salida
ISignalAI	Una interrupción desde una señal analógica de entrada
ISignalAO	Una interrupción desde una señal analógica de salida
ITimer	Una interrupción temporizada
TriggInt	Una interrupción relacionada con una posición (de la lista de selección de movimientos)
IPers	Una interrupción al cambiar una variable persistente.
IError	Pedir y activar una interrupción cuando se produce un error
IRMQMessage ⁱ	Una interrupción cuando se recibe un tipo de dato concreto en una cola de RAPID Message Queue

ⁱ Sólo si el robot está equipado con la opción *FlexPendant Interface*, *PC Interface* o *Multitasking*.

Cancelación de interrupciones

Instrucción	Se usa para:
IDelete	Cancelar (eliminar) una interrupción

Activación y desactivación de interrupciones

Instrucción	Se usa para:
ISleep	Desactivar una interrupción concreta
IWatch	Activar una interrupción concreta
IDisable	Desactivar todas las interrupciones

Continúa en la página siguiente

1 Programación básica en RAPID

1.11 Interrupciones

Continuación

Instrucción	Se usa para:
IEnable	Activar todas las interrupciones

Datos de interrupción

Instrucción	Se usa para:
GetTrapData	En rutinas TRAP, para obtener todos los datos sobre la interrupción que causó la ejecución de la rutina TRAP.
ReadErrData	En rutinas TRAP, para obtener información numérica (dominio, tipo y número) sobre un error, un cambio de estado o una advertencia que provocaron la ejecución de la rutina TRAP.

Tipos de datos de las interrupciones

Tipo de dato	Se usa para:
intnum	Definir la identidad de una interrupción
trapdata	Contener los datos de interrupción que causaron la ejecución de la rutina TRAP actual
errtype	Especificar un tipo de error (gravedad)
errdomain	Pedir y activar una interrupción cuando se produce un error
errdomain	Especificar un dominio de error

Safe Interrupt

Algunas instrucciones, por ejemplo `ITimer` y `ISignalDI`, pueden usarse conjuntamente con Safe Interrupt. Las interrupciones de tipo Safe Interrupt son interrupciones que quedan guardadas en una cola si llegan durante un paro o una ejecución paso a paso. Las interrupciones de las colas se procesan tan pronto como se inicia la ejecución continua y se gestionan en el orden *FIFO*. También se procesan las interrupciones presentes en la cola en el momento del paro. La instrucción `ISleep` no puede usarse junto con las interrupciones seguras.

Manipulación de interrupciones

La definición de una interrupción hace que el sistema la conozca. En la definición se especifica la condición de la interrupción, además de activar y habilitar la interrupción.

Ejemplo:

```
VAR intnum siglint;  
ISignalDI dil, high, siglint;
```

Es posible desactivar una interrupción que está activada (y viceversa).

Durante el periodo en que una interrupción está desactivada, su aparición no se detecta y se desecha, sin ejecutar ninguna rutina TRAP.

Ejemplo:

```
! deactivate  
ISleep siglint;  
  
! activate  
IWatch siglint;
```

Continúa en la página siguiente

Es posible deshabilitar una interrupción que está habilitada (y viceversa).

Durante el periodo en que una interrupción está deshabilitada, las interrupciones generadas del tipo especificado se almacenan en una cola y se elevan tan pronto como las interrupciones se habilitan de nuevo.

Ejemplo:

```
! disable
IDisable siglint;

1 enable
IEnable siglint;
```

La eliminación de una interrupción supone también la eliminación de su definición. No es necesario eliminar explícitamente la definición de una interrupción, pero no es posible definir una interrupción para una variable de interrupción hasta que se elimina la definición anterior.

Ejemplo:

```
IDelete siglint;
```

Rutinas TRAP

Las rutinas TRAP proporcionan una forma de responder a las interrupciones. Para conectar una rutina TRAP a una interrupción determinada, se utiliza la instrucción **CONNECT**. Cuando se produce una interrupción, el control se transfiere inmediatamente a la rutina TRAP asociada (si la hay). Si se produce una interrupción que no tiene ninguna rutina TRAP conectada, el evento se trata como un error no recuperable, es decir, que causa la finalización inmediata de la ejecución del programa.

Ejemplo:

```
VAR intnum empty;
VAR intnum full;
PROC main()
  ! Connect trap routines
  CONNECT empty WITH etrap;
  CONNECT full WITH ftrap;
  ! Define feeder interrupts
  ISignalDI di1, high, empty;
  ISignalDI di3, high, full;
  ...
  ! Delete interrupts
  IDelete empty;
  IDelete full;
ENDPROC
! Responds to "feeder empty" interrupt
TRAP etrap
  open_valve;
  RETURN;
ENDTRAP
! Responds to "feeder full" interrupt
TRAP ftrap
  close_valve;
```

Continúa en la página siguiente

1 Programación básica en RAPID

1.11 Interrupciones

Continuación

```
    RETURN;  
ENDTRAP
```

Es posible conectar varias interrupciones a una misma rutina TRAP. La variable de sistema `INTNO` contiene el número de la interrupción y puede usarse para identificar una interrupción desde una rutina TRAP. Una vez tomadas las acciones necesarias, es posible finalizar la rutina TRAP mediante la instrucción `RETURN` o cuando se alcanza el final (`ENDTRAP` o `ERROR`) de la rutina TRAP. La ejecución continúa en el punto en el que se produjo la interrupción.

1.12 Recuperación en caso de error

Introducción

Muchos de los errores que se producen cuando se está ejecutando un programa pueden gestionarse desde el programa, lo que significa que no es necesario interrumpir la ejecución. Estos errores son de un tipo detectado por el sistema, como por ejemplo la división por cero, o de un tipo elevado por el programa, por ejemplo cuando un programa eleva un error al obtenerse un valor incorrecto de un lector de códigos de barras.

Los errores de ejecución son situaciones anormales relacionadas con la ejecución de un fragmento determinado de un programa. Los errores impiden que la ejecución continúe (o que esta resulte peligrosa). Algunos de los errores posibles son el “desbordamiento de memoria” y la “división entre cero”.

Números de errores

Los errores se identifican por su número de error exclusivo y son siempre detectados por el sistema. La aparición de un error provoca la suspensión de la ejecución normal del programa y el control se entrega a un gestor de errores. El concepto de los gestores de errores es que sea posible responder a los errores que se producen durante la ejecución y, si es posible, recuperarse tras el error. Si no es posible continuar con la ejecución, al menos el gestor de errores puede hacer que el programa se detenga de la forma más correcta posible.

Principios de programación

Cuando se produce un error, se realiza una llamada al gestor de la rutina (si lo hay). También es posible crear un error desde el propio programa y saltar a continuación hacia el gestor de errores.

En los gestores de errores, es posible manejar los errores mediante instrucciones convencionales. El dato de sistema `ERRNO` puede usarse para determinar el tipo de error que se ha producido. Posteriormente, el retorno desde el gestor de errores puede realizarse de varias formas (`RETURN`, `RETRY`, `TRYNEXT` y `RAISE`).

Si la rutina actual no contiene un gestor de errores, el gestor de errores interno del robot toma directamente el control. El gestor de errores interno genera un mensaje de error y detiene la ejecución del programa, dejando el puntero del programa en la instrucción que provoca el problema.

Creación de una situación de error desde la instrucción de programa

Instrucción	Se usa para:
<code>RAISE</code>	“Crear” un error y llamar al gestor de errores

Registro de una instrucción de un número de error

Instrucción	Se usa para:
<code>BookErrNo</code>	Registrar un nuevo número de error de sistema de RAPID.

Continúa en la página siguiente

1 Programación básica en RAPID

1.12 Recuperación en caso de error

Continuación

Reinicio y retorno desde el gestor de errores

Instrucción	Se usa para:
EXIT	Detener la ejecución del programa en caso de un error no recuperable.
RAISE	Llamar al gestor de errores de la rutina que llamó a la rutina actual.
RETRY	Ejecutar de nuevo la instrucción que causó el error.
TRYNEXT	Ejecutar la instrucción siguiente a la instrucción que causó el error.
RETURN	Volver a la rutina que llamó a la rutina actual.
RaiseToUser	Desde una rutina NOSTEPIN, el error es elevado por el gestor de errores en el nivel de usuario.
StartMoveRetry	Una instrucción que sustituye al par de instrucciones StartMove y RETRY. Reanuda los movimientos y vuelve a ejecutar además la instrucción que generó el error.
SkipWarn	Omitir el último mensaje de advertencia pedido.
ResetRetryCount	Restablecer el número de reintentos contado.
Función	Se usa para:
RemainingRetries	Reintentos restantes aún pendientes.

Generación de errores de proceso

Instrucción	Se usa para:
ErrLog	Mostrar un mensaje de error en el FlexPendant y registrarlo en el registro de mensajes del robot.
ErrRaise	Crear un error en el programa y llamar a continuación al gestor de errores de la rutina.
Función	Se usa para:
ProcErrRecovery	Generar un error de proceso durante el movimiento del robot.

Datos para la gestión de errores

Tipo de dato	Se usa para:
errnum	El motivo del error
errstr	El texto de un mensaje de error

Configuración para la gestión de errores

Parámetro del sistema	Para definir
No Of Retry	El número de veces que se debe reintentar una instrucción fallida si el gestor de errores utiliza RETRY. No Of Retry pertenece al tipo System Misc del tema Controller.

Continúa en la página siguiente

Gestores de errores

Cualquier rutina puede contener un gestor de errores. El gestor de errores es en realidad parte de la rutina, y el ámbito de los datos de la rutina incluye también el gestor de errores de la propia rutina. Si se produce un error durante la ejecución de la rutina, el control se transfiere a su gestor de errores.

Ejemplo:

```
FUNC num safediv( num x, num y)
  RETURN x / y;
ERROR
  IF ERRNO = ERR_DIVZERO THEN
    TPWrite "The number cannot be equal to 0";
    RETURN x;
  ENDIF
ENDFUNC
```

La variable de sistema `ERRNO` contiene el número del error (más reciente) y puede usarse en el gestor de errores para identificar el error. Después de tomar todas las acciones oportunas, el gestor de errores puede:

- Reanudar la ejecución, comenzando por la instrucción en la que se produjo el error. Esto se hace mediante la instrucción `RETRY`. Si esta instrucción causa de nuevo el mismo error, se realizarán un máximo de cuatro recuperaciones, tras lo cual se detiene la ejecución. Para poder realizar más de cuatro reintentos, tiene que configurar el parámetro del sistema *No Of Retry*. Consulte *Manual de referencia técnica - Parámetros del sistema*.
- Reanudar la ejecución, comenzando por la instrucción siguiente a la instrucción en la que se produjo el error. Esto se hace mediante la instrucción `TRYNEXT`.
- Devolver el control a la rutina desde la que se llamó a la rutina actual. Para ello se utiliza la instrucción `RETURN`. Si la rutina es una función, la instrucción `RETURN` debe especificar un valor de retorno adecuado.
- Propagar el error a la rutina desde la que se llamó a la rutina actual. Para ello se utiliza la instrucción `RAISE`.

Gestor de errores del sistema

Cuando se produce 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`), se llama al *gestor de errores del sistema*. El gestor de errores del sistema sólo informa del error y detiene la ejecución.

En una cadena de llamadas a rutinas, cada una de ellas puede tener su propio gestor de errores. Si se produce un error en una rutina que tiene gestor de errores y el error se propaga explícitamente mediante la instrucción `RAISE`, el mismo error se eleva de nuevo en el punto de la llamada a la rutina (el error se *propaga*). Cuando se alcanza la parte superior de la cadena de llamadas (la rutina de entrada de la tarea) sin que se encuentre ningún gestor de errores o cuando se alcanza el final de cualquiera de los gestores de errores pertenecientes a la cadena de llamadas, se llama al gestor de errores del sistema. El gestor de errores del sistema sólo informa del error y detiene la ejecución. Dado que las llamadas a las rutinas `TRAP`

Continúa en la página siguiente

1 Programación básica en RAPID

1.12 Recuperación en caso de error

Continuación

sólo pueden hacerse desde el sistema (como respuesta a una interrupción), cualquier propagación de un error desde una rutina TRAP se realiza al gestor de errores del sistema.

La recuperación de errores no está disponible en el caso de las instrucciones del gestor de ejecución hacia atrás. Este tipo de errores se propaga siempre hacia el gestor de errores del sistema.

No es posible recuperar la ejecución tras los errores que se producen dentro de un gestor de errores. Este tipo de errores se propaga siempre hacia el gestor de errores del sistema.

Errores elevados por el programa

Además de los errores detectados y elevados por el robot, los programas pueden elevar explícitamente los errores mediante la instrucción RAISE. Esta posibilidad puede usarse para recuperar la ejecución en situaciones complejas. Por ejemplo, puede usarse para escapar de posiciones de código anidadas a gran profundidad. Es posible usar los números de error del 1 al 90 junto con la instrucción RAISE. Los errores elevados explícitamente de esta forma se tratan exactamente igual que los errores elevados por el sistema.

El registro de eventos

Los errores manejados por un gestor de errores siguen provocando un aviso en el registro de eventos. Mediante la consulta del registro de eventos es posible controlar qué errores se han producido.

Si desea que un error se maneje sin escribir ningún aviso en el registro de eventos, utilice la instrucción `SkipWarn` en el gestor de errores. Esto puede resultar útil si se utiliza el gestor de errores para probar algún hecho (por ejemplo, si existe un archivo determinado) sin dejar ninguna huella si la prueba no da un resultado positivo.

1.13 UNDO

Introducción

Las rutinas de RAPID pueden contener un gestor `UNDO`. El gestor se ejecuta automáticamente si el puntero del programa se mueve hacia el exterior de la rutina. Se ha previsto para eliminar los efectos secundarios resultantes de las rutinas ejecutadas parcialmente, por ejemplo, al cancelar instrucciones modales (por ejemplo, abrir un archivo). La mayor parte del lenguaje de RAPID puede utilizarse en un gestor `UNDO`, pero existen algunas limitaciones, por ejemplo en las instrucciones de movimiento.

Terminología

Los siguientes términos se refieren a `UNDO`.

- **UNDO:** la ejecución del código de limpieza antes de un restablecimiento del programa.
- **Gestor UNDO:** una parte opcional de un procedimiento o función de RAPID que contiene el código de RAPID que se ejecuta en `UNDO`.
- **Rutina UNDO:** un procedimiento o una función con un gestor `UNDO`.
- **Cadena de llamadas:** todos los procedimientos o funciones asociados actualmente entre sí a través de ejecuciones de rutina que no hayan finalizado aún. Se supone que se empieza en la rutina `Main` si no se especifica lo contrario.
- **Contexto UNDO:** cuando la rutina actual es parte de una cadena de llamadas que se inicia en un gestor `UNDO`.

Cuándo utilizar UNDO

Una rutina de RAPID puede anularse en cualquier punto con sólo situar el puntero de programa fuera de la rutina. En algunos casos, cuando el programa está ejecutando ciertas rutinas sensibles, no es posible anularlas. Es posible utilizar `UNDO` para proteger esas rutinas sensibles frente a un restablecimiento inesperado del programa. Con `UNDO` es posible conseguir que un código determinado se ejecute automáticamente si se anula la rutina. Este código debe realizar normalmente acciones de limpieza, por ejemplo, cerrar un archivo.

Comportamiento detallado de UNDO

Cuando se activa `UNDO`, se ejecutan todos los gestores `UNDO` de la cadena de llamadas actual. Estos gestores son partes opcionales de un procedimiento o una función de RAPID que contiene código de RAPID. Los gestores `UNDO` activos actualmente son aquellos que pertenecen a procedimientos o funciones que han iniciado su ejecución y no han terminado, es decir, las rutinas de la cadena de llamadas actual.

`UNDO` se activa cuando el puntero del programa se mueve inesperadamente hacia el exterior de una rutina `UNDO`, por ejemplo, si el usuario mueve el puntero del programa a `Main`. `UNDO` se inicia también si se ejecuta una instrucción `EXIT`, lo que hace que el programa se restablezca, o si el programa se restablece por alguna otra razón, por ejemplo, al cambiar cierta configuración o si se elimina el programa

Continúa en la página siguiente

1 Programación básica en RAPID

1.13 UNDO

Continuación

o el módulo. Sin embargo, UNDO no se inicia si el programa alcanza el final de la rutina o una sentencia `RETURN` y retorna de la forma normal desde la rutina.

Si hay más de una rutina UNDO en la cadena de llamadas, los gestores UNDO de las rutinas se procesarán en el mismo orden con el que habrían retornado las distintas rutinas, de abajo arriba. El gestor UNDO más cercano al final de la cadena de llamadas se ejecutará en primer lugar y el más cercano a Main se ejecutará en último lugar.

Limitaciones

Un gestor UNDO puede acceder a cualquier variable o símbolo al que se pueda llegar desde el cuerpo de la rutina normal, incluidas las variables declaradas localmente. Sin embargo, el código de RAPID que se va a ejecutar en contexto UNDO tiene ciertas limitaciones.

El gestor UNDO no debe contener `STOP`, `BREAK`, `RAISE` ni `RETURN`. Si se hace un intento de utilizar algunas de estas instrucciones en el contexto UNDO, la instrucción no se tendrá en cuenta y se generará un aviso de ELOG.

Las instrucciones de movimiento, por ejemplo `MoveL`, tampoco están permitidas en el contexto UNDO.

La ejecución es siempre continua en UNDO. No es posible ejecutar paso a paso. Cuando se inicia UNDO, el modo de ejecución cambia automáticamente a continuo. Una vez terminada la sesión de UNDO, se restablece el modo de ejecución anterior.

Si el programa se detiene durante la ejecución de un gestor UNDO, el resto del gestor no se ejecutará. Si hay gestores UNDO adicionales aún por ejecutar en la cadena de llamadas, tampoco se tendrán en cuenta. Esto dará lugar a un aviso de ELOG. Además se detiene debido a un error en tiempo de ejecución.

El puntero del programa no está visible en un gestor UNDO. Cuando se ejecuta UNDO, el puntero del programa permanece en su anterior ubicación, pero se actualiza cuando finalizan los gestores UNDO.

Una instrucción `EXIT` anula al UNDO de la misma forma que un error de tiempo de ejecución o un `Stop`. El resto de gestores UNDO no se tiene en cuenta y el puntero del programa se mueve a Main.

Ejemplo

El programa:

```
PROC B
  TPWrite "In Routine B";
  Exit;
UNDO
  TPWrite "In UNDO of routine B";
ENDPROC

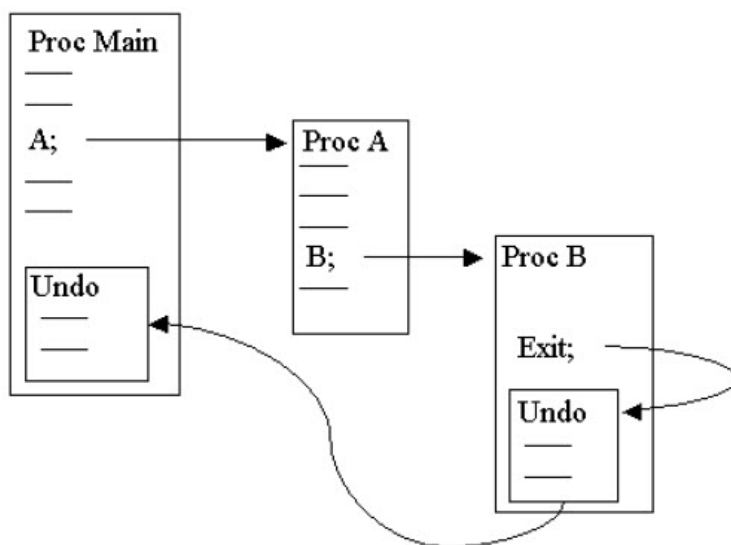
PROC A
  TPWrite "In Routine A";
  B;
ENDPROC
```

Continúa en la página siguiente

```
PROC main
  TPWrite "In main";
  A;
UNDO
  TPWrite "In UNDO of main";
ENDPROC
```

La salida:

```
In main
In Routine A
In Routine B
In UNDO of routine B
In UNDO of main
```



xx110000588

1 Programación básica en RAPID

1.14 Sistema y tiempo

1.14 Sistema y tiempo

Descripción

Las instrucciones de sistema y tiempo permiten al usuario medir, inspeccionar y registrar tiempos.

Principios de programación

Las instrucciones de reloj permiten utilizar relojes que funcionan en forma de cronómetros. De esta forma, es posible utilizar el programa del robot para temporizar cualquier evento que se desee.

Es posible obtener la hora o la fecha actuales en forma de una cadena de caracteres. A continuación, es posible mostrar esta cadena de caracteres al usuario en la pantalla del FlexPendant o utilizarla para almacenar la fecha y la hora junto con archivos de registro.

También es posible obtener componentes de la hora actual del sistema en forma de valores numéricos. Esto permite al programa del robot realizar una acción a una hora determinada o en un día determinado de la semana.

Utilización de un reloj para temporizar un evento

Instrucción	Se usa para:
ClkReset	Poner a cero un reloj utilizado para la temporización
ClkStart	Poner en marcha un reloj utilizado para la temporización
ClkStop	Detener un reloj utilizado para la temporización
Función	Se usa para:
ClkRead	Leer un reloj utilizado para la temporización
Tipo de dato	Se usa para:
clock	Temporización: almacena una medición de tiempo, en segundos

Lectura de la hora y la fecha actuales

Función	Se usa para:
CDate	Obtener la fecha actual en forma de cadena de caracteres
CTime	Obtener la hora actual en forma de cadena de caracteres
GetTime	Obtener la hora actual en forma de valor numérico

Obtener información de hora de un archivo

Función	Se usa para:
FileTime	Obtener la última hora de modificación de un archivo
ModTime	Obtener la hora de modificación del módulo cargado
ModExist	Comprobar si un módulo de programa existe.

Continúa en la página siguiente

Obtener el tamaño de memoria libre del programa

Función	Se usa para:
ProgMemFree	Obtener el tamaño de memoria libre del programa

1 Programación básica en RAPID

1.15 Matemáticas

1.15 Matemáticas

Descripción

Las instrucciones y funciones matemáticas se utilizan para calcular y cambiar el valor de la información.

Principios de programación

Los cálculos suelen realizarse mediante la instrucción de asignación, por ejemplo, `reg1:= reg2 + reg3 / 5`. También existen algunas instrucciones que se utilizan para cálculos simples, como eliminar el valor de una variable numérica.

Cálculos sencillos con datos numéricos

Instrucción	Se usa para:
Clear	Eliminar el valor
Add	Sumar o restar un valor
Incr	Incrementar en 1
Decr	Reducir en 1

Cálculos más avanzados

Instrucción	Se usa para:
<code>:=</code>	Realizar cálculos con cualquier tipo de dato.

Funciones aritméticas

Función	Se usa para:
Abs	Calcular el valor absoluto
AbsDnum	Calcular el valor absoluto
Round	Redondear un valor numérico
RoundDnum	Redondear un valor numérico
Trunc	Truncar un valor numérico
TruncDnum	Truncar un valor numérico
Sqrt	Calcular la raíz cuadrada
SqrtDnum	Calcular la raíz cuadrada
Exp	Calcular el valor exponencial en base "e"
Pow	Calcular el valor exponencial en la base deseada
PowDnum	Calcular el valor exponencial en la base deseada
ACos	Calcular el arco coseno
ACosDnum	Calcular el arco coseno
ASin	Calcular el arco seno
ASinDnum	Calcular el arco seno
ATan	Calcular el arco tangente en el rango [-90,90]

Continúa en la página siguiente

Función	Se usa para:
ATanDnum	Calcular el arco tangente en el rango [-90,90]
ATan2	Calcular el arco tangente en el rango [-180,180]
ATan2Dnum	Calcular el arco tangente en el rango [-180,180]
Cos	Calcular el coseno
CosDnum	Calcular el coseno
Sin	Calcular el seno
SinDnum	Calcular el seno
Tan	Calcular la tangente
TanDnum	Calcular la tangente
EulerZYX	Calcular ángulos de Euler a partir de una orientación
OrientZYX	Calcular la orientación a partir de ángulos de Euler
PoseInv	Invertir una pose
PoseMult	Multiplicar una pose
PoseVect	Multiplicar una pose y un vector
Vectmagn	Calcular la magnitud de un vector pos
DotProd	Calcular el producto escalar de dos vectores pos
NOrient	Normalizar una orientación no normalizada (cuaternio)

Funciones para dígitos de cadenas de caracteres

Función	Se usa para:
StrDigCmp	Comparación numérica de dos cadenas de caracteres que sólo contienen dígitos
StrDigCalc	Operaciones aritméticas en dos cadenas sólo con dígitos
Tipo de dato	Se usa para:
stringdig	Cadena de caracteres con sólo dígitos

Funciones de bits

Instrucción	Se usa para:
BitClear	Eliminar un bit especificado en un dato de byte o dnum definido.
BitSet	Cambiar a 1 bit especificado en un dato de byte o dnum definido.
Función	Se usa para:
BitCheck	Comprobar si un bit especificado de un dato de byte definido tiene el valor 1.
BitCheckDnum	Comprobar si un bit especificado de un dato de dnum definido tiene el valor 1.
BitAnd	Ejecutar una operación lógica bit a bit <i>AND</i> en tipos de datos byte.

Continúa en la página siguiente

1 Programación básica en RAPID

1.15 Matemáticas

Continuación

Función	Se usa para:
BitAndDnum	Ejecutar una operación lógica bit a bit <i>AND</i> en tipos de datos dnum.
BitNeg	Ejecutar una operación lógica bit a bit <i>NEGATION</i> en tipos de datos byte.
BitNegDnum	Ejecutar una operación lógica bit a bit <i>NEGATION</i> en tipos de datos dnum.
BitOr	Ejecutar una operación lógica bit a bit <i>OR</i> en tipos de datos byte.
BitOrDnum	Ejecutar una operación lógica bit a bit <i>OR</i> en tipos de datos dnum.
BitXOr	Ejecutar una operación lógica bit a bit <i>XOR</i> en tipos de datos byte.
BitXOrDnum	Ejecutar una operación lógica bit a bit <i>XOR</i> en tipos de datos dnum.
BitLSh	Ejecutar una operación lógica bit a bit de <i>DESPLAZAMIENTO A LA IZQUIERDA</i> en tipos de datos byte.
BitLShDnum	Ejecutar una operación lógica bit a bit de <i>DESPLAZAMIENTO A LA IZQUIERDA</i> en tipos de datos dnum.
BitRSh	Ejecutar una operación lógica bit a bit de <i>DESPLAZAMIENTO A LA DERECHA</i> en tipos de datos byte.
BitRShDnum	Ejecutar una operación lógica bit a bit de <i>DESPLAZAMIENTO A LA DERECHA</i> en tipos de datos dnum.

Tipo de dato	Se usa para:
byte	Se utiliza junto con instrucciones y funciones de manipulación de bits (8 bits).
dnum	Se utiliza junto con instrucciones y funciones de manipulación de bits (52 bits).

1.16 Comunicación con un ordenador externo

Descripción

Es posible controlar el robot desde un ordenador supervisor. En este caso, se utiliza un protocolo de comunicaciones especial para transferir la información.

Principios de programación

Dado que se utiliza un protocolo de comunicaciones convencional para transferir la información entre el robot y el ordenador, éstos pueden comprenderse mutuamente y no se requiere programación adicional. Por ejemplo, el ordenador puede cambiar valores de los datos del programa sin necesidad de programación (excepto a la hora de definir esta información). Sólo se requiere programación si es necesario enviar información controlada por el programa desde el robot al ordenador supervisor.

Envío de un mensaje controlado por el programa del robot a un ordenador

Instrucción	Se usa para:
SCWrite ⁱ	Enviar un mensaje al ordenador supervisor

ⁱ Sólo si el robot está equipado con la opción *PC interface/backup*.

1 Programación básica en RAPID

1.17 Funciones para operaciones con archivos

1.17 Funciones para operaciones con archivos

Instrucciones

Instrucción	Se usa para:
MakeDir	Crear un nuevo directorio.
RemoveDir	Eliminar un directorio.
OpenDir	Abrir un directorio para su examen posterior.
CloseDir	Cerrar un directorio como operación opuesta de OpenDir.
RemoveFile	Eliminar un archivo.
RenameFile	Cambiar el nombre de un archivo.
CopyFile	Copiar un archivo.

Funciones

Función	Se usa para:
ISFile	Comprobar el tipo de un archivo.
FSSize	Obtener el tamaño de un sistema de archivos.
FileSize	Obtener el tamaño de un archivo específico.
ReadDir	Leer la siguiente entrada de un directorio.

Tipos de datos

Tipo de dato	Se usa para:
dir	Recorrer estructuras de directorios.

1.18 Instrucciones de soporte de RAPID

Descripción

Existen varias funciones que soportan el lenguaje RAPID:

- Obtención de datos del sistema
- Lectura de datos de configuración
- Escritura de datos de configuración
- Reinicio del controlador
- Comprobación de datos del sistema
- Obtención de nombres de objetos
- Obtención de nombres de tareas
- Búsqueda de símbolos
- Obtención del tipo de evento actual, el gestor de ejecución o el nivel de ejecución
- Lectura de información de servicio

Obtención de datos del sistema

Instrucciones para capturar el valor y (opcionalmente) el nombre del símbolo del dato de sistema actual del tipo especificado.

Instrucción	Se usa para:
GetSysData	Obtener los datos y el nombre de la herramienta o del objeto de trabajo actuales.
ResetPPMoved	Restablecer el estado del puntero de programa movido en el modo manual.
SetSysData	Activar un nombre de dato de sistema especificado para un tipo de datos especificado.

Función	Se usa para:
IsSysID	Comprobar la identidad del sistema.
IsStopStateEvent	Obtener información acerca del movimiento del puntero de programa (PP).
PPMovedInManMode	Comprobar si el puntero de programa se ha movido en el modo manual.
RobOS	Comprobar si la ejecución se realiza en el controlador de robot (RC) o el controlador virtual (VC).

Obtención de información acerca del sistema

Función para obtener información acerca del número de serie, versión de software, tipo de robot, dirección IP de la red local o lenguaje del controlador.

Función	Se usa para:
GetSysInfo	Obtener información acerca del sistema.

Continúa en la página siguiente

1 Programación básica en RAPID

1.18 Instrucciones de soporte de RAPID

Continuación

Obtención de información acerca de la memoria

Función	Se usa para:
ProgMemFree	Obtener el tamaño de memoria libre del programa

Lectura de datos de configuración

Instrucciones para leer un atributo de un parámetro de sistema cuyo nombre se indica.

Instrucción	Se usa para:
ReadCfgData	Leer un atributo de un parámetro de sistema cuyo nombre se indica.

Escritura de datos de configuración

Instrucciones para escribir un atributo de un parámetro de sistema cuyo nombre se indica.

Instrucción	Se usa para:
WriteCfgData	Escribir un atributo de un parámetro de sistema cuyo nombre se indica.

Guardado de datos de configuración

Una instrucción que permite guardar los parámetros del sistema en un archivo.

Instrucción	Se usa para:
SaveCfgData	Guardar parámetros del sistema a un archivo

Reinicio del controlador

Instrucción	Se usa para:
WarmStart	Reiniciar el controlador, por ejemplo después de cambiar los parámetros de sistema desde RAPID.

Instrucciones para tablas de texto

Instrucción	Se usa para:
TextTabInstall	Instalar una tabla de textos en el sistema.

Función	Se usa para:
TextTabGet	Obtener el número de tabla de textos de una tabla de textos definida por el usuario.
TextGet	Obtener una cadena de caracteres de las tablas de texto del sistema.
TextTabFreeToUse	Comprobar si el nombre de la tabla de textos (cadena de caracteres de recursos de texto) está disponible para su uso.

Continúa en la página siguiente

Obtención de nombres de objetos

Una instrucción para obtener el nombre de un objeto de datos original para un argumento actual o un dato actual.

Instrucción	Se usa para:
ArgName	Obtener el nombre del objeto de datos original.

Obtención de información acerca de las tareas

Función	Se usa para:
GetTaskName	Obtener la identidad de la tarea de programa actual, con su nombre y número.
MotionPlannerNo	Obtener el número del planificador de movimientos actual.

Obtención del tipo de evento actual, el gestor de ejecución o el nivel de ejecución

Función	Se usa para:
EventType	Obtener el tipo de la rutina de evento actual.
ExecHandler	Obtener el tipo de gestor de ejecución.
ExecLevel	Obtener el nivel de ejecución.

Tipo de dato	Se usa para:
event_type	Tipo de rutina de evento.
handler_type	Tipo de gestor de ejecución.
exec_level	Nivel de ejecución.

Búsqueda de símbolos

Instrucciones para buscar objetos de datos en el sistema

Instrucción	Se usa para:
SetAllDataVal	Establecer un nuevo valor en todos los objetos de datos de un tipo determinado y que coincidan con una gramática determinada.
SetDataSearch	Junto con GetNextSym, permite obtener objetos de datos del sistema.
GetDataVal	Obtener un valor de un objeto de datos que se especifica mediante una variable de cadena de caracteres.
SetDataVal	Establecer un valor para un objeto de datos que se especifica mediante una variable de cadena de caracteres.

Función	Se usa para:
GetNextSym	Junto con SetDataSearch, permite obtener objetos de datos del sistema.

Tipo de dato	Se usa para:
datapos	Contiene información acerca de dónde está definido un objeto determinado dentro del sistema.

Continúa en la página siguiente

1 Programación básica en RAPID

1.18 Instrucciones de soporte de RAPID

Continuación

Lectura de información de servicio

Instrucción	Se usa para:
GetServiceInfo	Leer información de servicio del sistema.

1.19 Calibración y servicio

Descripción

Existen varias instrucciones destinadas a la calibración y la comprobación del sistema de robot.

Calibración de la herramienta

Instrucción	Se usa para:
MToolRotCalib	Calibrar la rotación de una herramienta móvil
MToolTCPCalib	Calibrar el TCP (punto central de la herramienta) de una herramienta móvil
SToolRotCalib	Calibrar el TCP (punto central de la herramienta) y la rotación de una herramienta estacionaria
SToolTCPCalib	Calibrar el TCP (punto central de la herramienta) de una herramienta estacionaria

Distintos métodos de calibración

Función	Se usa para:
CalcRotAxisFrame	Calcular el sistema de coordenadas del usuario de un tipo de eje giratorio.
CalcRotAxFrameZ	Calcular el sistema de coordenadas del usuario de un tipo de eje giratorio cuando el robot principal y el eje adicional se encuentran en tareas diferentes de RAPID.
DefAccFrame	Definir una base de coordenadas desde las posiciones originales y las desplazadas.

Envío de un valor a la señal de test del robot

Es posible enviar una señal de referencia, por ejemplo la velocidad de un motor, a una señal analógica de salida situada en la tarjeta de bus del robot.

Instrucción	Se usa para:
TestSignDefine	Definir una señal de test
TestSignReset	Restablecer todas las definiciones de señales de test
Función	Se usa para:
TestSignRead	Leer el valor de una señal de test
Tipo de dato	Se usa para:
testsignal	Para la instrucción de programación TestSignDefine

Continúa en la página siguiente

1 Programación básica en RAPID

1.19 Calibración y servicio

Continuación

Registro de una ejecución

La información registrada se almacena en un archivo para su análisis posterior y se utiliza en labores de depuración de programas de RAPID, específicamente en sistemas multitarea.

Instrucción	Se usa para:
SpyStart	Iniciar la grabación de instrucciones y datos de tiempo durante la ejecución.
SpyStop	Detener la grabación de datos de tiempo durante la ejecución.

1.20 Funciones para cadenas de caracteres

Descripción

Las funciones para cadenas de caracteres se utilizan en operaciones con cadenas de caracteres, como copia, concatenación, comparación, búsqueda, conversión, etc.

Operaciones básicas

Tipo de dato	Se usa para:
string	Cadena. Constantes predefinidas: STR_DIGIT, STR_UPPER, STR_LOWER y STR_WHITE.
Instrucción/operador	Se usa para:
:=	Asignar un valor (una copia de la cadena de caracteres)
+	Concatenación de cadenas
Función	Se usa para:
StrLen	Determinar la longitud de una cadena de caracteres
StrPart	Obtener parte de una cadena de caracteres

Comparación y búsqueda

Operador	Se usa para:
=	Comprobar si un valor es igual a otro
<>	Comprobar si un valor es distinto de otro
Función	Se usa para:
StrMemb	Comprobar si un carácter aparece dentro de un conjunto de caracteres
StrFind	Buscar un carácter en una cadena de caracteres
StrMatch	Buscar un patrón dentro de una cadena de caracteres
StrOrder	Comprobar si varias cadenas de caracteres están ordenadas

Conversión

Función	Se usa para:
DnumToNum	Convertir un valor numérico dnum en un valor numérico num
DnumToStr	Convertir un valor numérico en una cadena de caracteres
NumToDnum	Convertir un valor numérico num en un valor numérico dnum
NumToStr	Convertir un valor numérico en una cadena de caracteres
ValToStr	Convertir un valor en una cadena de caracteres
StrToVal	Convertir una cadena de caracteres en un valor
StrMap	Mapear una cadena de caracteres

Continúa en la página siguiente

1 Programación básica en RAPID

1.20 Funciones para cadenas de caracteres

Continuación

Función	Se usa para:
StrToByte	Convertir una cadena de caracteres en un byte
ByteToStr	Convertir un byte en una cadena de caracteres
DecToHex	Convertir un número especificado en una cadena que admita lectura de la base 10 a la base 16
HexToDec	Convertir un número especificado en una cadena que admita lectura de la base 16 a la base 10

1.21 Multitarea

Descripción

Los eventos de una célula de robot suelen producirse en paralelo. Entonces, ¿por qué los programas no funcionan en paralelo?

RAPID en modo multitarea es una forma de ejecutar los programas en (seudo)paralelo. Un programa en paralelo puede situarse en segundo plano o en primer plano respecto de otro programa. También puede situarse en el mismo nivel que otro programa.

Para conocer todos los parámetros, consulte *Manual de referencia técnica - Parámetros del sistema*.

Limitaciones

Existen determinadas restricciones en el uso de Multitasking RAPID.

- No combine programas en paralelo con un PLC. El tiempo de respuesta es el mismo que el tiempo de respuesta a interrupciones de una tarea. Obviamente, esto es cierto cuando la tarea no se encuentra en segundo plano de otro programa ocupado.
- Cuando se ejecuta una instrucción `Wait` en el modo manual, aparece un cuadro de simulación después de 3 segundos. Esto sólo se produce en las tareas de tipo **NORMAL**.
- Las instrucciones de movimiento sólo pueden ejecutarse en la tarea de movimiento (la tarea enlazada con la instancia 0 de programa, consulte *Manual de referencia técnica - Parámetros del sistema*).
- La ejecución de una tarea se detiene durante el tiempo en el que otras tareas están utilizando el sistema de archivos, es decir, cuando el usuario decide guardar o abrir un programa o si el programa de una tarea utiliza las instrucciones `load/erase/read/write`.
- El FlexPendant no tiene acceso a ninguna tarea distinta de una tarea **NORMAL**. Por ello, el desarrollo de programas de RAPID para otras tareas **SEMISTATIC** o **STATIC** sólo se permite si se carga el código en la tarea **NORMAL**, o bien, fuera de línea.

Conceptos básicos

Para usar esta función, es necesario configurar el robot con una TAREA adicional para cada programa adicional. Cada tarea puede ser del tipo **NORMAL**, **STATIC** o **SEMISTATIC**.

Es posible ejecutar hasta 20 tareas diferentes enseudoparalelo. Cada tarea se compone de un conjunto de módulos, de forma muy parecida al programa normal. Todos los módulos se consideran locales dentro de cada tarea.

Las variables, constantes y variables persistentes son locales en cada tarea, pero no así las variables persistentes globales. Una variable persistente es global de forma predeterminada, siempre y cuando no esté declarada como **LOCAL** o **TASK**. Una variable persistente con el mismo nombre y tipo está disponible en todas las tareas en las que esté declarada. Si existen dos variables persistentes globales

Continúa en la página siguiente

1 Programación básica en RAPID

1.21 Multitarea

Continuación

con el mismo nombre, pero tienen un tipo o un tamaño (un número de elementos de matriz) diferente, se produce un error de tiempo de ejecución.

Las tareas tienen su propia gestión de rutinas TRAP y las rutinas de eventos sólo se disparan si lo indica el propio sistema de tarea (por ejemplo, iniciar, detener, reiniciar....).

Instrucciones y funciones generales

Instrucción	Se usa para:
WaitSyncTask ⁱ	Sincronizar varias tareas de programa en un punto especial de cada programa

ⁱ Sólo si el robot está equipado con la opción *MultiTasking*.

Función	Se usa para:
TestAndSet	Obtener el derecho de uso exclusivo de áreas de código concretas de RAPID o recursos del sistema (tipo de sondeo de usuario)
WaitTestAndSet	Obtener el derecho de uso exclusivo de áreas de código concretas de RAPID o recursos del sistema (tipo de control de interrupción)
TaskRunMec	Comprobar si la tarea de programa controla alguna unidad mecánica
TaskRunRob	Comprobar si la tarea de programa controla algún robot con TCP
GetMecUnitName	Obtener el nombre de la unidad mecánica

Tipo de dato	Se usa para:
taskId	Identificar las tareas de programa disponibles en el sistema
syncident	Especificar el nombre de un punto de sincronización
tasks	Especificar varias tareas de programa de RAPID

Sistema MultiMove con robots coordinados

Instrucción	Se usa para:
SyncMoveOn ⁱ	Iniciar una secuencia de movimientos sincronizados
SyncMoveOff	Finalizar el movimiento sincronizado
SyncMoveUndo	Restablecer el movimiento sincronizado

ⁱ Sólo si el robot está equipado con la opción *MultiMove Coordinated*.

Función	Se usa para:
IsSyncMoveOn	Indicar si la tarea actual está en el modo sincronizado
TasksInSync	Devolver el número de tareas sincronizadas

Tipo de dato	Se usa para:
syncident ⁱ	Para especificar el nombre de un punto de sincronización
tasks	Especificar varias tareas de programa de RAPID

Continúa en la página siguiente

Tipo de dato	Se usa para:
identno	Identidad para las instrucciones de movimiento

i Sólo si el robot está equipado con la opción *MultiTasking*.

Sincronización de tareas

En muchas aplicaciones, una tarea en paralelo sólo supervisa alguna unidad de célula, de una forma bastante independiente de las demás tareas que se están ejecutando. En estos casos, no se necesita ningún mecanismo de sincronización. Sin embargo, existen otras aplicaciones que necesitan saber qué está ocurriendo en la tarea principal, por ejemplo.

Sincronización mediante consulta

Esta forma es la más sencilla, pero la que presenta un peor rendimiento. Se utilizan variables persistentes con las instrucciones `WaitUntil`, `IF`, `WHILE` o `GOTO`.

Si se utiliza la instrucción `WaitUntil`, ésta realiza una consulta interna cada 100 ms. No realice consultas con frecuencias mayores en otras aplicaciones.

Ejemplo

TAREA 1:

```
MODULE module1
PERS bool startsync:=FALSE;
PROC main()
    startsync:= TRUE;
ENDPROC
ENDMODULE
```

TAREA 2:

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

Sincronización mediante una interrupción

Se utilizan las instrucciones `SetDO` y `ISignalDO`.

Ejemplo

TAREA 1:

```
MODULE module1
PROC main()
    SetDO do1,1;
ENDPROC
ENDMODULE
```

TAREA 2:

```
MODULE module2
VAR intnum isiint1;
PROC main()
    CONNECT isiint1 WITH isi_trap;
```

Continúa en la página siguiente

1 Programación básica en RAPID

1.21 Multitarea

Continuación

```
ISignalDO do1, 1, isiint1;

WHILE TRUE DO
    WaitTime 200;
ENDWHILE

IDelete isiint1;
ENDPROC

TRAP isi_trap
.
ENDTRAP
ENDMODULE
```

Comunicación entre tareas

Es posible utilizar variables persistentes para enviar cualquier tipo de datos entre dos (o más) tareas persistentes globales.

Una variable persistente global es global en todas las tareas. La variable persistente debe ser del mismo tipo y tamaño (las mismas dimensiones de matriz) en todas las tareas en las que se declare. De lo contrario, se producirán errores de tiempo de ejecución.

Ejemplo

TAREA 1:

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

    stringtosend:="this is a test";

    startsync:= TRUE

ENDPROC
ENDMODULE
```

TASK 2:

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

    WaitUntil startsync;
    !read string
    IF stringtosend = "this is a test" THEN
        ...
    ENDIF
ENDPROC
ENDMODULE
```

Continúa en la página siguiente

Tipo de tarea

Cada tarea puede ser del tipo **NORMAL**, **STATIC** o **SEMISTATIC**.

Las tareas de tipo **STATIC** y **SEMISTATIC** se inician con la secuencia de puesta en marcha del sistema. Si la tarea es de tipo **STATIC**, se reinicia en la posición actual (el punto en el que se encontrara el PP en el momento de apagar el sistema). Si se cambia el tipo a **SEMISTATIC**, se reiniciará desde el principio cada vez que se encienda la alimentación y los módulos especificados en los parámetros del sistema se recargan si el archivo de módulo es más reciente que el módulo cargado.

Las tareas de tipo **NORMAL** no se inician durante la puesta en marcha. Se inician de la forma normal, por ejemplo, desde el FlexPendant.

Prioridades

La forma predeterminada de ejecutar las tareas es ejecutar todas ellas en el mismo nivel y por turnos (un paso básico en cada instancia). Sin embargo, es posible cambiar la prioridad de una tarea, poniéndola en segundo plano de otra. En este caso, la tarea en segundo plano sólo se ejecuta cuando la tarea en primer plano está esperando determinados eventos o cuando se haya detenido su ejecución (en espera). Los programas de robot que contienen instrucciones de movimiento están en espera la mayor parte del tiempo.

El ejemplo siguiente describe algunas situaciones en las que el sistema tiene 10 tareas (consulte la *Figura 9*)

Cadena de ciclo 1: tareas 1, 2 y 9 ocupadas.

Cadena de ciclo 2: tareas 1, 4, 5, 6 y 9 ocupadas; tareas 2 y 3 en espera.

Cadena de ciclo 3: tareas 3, 5 y 6 ocupadas; tareas 1, 2, 9 y 10 en espera.

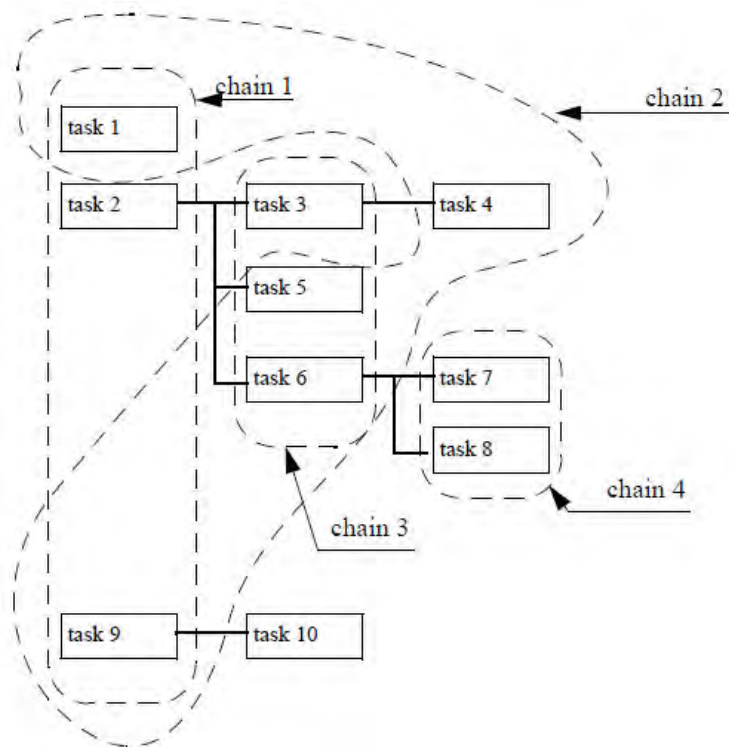
Continúa en la página siguiente

1 Programación básica en RAPID

1.21 Multitarea

Continuación

Cadena de ciclo 4: tareas 7 y 8 ocupadas; tareas 1, 2, 3, 4, 5, 6, 9 y 10 en espera.



xx1100000589

Figura 9: Las tareas pueden tener prioridades diferentes

TrustLevel

TrustLevel gestiona el comportamiento del sistema cuando una tarea de tipo **SEMISTATIC** o **STATIC** se detiene por algún motivo o cuando no es posible ejecutarla.

- **SysFail**: éste es el comportamiento predeterminado. Todas las demás tareas de tipo **NORMAL** se detienen también y el sistema pasa al estado **SYS_FAIL**. Se rechazarán todas las solicitudes de movimiento e inicio de programa. Sólo un nuevo arranque en caliente permite restablecer el sistema. Esto debe utilizarse cuando la tarea tiene determinadas supervisiones de seguridad.
- **SysHalt**: todas las tareas de tipo **NORMAL** se detendrán. El sistema pasa forzosamente al modo **Motors OFF**. Al poner el sistema en **Motors ON**, el sistema se restablece. Cuando se cambia el sistema al modo **Motors ON**, es posible mover el motor, pero se rechazan los nuevos intentos de iniciar el programa. Un nuevo arranque en caliente restablece el sistema.
- **SysStop**: todas las tareas de tipo **NORMAL** se detendrán, pero se pueden reiniciar. También se permite hacer movimientos.
- **NoSafety** : sólo se detiene la tarea actual.

Consulte *Manual de referencia técnica - Parámetros del sistema*, tema *Controller*, tipo *Task*.

Continúa en la página siguiente

Recomendación

Al especificar prioridades para las tareas, tenga en cuenta lo siguiente:

- Utilice siempre el mecanismo de interrupción o bucles con retardo en las tareas de supervisión. De lo contrario, el FlexPendant no tendrá nunca tiempo suficiente para interactuar con el usuario. Además, si la tarea de supervisión se ejecuta en primer plano, nunca permitirá la ejecución de otras tareas en segundo plano.

1.22 Ejecución hacia atrás

Descripción

Los programas pueden ejecutarse hacia atrás, una instrucción cada vez. La ejecución hacia atrás está sujeta a las restricciones generales siguientes:

- No es posible retroceder paso a paso hasta salir de una sentencia IF, FOR, WHILE y TEST.
- No es posible ejecutar paso a paso hacia atrás para salir de una rutina una vez alcanzado el principio de la rutina.
- Ni las instrucciones de parámetros de movimiento ni otras instrucciones que afectan al movimiento pueden ejecutarse hacia atrás. Si se intenta ejecutar una instrucción de este tipo, se escribe un aviso en el registro de eventos.

Gestores de ejecución hacia atrás

Los procedimientos pueden contener gestores de ejecución hacia atrás en los que se define la ejecución hacia atrás de una llamada a un procedimiento. Si se llama a una rutina que se encuentra dentro del gestor de ejecución hacia atrás, dicha rutina se ejecuta hacia delante.

El gestor de ejecución hacia atrás es en realidad parte del procedimiento, y el ámbito de los datos de cualquier rutina también incluye el gestor de ejecución hacia atrás del propio procedimiento.

Ni las instrucciones del gestor de ejecución hacia atrás ni las del gestor de errores de una rutina pueden ejecutarse hacia atrás. La ejecución hacia atrás no puede estar anidada, es decir, no es posible ejecutar simultáneamente hacia atrás dos instrucciones de una cadena de llamadas.

No es posible ejecutar hacia atrás los procedimientos que no disponen de un gestor de ejecución hacia atrás. Los procedimientos cuyos gestores de ejecución hacia atrás están vacíos se ejecutan como “sin operaciones”.

Ejemplo 1

```
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 se llama al procedimiento durante una ejecución en avance, se produce lo siguiente:

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

Ejemplo 2

```
PROC MoveTo ()
  MoveL p1,v500,z10,tool1;
```

Continúa en la página siguiente

```

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 se llama al procedimiento durante una ejecución en avance, se ejecuta el siguiente código (el código del procedimiento hasta el gestor de ejecución hacia atrás):

```

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

```

Cuando se llama al procedimiento durante una ejecución hacia atrás, se ejecuta el siguiente código (el código del gestor de ejecución hacia atrás):

```

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

```

Limitación de las instrucciones de movimiento del gestor de ejecución hacia atrás

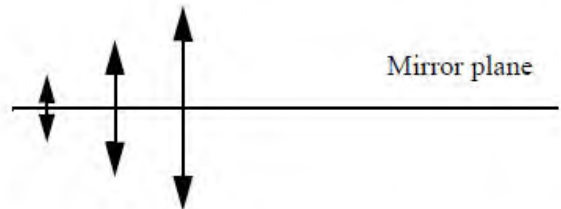
El tipo y la secuencia de instrucciones de movimiento del gestor de ejecución hacia atrás deben corresponder al espejo del tipo y la secuencia de instrucciones de la ejecución en avance de la misma rutina.

```

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

```

xx1100000633



Recuerde que el orden de CirPoint `CirPoint p2` y ToPoint `p3` de `MoveC` deben ser iguales.

Las instrucciones de movimiento afectadas son todas aquellas instrucciones que produzcan cualquier movimiento en el robot o en los ejes adicionales, como por ejemplo `MoveL`, `SearchC`, `TriggJ`, `ArcC` o `PaintL`.



¡AVISO!

Cualquier método de programación que no tenga en cuenta esta limitación en la programación del gestor de ejecución hacia atrás puede dar lugar a un movimiento de ejecución hacia atrás defectuoso. El movimiento lineal puede dar lugar a un movimiento circular y viceversa en alguna parte de la trayectoria de ejecución hacia atrás.

Continúa en la página siguiente

1 Programación básica en RAPID

1.22 Ejecución hacia atrás

Continuación

Comportamiento de la ejecución hacia atrás

MoveC y rutinas nostepin

Cuando se ejecuta paso a paso hacia delante y se pasa por una instrucción `MoveC`, el robot se detiene en el punto circular (la instrucción se ejecuta en dos pasos). Sin embargo, cuando se ejecuta paso a paso hacia atrás y se pasa por una instrucción `MoveC`, el robot no se detiene en el punto circular (la instrucción se ejecuta en un solo paso).

No se permite pasar de la ejecución en avance a la ejecución hacia atrás cuando el robot está ejecutando una instrucción `MoveC`.

No se permite pasar de la ejecución en avance a la ejecución hacia atrás, ni viceversa, en una rutina `nostepin`.

Objetivo, tipo de movimiento y velocidad

Al ejecutar paso a paso hacia delante a través del código del programa, un puntero de programa apunta a la siguiente instrucción a ejecutar y un puntero de movimiento apunta a la instrucción de movimiento que está realizando el robot en ese momento.

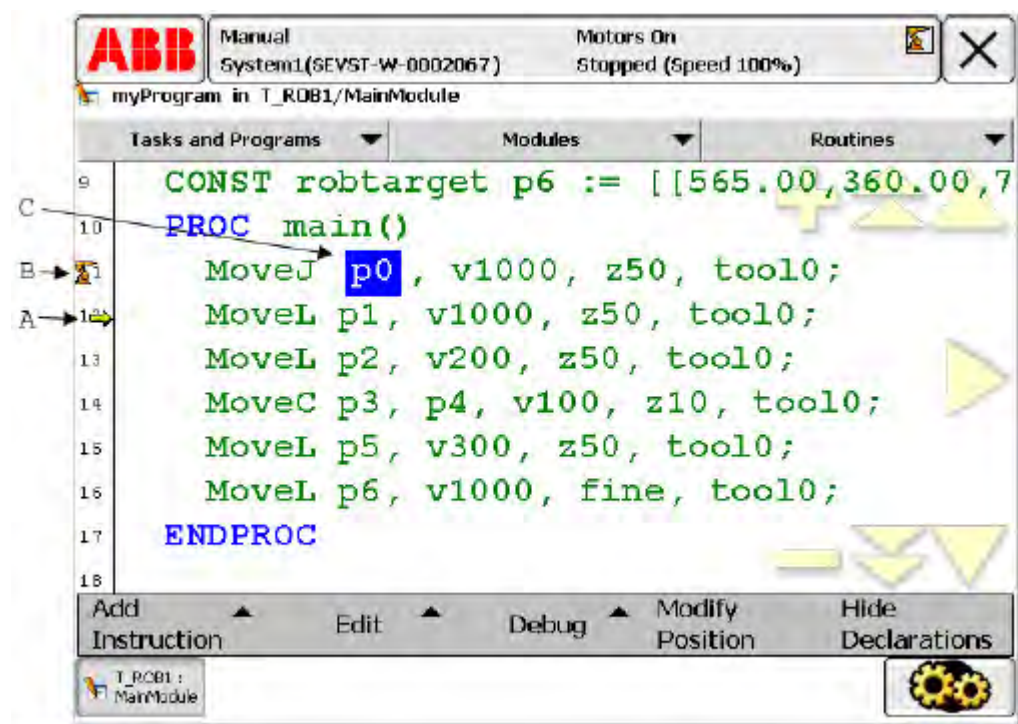
Al ejecutar paso a paso hacia atrás a través del código de programa, el puntero del programa apunta a la instrucción situada por encima del puntero de movimiento. Cuando el puntero de programa apunta a una instrucción de movimiento y el puntero de movimiento apunta a otra, el siguiente movimiento hacia atrás lleva hasta el objetivo al que apunta el puntero de programa, utilizando el tipo de movimiento y velocidad al que apunta el puntero de movimiento.

Una excepción en cuanto a la velocidad de ejecución hacia atrás es la instrucción `MoveExtJ`. Esta instrucción utiliza la velocidad relacionada con el `robtarg` tanto para la ejecución en avance como para la ejecución hacia atrás.

Continúa en la página siguiente

Ejemplo

Este ejemplo ilustra el comportamiento que se produce al ejecutar paso a paso hacia atrás a través de instrucciones de movimiento. El puntero de programa y el puntero de movimiento ayudan a controlar dónde se encuentra la ejecución de RAPID y la posición actual del robot.



xx1100000634

- A Puntero de programa
- B Puntero de movimiento
- C Se resalta el `robtargt` hacia el que se mueve el robot o el que ya ha alcanzado.

- 1 El programa se ejecuta paso a paso hacia delante hasta que el robot está en p5. El puntero de movimiento apunta a p5 y el puntero de programa apunta a la siguiente instrucción de movimiento (`MoveL p6`).
- 2 La primera vez que se presiona el botón Retroceder, el robot no se mueve, pero el puntero de programa se mueve hasta la instrucción anterior (`MoveC p3, p4`). Así se indica que ésta es la instrucción que se ejecuta la próxima vez que se presione Retroceder.
- 3 La segunda vez que se presione el botón Retroceder, el robot se mueve linealmente hasta p4 con la velocidad v300. El objetivo de este movimiento (p4) se toma de la instrucción `MoveC`. El tipo de movimiento (lineal) y la velocidad se toman de la siguiente instrucción (`MoveL p5`). El puntero de movimiento apunta a p4 y el puntero de programa se desplaza hacia arriba hasta `MoveL p2`.
- 4 La tercera vez que se presione el botón Retroceder, el robot se mueve circularmente hasta p2 pasando por p3 y con la velocidad v100. El objetivo

Continúa en la página siguiente

1 Programación básica en RAPID

1.22 Ejecución hacia atrás

Continuación

- p2 se toma de la instrucción `MoveL p2`. El tipo de movimiento (circular), el punto circular (p3) y la velocidad se toman de la instrucción `MoveC`. El puntero de movimiento apunta a p2 y el puntero de programa se desplaza hacia arriba hasta `MoveL p1`.
- 5 La cuarta vez que se presiona el botón Retroceder, el robot se mueve linealmente hasta p1 con la velocidad v200. El puntero de movimiento apunta a p1 y el puntero de programa se desplaza hacia arriba hasta `MoveJ p0`.
 - 6 La primera vez que se presiona el botón Avanzar, el robot no se mueve, pero el puntero de programa se mueve hasta la instrucción siguiente (`MoveL p2`).
 - 7 La segunda vez que se presione el botón Avanzar, el robot se moverá hasta p2 con la velocidad v200.

2 Programación de movimiento y E/S

2.1 Sistemas de coordenadas

2.1.1 Punto central de la herramienta (TCP)

Descripción

La posición del robot y sus movimientos están relacionados en todo momento con el punto central de la herramienta (TCP). Este punto suele definirse como una parte determinada de la herramienta, por ejemplo, la boquilla de una pistola de adhesivo, el centro de una pinza o el extremo de una herramienta rectificadora.

Es posible definir varios TCP (varias herramientas), pero sólo una está activa en cada momento. Cuando se registra una posición, la información registrada corresponde al TCP. Éste es también el punto que se desplaza a lo largo de una trayectoria determinada y a una velocidad especificada.

Si el robot está sosteniendo un objeto de trabajo y trabajando con una herramienta estacionaria, se utiliza un TCP estacionario. Si dicha herramienta está activa, la trayectoria programada y la velocidad son las del objeto de trabajo. Consulte [TCP estacionarios en la página 123](#).

Información relacionada

	Se describe en
Definición del sistema de coordenadas mundo	<i>Manual de referencia técnica - Parámetros del sistema</i>
Definición del sistema de coordenadas del usuario	<i>Manual del operador - IRC5 con FlexPendant</i>
Definición del sistema de coordenadas del objeto	<i>Manual del operador - IRC5 con FlexPendant</i>
Definición del sistema de coordenadas de la herramienta	<i>Manual del operador - IRC5 con FlexPendant</i>
Definición de un punto central de la herramienta	<i>Manual del operador - IRC5 con FlexPendant</i>
Definición de una base de coordenadas de desplazamiento	<i>Manual del operador - IRC5 con FlexPendant</i>
Movimiento en distintos sistemas de coordenadas	<i>Manual del operador - IRC5 con FlexPendant</i>

2 Programación de movimiento y E/S

2.1.2 Sistemas de coordenadas utilizados para determinar la posición del TCP

2.1.2 Sistemas de coordenadas utilizados para determinar la posición del TCP

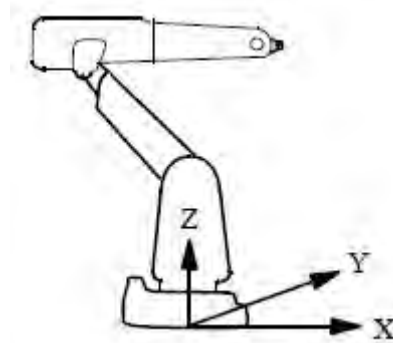
Descripción

La posición de la herramienta (TCP) puede especificarse con distintos sistemas de coordenadas para facilitar la programación y el ajuste de los programas.

El sistema de coordenadas definido depende de lo que se espera que haga el robot. Si no hay ningún sistema de coordenadas definido, las posiciones del robot se definen con el sistema de coordenadas de la base.

Sistema de coordenadas de la base

En una aplicación sencilla, la programación puede realizarse a partir del sistema de coordenadas de la base, en la que el eje z coincide con el eje 1 del robot (consulte la *Figura 10*).



xx1100000611

Figura 10: El sistema de coordenadas de la base

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

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

Sistema de coordenadas mundo

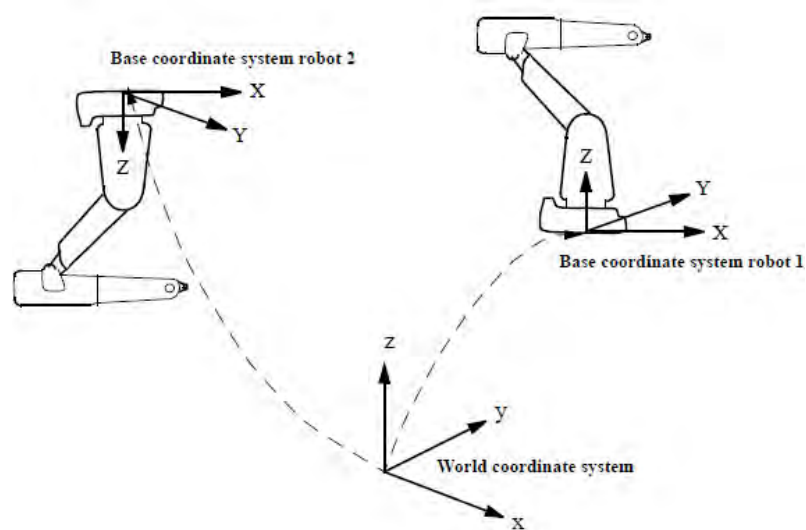
Si el robot está montado en el suelo, la programación con el sistema de coordenadas de la base resulta sencillo. Sin embargo, si está montado en posición invertida (suspendido), la programación con el sistema de coordenadas de la base resulta más difícil porque las direcciones de los ejes no son las mismas que las direcciones principales del espacio de trabajo. En estos casos, resulta útil definir un sistema de coordenadas mundo. Si no se define específicamente un sistema de coordenadas mundo, éste coincide con el sistema de coordenadas de la base.

En ocasiones, varios robots trabajan dentro de un mismo espacio de trabajo de un centro de producción. En este caso, se utiliza un mismo sistema de coordenadas mundo para que los programas de los robots puedan comunicarse unos con otros.

Continúa en la página siguiente

2.1.2 Sistemas de coordenadas utilizados para determinar la posición del TCP *Continuación*

También puede resultar ventajoso utilizar este tipo de sistemas si las posiciones parten de un punto fijo del taller. Consulte el ejemplo de la *Figura 11*.



xx1100000612

Figura 11: Dos robots (uno de los cuales está suspendido) con un mismo sistema de coordenadas mundo

Continúa en la página siguiente

2 Programación de movimiento y E/S

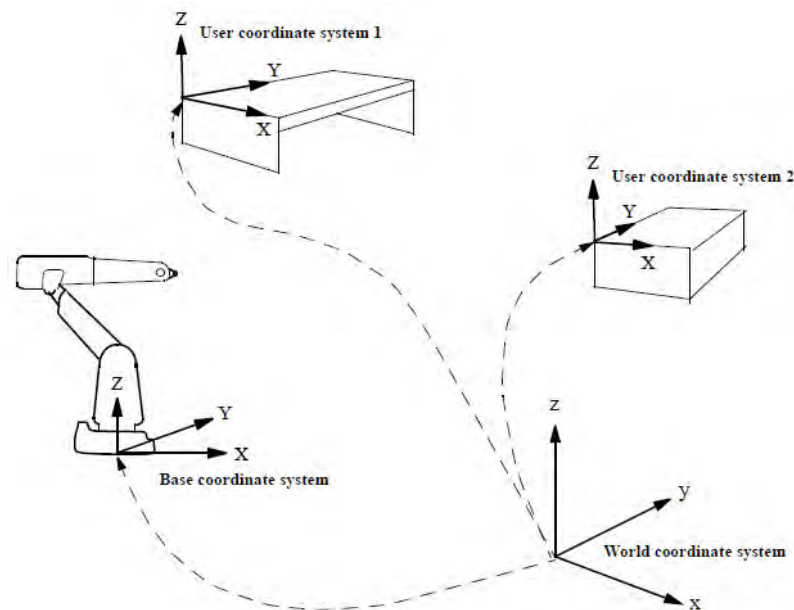
2.1.2 Sistemas de coordenadas utilizados para determinar la posición del TCP

Continuación

Sistemas de coordenadas del usuario

Un mismo robot puede trabajar con distintos útiles o superficies de trabajo que pueden tener posiciones y orientaciones diferentes. Es posible definir un sistema de coordenadas del usuario para cada útil. Si se almacenan todas las posiciones en coordenadas de objetos, deja de ser necesario reprogramar el robot si necesita mover o girar un útil. Al mover o girar el sistema de coordenadas del usuario en la misma magnitud que el útil, todas las posiciones programadas siguen al accesorio y no es necesario modificar la programación.

El sistema de coordenadas del usuario se define a partir del sistema de coordenadas mundo (consulte la *Figura 12*).



xx1100000613

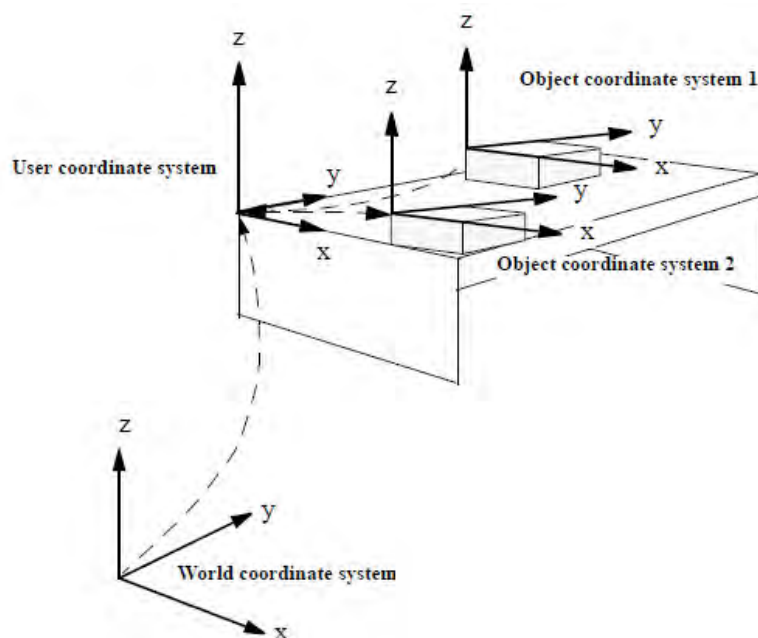
Figura 12: Dos sistemas de coordenadas de usuario describen la posición de dos útiles diferentes

Continúa en la página siguiente

Sistema de coordenadas de objeto

El sistema de coordenadas del usuario se utiliza para conseguir distintos sistemas de coordenadas para distintos útiles o superficies de trabajo. Sin embargo, cada útil puede tener varios objetos de trabajo que deben ser procesados o manipulados por el robot. Por tanto, con frecuencia resulta útil definir un sistema de coordenadas para cada objeto con el fin de facilitar el ajuste del programa si se mueve el objeto o si es necesario programar en otra posición un objeto nuevo idéntico al anterior. El sistema de coordenadas que se refiere a un objeto se conoce como sistema de coordenadas de objeto. Este sistema de coordenadas también resulta muy adecuado durante la programación fuera de línea, dado que suele ser posible tomar directamente las posiciones de un plano del objeto de trabajo. El sistema de coordenadas de objeto también puede usarse al mover el robot.

El sistema de coordenadas de objeto se define a partir del sistema de coordenadas del usuario (consulte la *Figura 13*).



xx1100000614

Figura 13: Dos sistemas de coordenadas de objeto describen la posición de dos objetos de trabajo diferentes situados en el mismo útil

Las posiciones programadas se definen en todo caso de forma relativa a un sistema de coordenadas de objeto. Si se mueve o gira un útil, este cambio puede compensarse mediante el movimiento o el giro del sistema de coordenadas del usuario. No es necesario modificar las posiciones programadas ni los sistemas de coordenadas de objetos que se hayan definido. Si se mueve o gira el objeto de trabajo, este cambio puede compensarse mediante el movimiento o el giro del sistema de coordenadas de objeto.

Si el sistema de coordenadas del usuario es móvil, es decir, que se utilizan ejes adicionales coordinados, el sistema de coordenadas de objeto se mueve junto con el sistema de coordenadas del usuario. Esto hace posible mover el robot en relación con el objeto aunque se esté manipulando el útil de trabajo.

Continúa en la página siguiente

2 Programación de movimiento y E/S

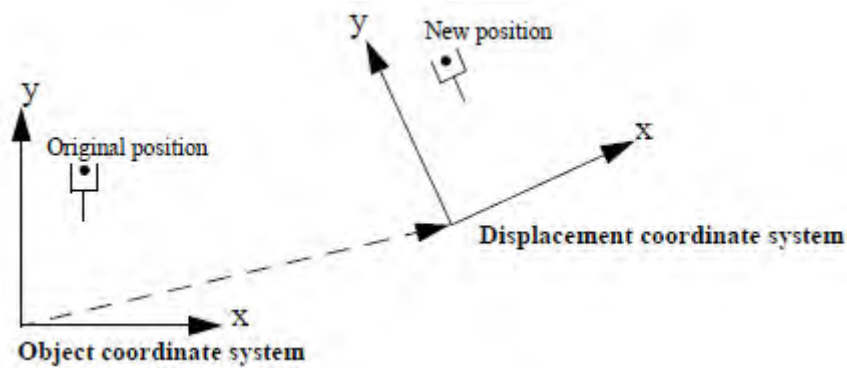
2.1.2 Sistemas de coordenadas utilizados para determinar la posición del TCP

Continuación

Sistema de coordenadas de desplazamiento

En ocasiones, es necesario realizar una misma trayectoria en varios lugares de un mismo objeto. Para evitar la reprogramación de todas las posiciones cada vez, es posible definir un sistema de coordenadas conocido como sistema de coordenadas de desplazamiento, que también puede usarse junto con las búsquedas, para compensar las diferencias existentes en las posiciones de las distintas piezas.

El sistema de coordenadas de desplazamiento se define a partir del sistema de coordenadas de objeto (consulte la *Figura 14*).



xx1100000615

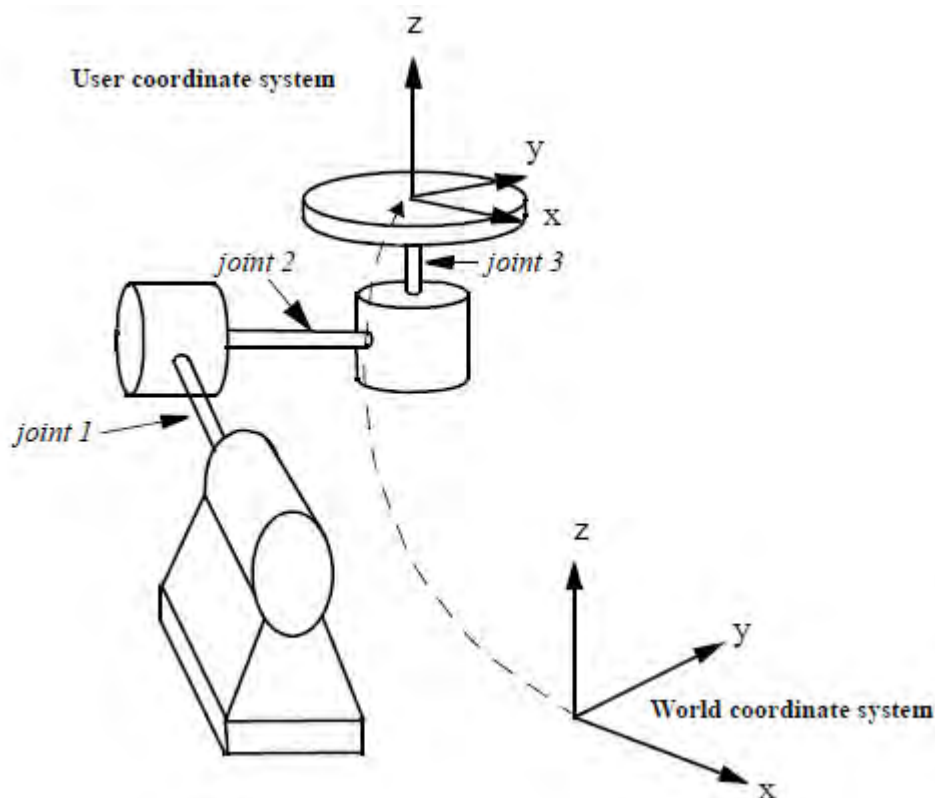
Figura 14: Si está activado el desplazamiento del programa, se desplazan todas las posiciones.

Continúa en la página siguiente

Ejes adicionales coordinados

Coordinación del sistema de coordenadas del usuario

Si se sitúa un objeto de trabajo sobre una unidad mecánica externa que se mueve mientras el robot está siguiendo la trayectoria definida con el sistema de coordenadas de objeto, es posible definir un sistema móvil de coordenadas del usuario. En este caso, la posición y la orientación del sistema de coordenadas del usuario dependerán de la rotación de los ejes de la unidad externos. Por tanto, la trayectoria y la velocidad programadas dependerán del objeto de trabajo (consulte la *Figura 15*) y no es necesario tener en cuenta el hecho de que el objeto sea movido por la unidad externa.



xx1100000616

Figura 15: Un sistema de coordenadas del usuario, definido de forma que siga los movimientos de una unidad mecánica externa con 3 ejes.

Continúa en la página siguiente

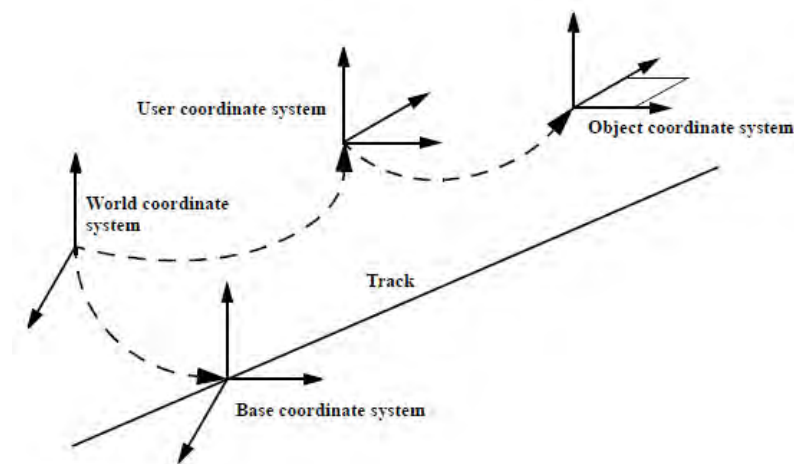
2 Programación de movimiento y E/S

2.1.2 Sistemas de coordenadas utilizados para determinar la posición del TCP

Continuación

Coordinación del sistema de coordenadas de la base

También es posible definir un sistema móvil de coordenadas para la base del robot. Esto resulta interesante en la instalación, por ejemplo, si el robot se monta sobre un track o un pórtico. Al igual que el sistema móvil de coordenadas del usuario, la posición y la orientación del sistema de coordenadas de la base dependerán de los movimientos de la unidad externa. La trayectoria y la velocidad programadas dependerán del sistema de coordenadas de objeto (*Figura 16*) y no es necesario tener en cuenta el hecho de que la base del robot es movida por una unidad externa. Es posible tener definidos simultáneamente un sistema coordinado de coordenadas del usuario y un sistema coordinado de coordenadas de la base.



xx1100000617

Figura 16: Interpolación de coordenadas con un track por el que se mueve el sistema de coordenadas de la base del robot.

Para poder calcular los sistemas de coordenadas del usuario y de la base cuando las unidades utilizadas están en movimiento, el robot debe tener en cuenta lo siguiente:

- Las posiciones de calibración de los sistemas de coordenadas del usuario y de la base
- Las relaciones existentes entre los ángulos de los ejes adicionales y la traslación/rotación de los sistemas de coordenadas del usuario y de la base.
- Estas relaciones se definen a través de los parámetros del sistema.

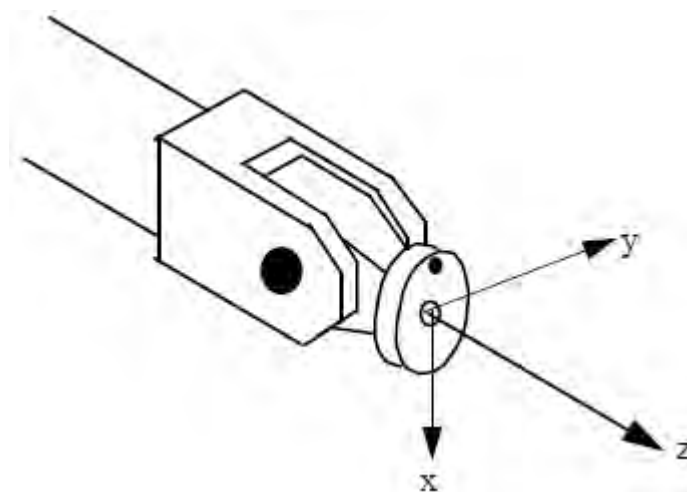
2.1.3 Sistemas de coordenadas utilizados para determinar la dirección de la herramienta

Descripción

La orientación de una herramienta en una posición programada depende de la orientación del sistema de coordenadas de la herramienta. El sistema de coordenadas de la herramienta depende del sistema de coordenadas de la muñeca, definido en la brida de montaje de la muñeca del robot.

Sistema de coordenadas de muñeca

En una aplicación sencilla, el sistema de coordenadas de la muñeca puede usarse para definir la orientación de la herramienta. En este caso, el eje z coincide con el eje 6 del robot (consulte la *Figura 17*).



xx1100000619

Figura 17: El sistema de coordenadas de la muñeca

El sistema de coordenadas de la muñeca no puede cambiarse y es siempre el mismo que el de la brida de montaje del robot en los aspectos siguientes:

- El *origen* está situado en el centro de la brida de montaje (en la superficie de montaje).
- El *eje x* apunta en el sentido opuesto, hacia el orificio de control de la brida de montaje.
- El *eje z* apunta hacia fuera, en ángulo recto respecto de la brida de montaje.

Sistema de coordenadas de la herramienta

La herramienta montada en la brida de montaje del robot requiere con frecuencia su propio sistema de coordenadas para la definición de su TCP, que es el origen del sistema de coordenadas de la herramienta. El sistema de coordenadas de la herramienta también puede usarse para obtener los sentidos de movimiento adecuados durante los desplazamientos del robot.

Continúa en la página siguiente

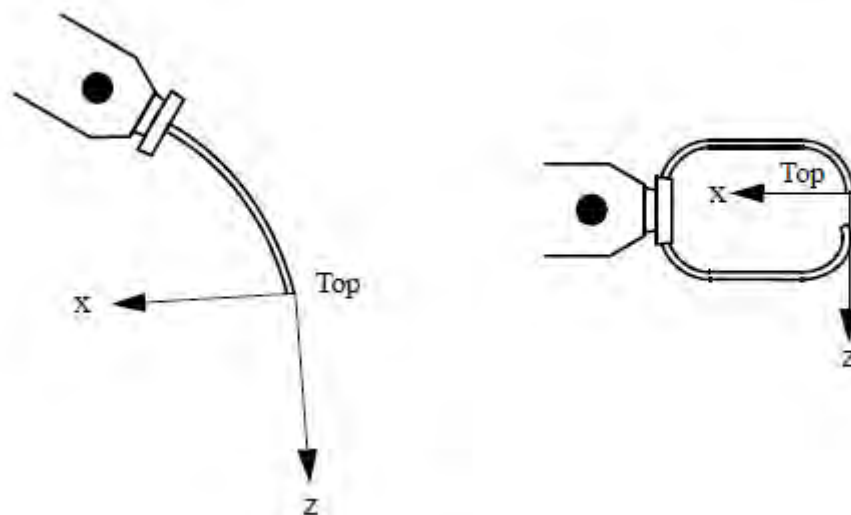
2 Programación de movimiento y E/S

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

Continuación

Si se daña o sustituye una herramienta, lo único que es necesario hacer es redefinir el sistema de coordenadas de la herramienta. Normalmente, no es necesario sustituir el programa.

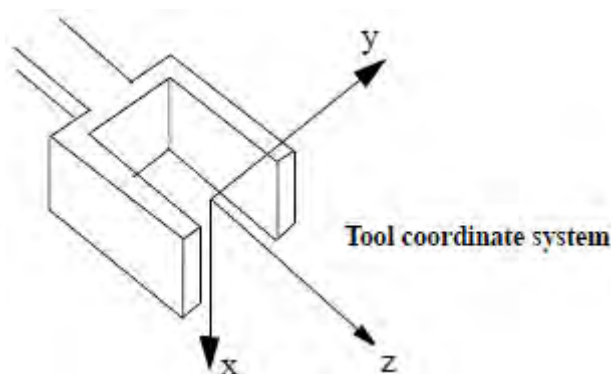
El TCP (origen) se selecciona como el punto de la herramienta que debe posicionarse correctamente, por ejemplo, la boquilla de una pistola de adhesivo. Los ejes de coordenadas de la herramienta se definen como los naturales para la herramienta correspondiente.



xx1100000618

Figura 18: Sistema de coordenadas de la herramienta definido habitualmente para una pistola de soldadura al arco (izquierda) y una pistola de soldadura por puntos (derecha).

El sistema de coordenadas de la herramienta se define a partir del sistema de coordenadas de la muñeca (consulte la Figura 19).



xx1100000642

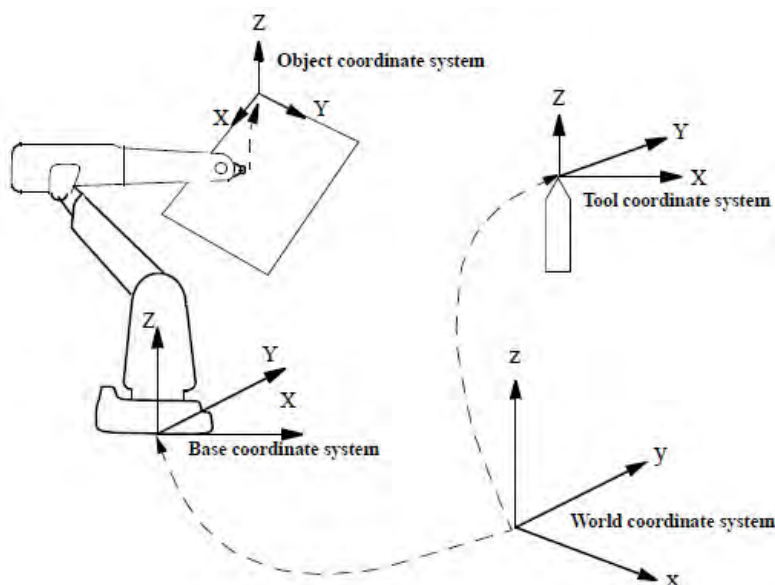
Figura 19: El sistema de coordenadas de la herramienta se define a partir del sistema de coordenadas de la muñeca. En este caso corresponde a una pinza.

Continúa en la página siguiente

TCP estacionarios

Si el robot está sosteniendo un objeto de trabajo y trabajando con una herramienta estacionaria, se utiliza un TCP estacionario. Si dicha herramienta está activa, la trayectoria programada y la velocidad son las del objeto de trabajo sostenido por el robot.

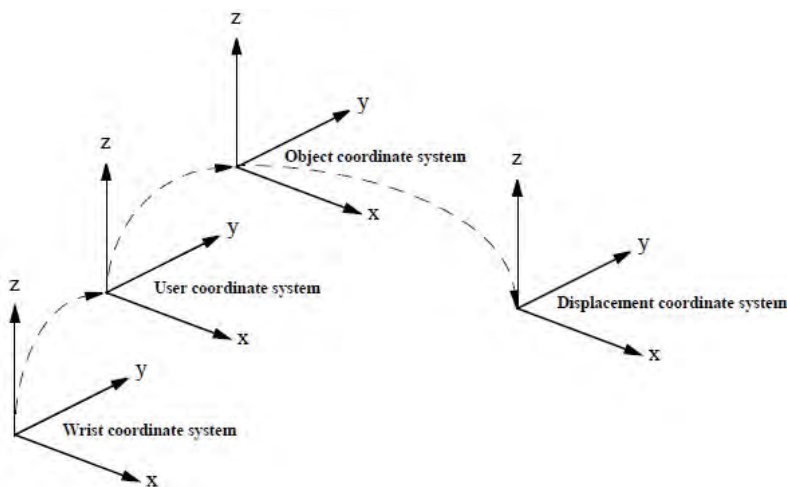
Esto significa que los sistemas de coordenadas se invierten, como en la *Figura 20*.



xx110000635

Figura 20: Si se utiliza un TCP estacionario, el sistema de coordenadas de objeto suele basarse en el sistema de coordenadas de la muñeca

En el ejemplo de la *Figura 20*, no se utiliza ni el sistema de coordenadas del usuario ni el desplazamiento de programa. Sin embargo, es posible utilizarlos y, si se usan, dependerán el uno del otro como se muestra en la *Figura 21*.



xx110000636

Figura 21: El desplazamiento del programa también puede usarse con TCP estacionarios

2 Programación de movimiento y E/S

2.2.1 Introducción

2.2 Posicionamiento durante la ejecución del programa

2.2.1 Introducción

Cómo se realizan los movimientos

Durante la ejecución del programa, las instrucciones de posicionamiento del programa del robot controlan todos los movimientos. La tarea principal de las instrucciones de posicionamiento es ofrecer la información siguiente acerca de cómo 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 adicionales).
- El método de interpolación utilizado para llegar al punto de destino, por ejemplo, la interpolación de los ejes, la interpolación lineal o la interpolación circular.
- La velocidad de los ejes del robot y de los ejes adicionales.
- Los datos de zona (definen cómo deben atravesar el punto de destino tanto el robot como los ejes adicionales).
- Los sistemas de coordenadas (herramienta, usuario y objeto) utilizados para el movimiento.

Como alternativa para la definición de la velocidad del robot y de los ejes adicionales, es posible programar el tiempo del movimiento. Sin embargo, debe evitarlo si utiliza la función de oscilación. En su lugar, debe utilizar las velocidades de los ejes de orientación y los ejes adicionales para limitar la velocidad cuando se hacen movimientos pequeños del TCP o éste no se mueve.



¡AVISO!

En aplicaciones de manipulación de materiales y palés con movimientos intensivos y frecuentes, la supervisión del sistema de accionamiento puede saltar y detener el robot para impedir el sobrecalentamiento de las unidades de accionamiento o los motores. Si esto se produce, es necesario aumentar ligeramente el tiempo de ciclo mediante la reducción de la velocidad o la aceleración programadas.

Información relacionada

	Descrito en:
Definición de velocidad	<i>Manual de referencia técnica - Instrucciones, funciones y tipos de datos de RAPID</i>
Definición de zonas (trayectorias de esquina)	<i>Manual de referencia técnica - Instrucciones, funciones y tipos de datos de RAPID</i>
Instrucción para la interpolación de ejes	<i>Manual de referencia técnica - Instrucciones, funciones y tipos de datos de RAPID</i>
Instrucción para la interpolación lineal	<i>Manual de referencia técnica - Instrucciones, funciones y tipos de datos de RAPID</i>

Continúa en la página siguiente

	Descrito en:
Instrucción para la interpolación circular	<i>Manual de referencia técnica - Instrucciones, funciones y tipos de datos de RAPID</i>
Instrucción para la interpolación modificada	<i>Manual de referencia técnica - Instrucciones, funciones y tipos de datos de RAPID</i>
Singularidad	<i>Singularidades en la página 160</i>
Ejecución simultánea de programas	<i>Sincronización con instrucciones lógicas en la página 142</i>
Optimización de la CPU	<i>Manual de referencia técnica - Parámetros del sistema</i>

2 Programación de movimiento y E/S

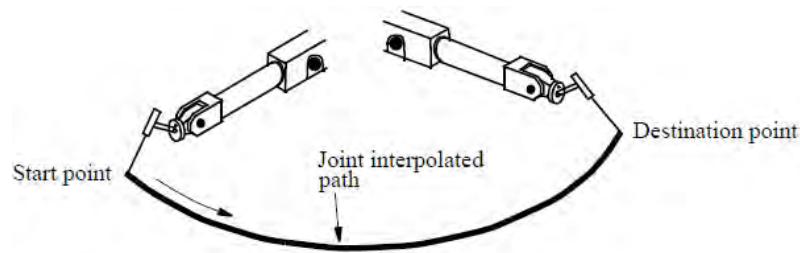
2.2.2 Interpolación de la posición y la orientación de la herramienta

2.2.2 Interpolación de la posición y la orientación de la herramienta

Interpolación de ejes

Si la exactitud de la trayectoria no es primordial, este tipo de movimiento se utiliza para mover rápidamente la herramienta de una posición a otra. La interpolación de ejes también permite el movimiento de un eje desde cualquier posición hasta otra dentro de su mismo espacio de trabajo con un solo movimiento.

Todos los ejes se mueven desde el punto de partida hasta el punto de destino con una velocidad constante de los ejes (consulte la *Figura 22*).



xx1100000637

Figura 22: La interpolación suele ser la forma más rápida de trasladarse de un punto a otro, debido a que los ejes del robot siguen la trayectoria más próxima entre el punto de inicio y el punto de destino (desde la perspectiva de los ángulos de los ejes).

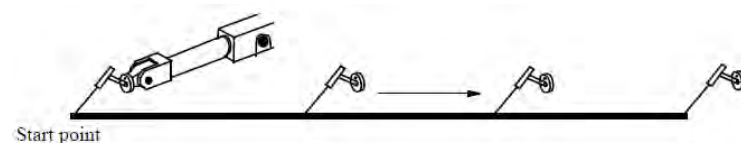
La velocidad del punto central de la herramienta se expresa en mm/s (en el sistema de coordenadas de objeto). Dado que la interpolación se realiza eje por eje, la velocidad no será exactamente el valor programado.

Durante la interpolación, se determina la velocidad del eje delimitador, es decir, el eje que se desplaza con mayor rapidez respecto de su velocidad máxima para conseguir el movimiento. A continuación, se calculan las velocidades de los demás ejes de forma que todos ellos alcancen el punto de destino al mismo tiempo.

Todos los ejes están coordinados, con el fin de conseguir una trayectoria independiente de la velocidad. La aceleración se optimiza automáticamente para conseguir el máximo rendimiento del robot.

Interpolación lineal

Durante la interpolación lineal el TCP se desplaza a lo largo de una línea recta entre los puntos de inicio y destino (consulte la *Figura 23*).



xx1100000638

Figura 23: Interpolación lineal sin reorientar la herramienta

Para obtener una trayectoria lineal en el sistema de coordenadas de objeto, los ejes del robot deben seguir una trayectoria no lineal en el espacio del eje. Cuanto menos lineal es la configuración del robot, mayores son las aceleraciones y

Continúa en la página siguiente

deceleraciones que se requieren para hacer que la herramienta se mueva en una línea recta y consiga la orientación deseada para la herramienta. Si la configuración es extremadamente no lineal (por ejemplo, cerca de singularidades de la muñeca y del brazo), uno o varios de los ejes requerirán más par del que pueden generar los motores. En este caso, se reduce automáticamente la velocidad de todos los ejes.

La orientación de la herramienta permanece constante durante todo el movimiento, a no ser que se haya programado una reorientación. Si se reorienta la herramienta, ésta gira a una velocidad constante.

Es posible especificar una velocidad máxima de rotación (en grados por segundo) durante la rotación de la herramienta. Si se utiliza en este caso un valor bajo, la reorientación resultará suave, independientemente de la velocidad definida para el punto central de la herramienta. Si se utiliza un valor elevado, la velocidad de reorientación sólo está limitada por las velocidades máximas de los motores. Siempre y cuando ninguno de los motores supere el límite de par, se mantiene la velocidad definida. Por otro lado, si uno de los motores supera el límite actual, se reduce la velocidad de todo el movimiento (tanto respecto de la posición como respecto de la orientación).

Todos los ejes están coordinados con el fin de conseguir una trayectoria independiente de la velocidad. La aceleración se optimiza automáticamente.

Interpolación circular

Las trayectorias circulares se definen mediante tres posiciones programadas que definen un segmento de círculo. El primer punto que debe programarse es el inicio del segmento de círculo. El punto siguiente es un punto de apoyo (un punto de círculo) utilizado para definir la curvatura del círculo. El tercer punto indica el final del círculo (consulte la *Figura 24*).

Los tres puntos programados deben estar repartidos en intervalos regulares a lo largo del arco del círculo, para conseguir que éste sea lo más exacto posible.

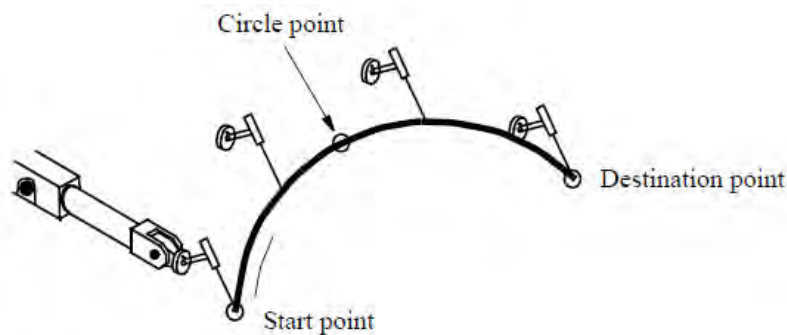
La orientación definida para el punto de apoyo se utiliza para elegir entre un giro corto y largo para la orientación, desde el punto de inicio hasta el punto de destino. Si la orientación programada es la misma respecto del círculo en los puntos de inicio y de destino y la orientación del punto de apoyo se acerca a la misma

2 Programación de movimiento y E/S

2.2.2 Interpolación de la posición y la orientación de la herramienta

Continuación

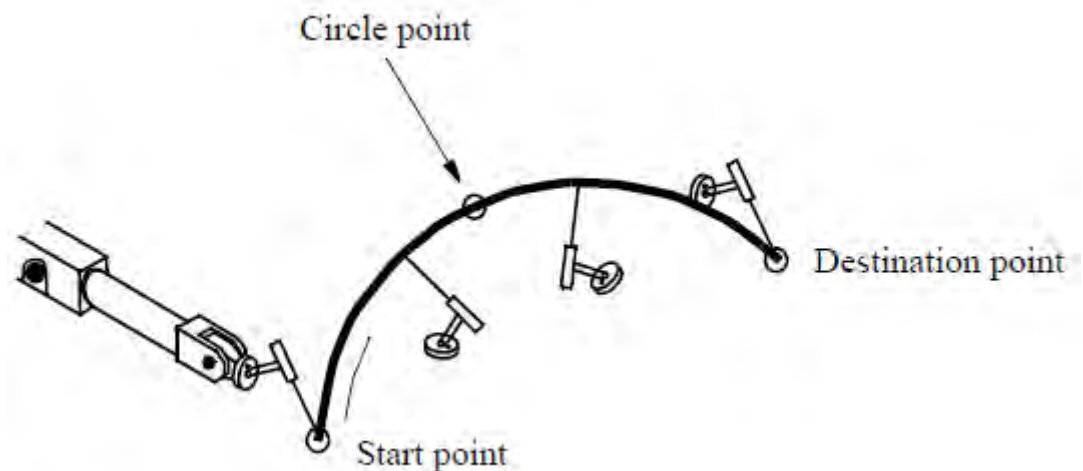
orientación respecto del círculo, la orientación de la herramienta permanece constante respecto de la trayectoria.



xx1100000639

Figura 24: Interpolación circular con un breve giro para parte de un círculo (segmento de círculo), con un punto de inicio, un punto de círculo y un punto de destino

Sin embargo, si se programa para el punto de apoyo una orientación más cercana a la orientación girada 180°, se selecciona el giro alternativo (consulte la *Figura 25*).



xx1100000640

Figura 25: La interpolación circular con un giro largo para la orientación se consigue definiendo la orientación en el punto de círculo en el sentido opuesto respecto del punto de inicio.

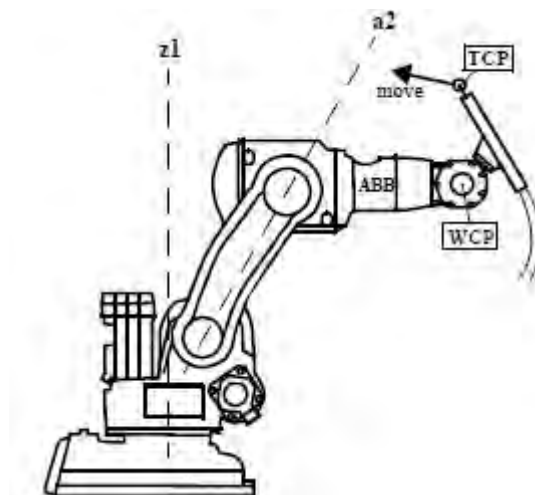
Siempre y cuando los pares de los motores no sobrepasen los valores máximos permitidos, la herramienta se mueve a la velocidad programada a lo largo del arco del círculo. Si el par de cualquiera de los motores es insuficiente, la velocidad se reduce automáticamente en las partes de la trayectoria circular en las que el rendimiento del motor no es suficiente.

Todos los ejes están coordinados con el fin de conseguir una trayectoria independiente de la velocidad. La aceleración se optimiza automáticamente.

Continúa en la página siguiente

SingArea\Wrist

Durante la ejecución en las proximidades de un punto singular, la interpolación lineal o circular puede ser problemática. En este caso, la mejor opción es utilizar la interpolación modificada, lo que significa que los ejes de la muñeca se interpolan eje por eje, con el TCP siguiendo una trayectoria lineal o circular. Sin embargo, la orientación de la herramienta será algo diferente de la orientación programada. La orientación resultante en el punto programado también puede ser distinta de la orientación programada, debido a dos singularidades.



xx1100000641

La primera singularidad se produce cuando el TCP está justo delante en línea recta del eje 2 (a2 en la figura anterior). El TCP no puede pasar al otro lado del eje 2. En su lugar, los ejes 2 y 3 se doblan un poco más para mantener el TCP en el mismo lado y en la orientación final del movimiento. A continuación, se alejan en la misma magnitud de la orientación programada.

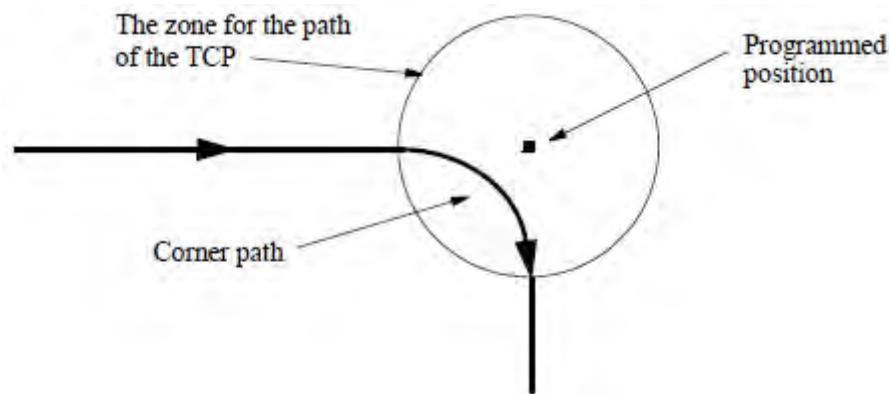
La segunda singularidad se produce cuando se prevé que el TCP pase cerca del eje z del eje 1 (z1 en la figura anterior). En este caso, el eje 1 se da la vuelta a la máxima velocidad y la reorientación de la herramienta se realiza a continuación de la misma forma. El sentido de giro depende de hacia qué lado se dirige el TCP. Se recomienda cambiar a la interpolación de ejes (MoveJ) cerca del eje z. Recuerde que el elemento que causa la singularidad es el TCP, no el WCP como cuando se usan los parámetros `SingArea\Off`.

En el caso de `SingArea\Wrist`, la orientación en el punto de apoyo del círculo será la misma que la programada. Sin embargo, la herramienta no tendrá una dirección constante respecto del plano de círculo como en el caso de la interpolación circular normal. Si la trayectoria circular atraviesa una singularidad, la orientación de las posiciones programadas requiere en ocasiones una modificación para evitar los grandes movimientos de la muñeca que pueden producirse si se genera una reconfiguración total de la muñeca cuando se ejecuta el círculo (los ejes 4 y 6 se mueven 180 grados cada uno).

2.2.3 Interpolación de trayectorias de esquina

Descripción

El punto de destino se define como un punto de paro para conseguir el movimiento de un punto a otro. Esto significa que el robot y todos los ejes adicionales se detienen y que no será posible continuar con el posicionamiento hasta que las velocidades de todos los ejes sean cero y los ejes estén cerca de sus destinos. Se utilizan puntos de paso para conseguir un movimiento continuo más allá de las posiciones programadas. De esta forma, es posible atravesar las posiciones a alta velocidad sin necesidad de reducir innecesariamente la velocidad. Los puntos de paso generan una trayectoria de esquina (trayectoria de parábola) más allá de la posición programada, lo que suele implicar que la posición programada nunca se alcanza realmente. El inicio y el final de esta trayectoria de esquina se definen mediante una zona alrededor de la posición programada (consulte la *Figura 26*).



xx1100000643

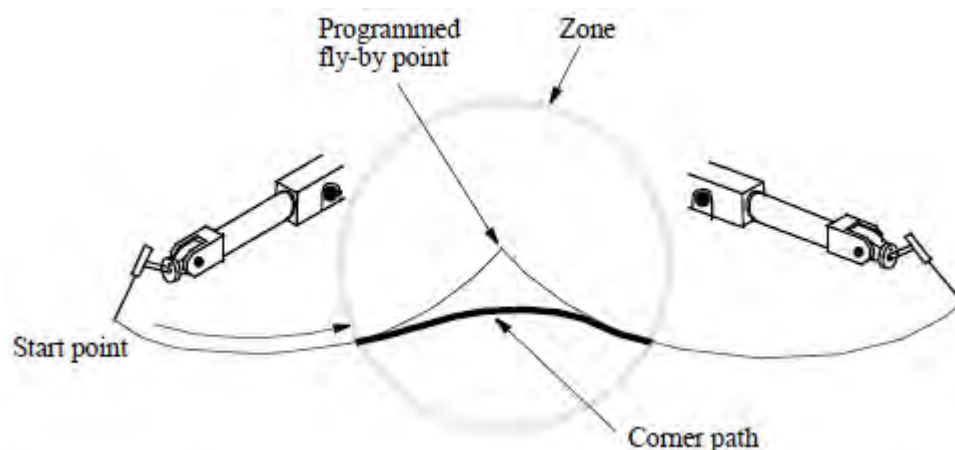
Figura 26: Los puntos de paso generan una trayectoria de esquina para pasar la posición programada.

Todos los ejes están coordinados con el fin de conseguir una trayectoria independiente de la velocidad. La aceleración se optimiza automáticamente.

Interpolación de ejes en trayectorias de esquina

El tamaño de las trayectorias de esquina (zonas) del movimiento del TCP se expresan en mm (consulte la *Figura 27*). Dado que la interpolación se realiza eje por eje, el tamaño de las zonas (en mm) debe recalcularse en ángulos de eje (radianes). Este cálculo tiene un factor de error (normalmente un 10% como máximo), lo que significa que la zona real se desviará hasta cierto punto de la programada.

Si se han programado distintas velocidades antes o después de la posición, la transición de una velocidad a otra será suave y se realizará dentro de la trayectoria de esquina sin afectar a la trayectoria real.

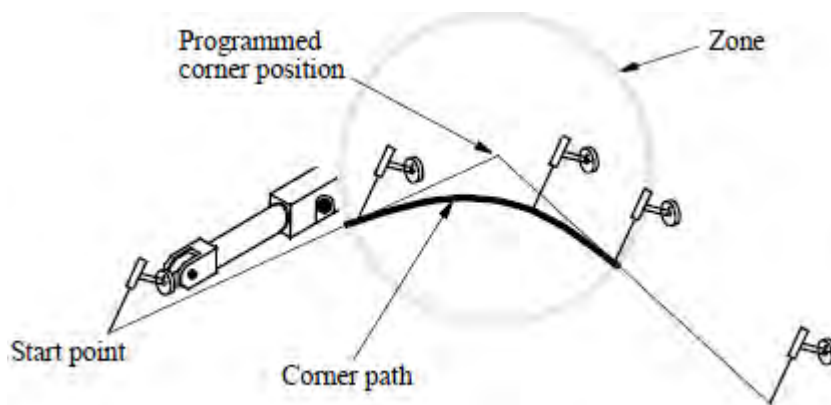


xx1100000644

Figura 27: Durante la interpolación de ejes, se genera una trayectoria de esquina para atravesar un punto de paso

Interpolación lineal de una posición en trayectorias de esquina

El tamaño de las trayectorias de esquina (zonas) del movimiento del TCP se expresan en mm (consulte la *Figura 28*).



xx1100000645

Figura 28: Durante la interpolación lineal, se genera una trayectoria de esquina para atravesar un punto de paso

Continúa en la página siguiente

2 Programación de movimiento y E/S

2.2.3 Interpolación de trayectorias de esquina

Continuación

Si se han programado velocidades distintas antes o después de la posición de la esquina, la transición será suave y se realiza dentro de la trayectoria de la esquina sin afectar a la trayectoria real.

Si la herramienta debe realizar un proceso (como soldar por arco, aplicar adhesivo o cortar con agua) a lo largo de la trayectoria de esquina, es posible ajustar el tamaño de la zona para conseguir la trayectoria deseada. Si la forma de la trayectoria parabólica de esquina no se corresponde con la geometría del objeto, es posible situar las posiciones programadas más cerca unas de otras, lo que permite aproximarse a la trayectoria deseada mediante dos o más trayectorias parabólicas más pequeñas.

Interpolación lineal de la orientación en trayectorias de esquina

Es posible definir zonas para las orientaciones de las herramientas, del mismo modo que es posible definir zonas para las posiciones de las herramientas. La zona de orientación suele definirse con un tamaño mayor que la zona de posición. En este caso, la reorientación empezará la interpolación hacia la orientación de la siguiente posición antes de que comience la trayectoria de la esquina. Por tanto, la reorientación será más suave y lo más probable es que no sea necesario reducir la velocidad para realizar la reorientación.

La herramienta se reorienta de forma que la orientación al final de la zona sea la misma que si se hubiera programado un punto de paro (consulte la *Figura 29a-c*).

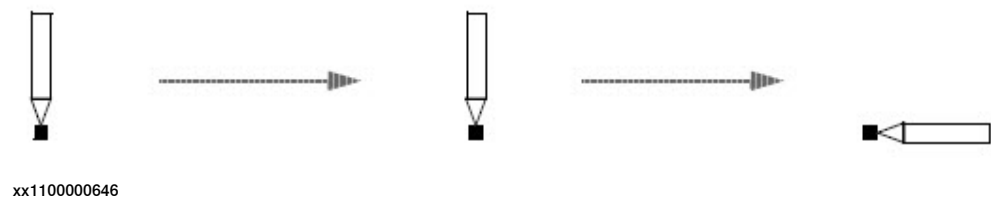


Figura 29a: Forma de programar tres posiciones con diferentes orientaciones de herramienta.

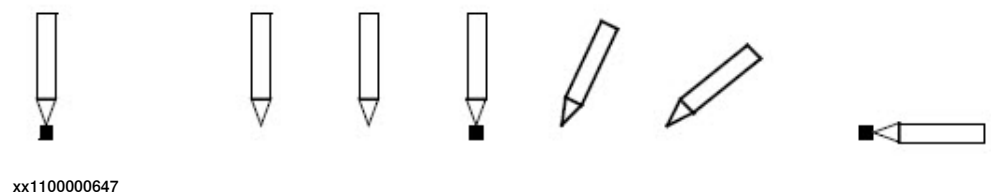
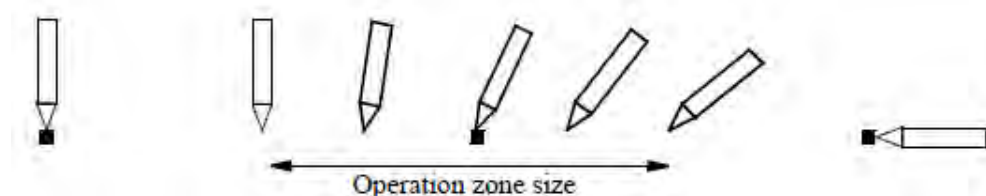


Figura 29b: Si todas las posiciones fueran puntos de paro, la ejecución del programa tendría el aspecto siguiente.

Continúa en la página siguiente



xx1100000648

Figura 29c: Si el punto intermedio fuera un punto de paso, la ejecución del programa tendría el aspecto siguiente.

La zona de orientación del movimiento de la herramienta suele expresarse en mm. De esta forma, es posible determinar directamente en qué parte de la trayectoria comienza y termina la zona de orientación. Si la herramienta no se mueve, el tamaño de la zona se expresa con grados de ángulo de rotación en lugar de mm del TCP.

Si se programan distintas velocidades de reorientación antes y después del punto de paso y si las velocidades de reorientación limitan el movimiento, la transición de una velocidad a otra se realizará suavemente dentro de la trayectoria de esquina.

Interpolación de ejes adicionales en trayectorias de esquina

También es posible definir zonas para los ejes adicionales, de la misma forma que para la orientación. Si la zona del eje adicional definida es más grande que la zona del TCP, la interpolación de los ejes adicionales hacia el destino de la siguiente posición programada comenzará antes de que comience la trayectoria de la esquina del TCP. Esto puede usarse para suavizar los movimientos de los ejes adicionales, del mismo modo que la zona de orientación se utiliza para suavizar los movimientos de la muñeca.

Continúa en la página siguiente

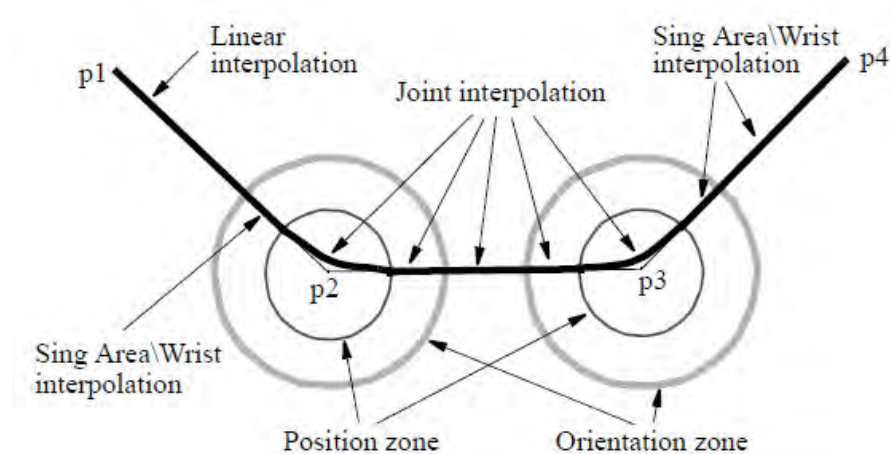
2 Programación de movimiento y E/S

2.2.3 Interpolación de trayectorias de esquina

Continuación

Trayectorias de esquina al cambiar de método de interpolación

Las trayectorias de esquina también se generan cuando se sustituye un método de interpolación por otro. El método de interpolación utilizado en las trayectorias de esquina reales se elige de tal forma que la transición de un método a otro sea lo más suave posible. Si las zonas de orientación y posición de la trayectoria de esquina no son del mismo tamaño, es posible utilizar más de un método de interpolación en la trayectoria de esquina (consulte la *Figura 30*).



xx1100000649

Figura 30: Interpolación al cambiar de un método de interpolación a otro. La interpolación lineal se ha programado entre p1 y p2; la interpolación de ejes, entre p2 y p3; y la interpolación Sing Area\Wrist, entre p3 y p4.

Si se cambia la interpolación de un movimiento normal de TCP a una reorientación sin movimiento de TCP o viceversa, no se genera ninguna zona de esquina. Lo mismo se produce si se cambia la interpolación a o de un movimiento de ejes externos sin movimiento de TCP.

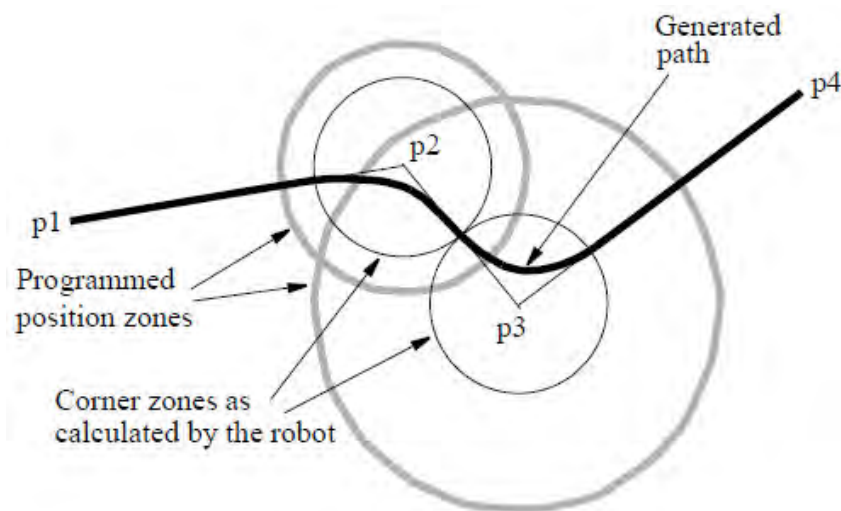
Interpolación al cambiar de sistema de coordenadas

Cuando se produce un cambio de sistema de coordenadas en una trayectoria de esquina, por ejemplo, con un nuevo TCP o un nuevo objeto de trabajo, se utiliza la interpolación de ejes de la trayectoria de esquina. Esto también se aplica al cambiar de una operación coordinada a una operación no coordinada y viceversa.

Continúa en la página siguiente

Trayectorias de esquina con zonas superpuestas

Si las posiciones programadas se encuentran muy cerca la una de la otra, no es inusual que las zonas programadas se solapen. Para obtener una trayectoria bien definida y conseguir la velocidad óptima en todo momento, el robot reduce el tamaño de la zona a la mitad de la distancia existente de una posición solapada a otra (consulte la *Figura 31*). El mismo radio de zona se utiliza siempre para las entradas y salidas desde una posición programada, con el fin de obtener trayectorias de esquina simétricas.



xx1100000650

Figura 31: Interpolación con zonas de posición superpuestas. Las zonas alrededor de p2 y p3 son más grandes que la mitad de la diferencia existente entre p2 y p3. Por tanto, el robot reduce el tamaño de las zonas para que sean iguales a la mitad de la distancia existente de p2 a p3, generando con ello trayectorias de esquina simétricas dentro de las zonas.

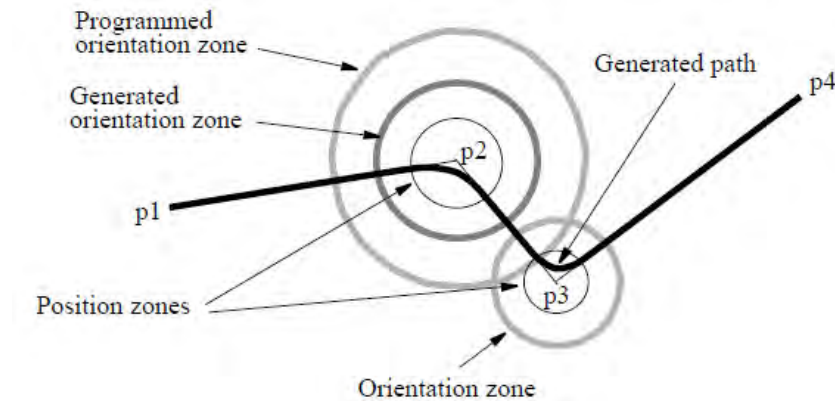
Continúa en la página siguiente

2 Programación de movimiento y E/S

2.2.3 Interpolación de trayectorias de esquina

Continuación

Tanto la posición como la orientación de las zonas de las trayectorias de esquina pueden quedar superpuestas. Tan pronto como una de estas zonas de trayectoria se superpone, la zona se reduce (consulte la *Figura 32*).



xx1100000651

Figura 32: Interpolación con zonas de orientación superpuestas. La zona de orientación de p2 es mayor que la mitad de la distancia existente entre p2 y p3 y por tanto se reduce a la mitad de la distancia existente entre p2 y p3. Las zonas de posición no se superponen y por tanto no se reducen. La zona de orientación de p3 no se reduce tampoco.

Planificación de tiempo para los puntos de paso

Ocasionalmente, si el siguiente movimiento no está planificado en el tiempo, los puntos de paso programados pueden dar lugar a un punto de paro. Esto puede ocurrir en las situaciones siguientes:

- Cuando se programa un conjunto de instrucciones lógicas con tiempos de ejecución largos entre dos movimientos cortos.
- Cuando los puntos están muy cercanos entre sí y se trabaja a velocidades elevadas.

Si los puntos de paro son un problema, utilice la ejecución simultánea de programas.

2.2.4 Ejes independientes

Descripción

Los ejes independientes son ejes que se mueven independientemente de los demás ejes del sistema de robot. Es posible cambiar un eje al modo independiente y devolver más tarde al modo normal.

Un conjunto especial de instrucciones se encarga de los ejes independientes. Cuatro instrucciones de movimiento diferentes especifican el movimiento del eje. Por ejemplo, la instrucción `IndCMove` pone en marcha el eje para un movimiento continuo. A partir de ese momento, el eje se mantiene en movimiento a una velocidad constante (independientemente de lo que haga el robot) hasta que se ejecuta una nueva instrucción de eje independiente.

Para volver al modo normal se utiliza una instrucción de restablecimiento, `IndReset`. Esta instrucción de restablecimiento también puede establecer una nueva referencia de medición (un tipo de nueva sincronización para el eje). Una vez que el eje ha vuelto al modo normal, es posible utilizarlo como un eje normal.

Ejecución de programas

Un eje cambia inmediatamente el modo independiente cuando se ejecuta una instrucción `Ind_Move`. Esto se produce incluso si se está moviendo el eje en ese momento, por ejemplo, cuando se ha programado como punto de paso un punto anterior o cuando se utiliza la ejecución simultánea de programas.

Si se ejecuta una nueva instrucción `Ind_Move` antes de que finalice la última, la nueva instrucción sustituye inmediatamente a la anterior.

Si se detiene la ejecución de un programa mientras se está moviendo un eje independiente, dicho eje se detiene. Cuando se reinicia el programa, el eje independiente se pone en movimiento inmediatamente. No se realiza ninguna coordinación activa entre los ejes independientes y los demás ejes en modo normal.

Si se produce una caída de alimentación mientras hay un eje en modo independiente, no es posible reanudar el programa. En este caso aparece un mensaje de error y es necesario reiniciar el programa desde el principio.

Recuerde que las unidades mecánicas no pueden ser desactivadas cuando uno de sus ejes está en modo independiente.

Continúa en la página siguiente

2 Programación de movimiento y E/S

2.2.4 Ejes independientes

Continuación

Ejecución paso a paso

Durante la ejecución paso a paso, un eje independiente se ejecuta sólo cuando se está ejecutando otra instrucción. El movimiento del eje también se produce paso a paso, en línea con la ejecución de otros instrumentos, como se muestra en la *Figura 33*.

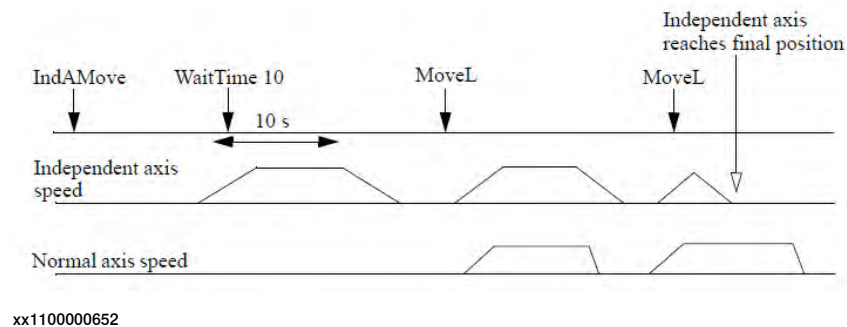


Figura 33: Ejecución paso a paso de ejes independientes.

Movimiento

Los ejes en el modo independiente no pueden tener movimientos asignados. Si se intenta ejecutar el eje manualmente, éste no se mueve y se muestra un mensaje de error. Ejecute una instrucción `IndReset` o mueva el puntero del programa a `main` para poder salir del modo independiente.

Área de trabajo

El área de trabajo físico es el movimiento total del eje.

El área de trabajo lógico es el área utilizada por las instrucciones de RAPID y que se lee en la ventana de movimientos.

Después de la sincronización (cuentarrevoluciones actualizado), las áreas de trabajo físico y lógico coinciden. La instrucción `IndReset` permite trasladar el área de trabajo lógica, como se muestra en la *Figura 34*.

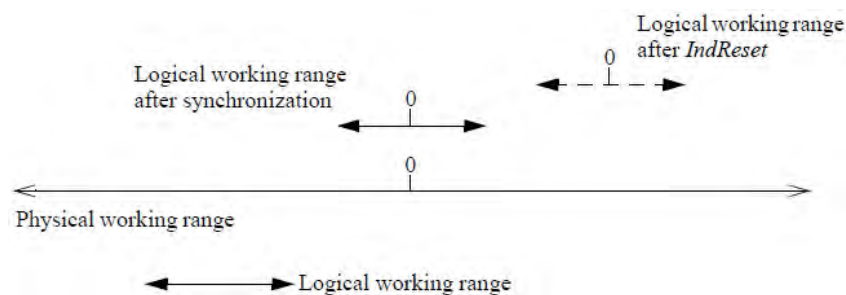


Figura 34: El área de trabajo lógico puede trasladarse mediante la instrucción `IndReset`.

La resolución de las posiciones se reduce cuando se alejan de la posición lógica 0. Una resolución baja unida a un controlador demasiado apretado puede dar lugar a un par inaceptable, a ruidos y a inestabilidad del controlador. Compruebe durante la instalación el ajuste del controlador y el rendimiento del eje cerca del límite del

Continúa en la página siguiente

área de trabajo. Compruebe también si la resolución de las posiciones y el rendimiento de la trayectoria son aceptables.

Velocidad y aceleración

En el modo manual con velocidad reducida, la velocidad se reduce al mismo nivel que si el eje estuviera funcionando de modo no independiente. Recuerde que la función `IndSpeed` no devuelve un valor `TRUE` si se reduce la velocidad del eje.

La instrucción `VelSet` y la corrección de velocidad en porcentaje a través de la ventana de producción se siguen aplicando cuando se usa el movimiento independiente. Recuerde que la corrección a través de la ventana de producción inhibe el valor `TRUE` generado por la función `IndSpeed`.

En el modo independiente, el valor más bajo de aceleración y deceleración, especificado en el archivo de configuración, se utiliza tanto para la aceleración como la deceleración. Este valor puede reducirse con un valor de pendiente en la instrucción (1 - 100%). La instrucción `AccSet` no afecta a los ejes en el modo independiente.

Ejes del robot

El único eje del robot que puede usarse como eje independiente es el eje 6. Normalmente, junto con ese eje se utiliza la instrucción `IndReset`. Sin embargo, también puede usarse la instrucción `IndReset` con el eje 4 de los modelos IRB 1600, 2600 y 4600 (no en las versiones de ID). Si se utiliza `IndReset` con el eje 4 del robot, el eje 6 no debe estar en el modo independiente.

Si se utiliza el eje 6 como eje independiente, pueden producirse problemas de singularidad porque se sigue utilizando la funcional normal de transformación de coordenadas para 6 ejes. Si aparece algún problema, ejecute el mismo programa con el eje 6 en el modo normal. Modifique los puntos o utilice las instrucciones `SingArea\Wrist` o `MoveJ`.

El eje 6 también está activo internamente en el cálculo de rendimiento de la trayectoria. Como resultado, un movimiento interno del eje 6 puede reducir la velocidad de los demás ejes del sistema.

El área de trabajo independiente del eje 6 se define con los ejes 4 y 5 en la posición inicial. Si el eje 4 ó 5 está fuera de la posición inicial, el área de trabajo del eje 6 se mueve debido al acoplamiento de los engranajes. Sin embargo, la posición leída desde el FlexPendant para el eje 6 se compensa con las posiciones de los ejes 4 y 5 a través del acoplamiento de los engranajes.

2.2.5 Servo suave

Descripción

En algunas aplicaciones se requiere un servo, que actúa como un resorte mecánico. Esto significa que la fuerza del robot sobre el objeto de trabajo aumenta como una función de la distancia existente entre la posición programada (más allá del objeto de trabajo) y la posición de contacto (entre la herramienta del robot y el objeto de trabajo).

Softness

La relación entre la desviación de la posición y la fuerza se define por un parámetro conocido como *suavidad*. Cuanto mayor es el valor del parámetro de suavidad, mayor es la desviación de posición que se requiere para conseguir la misma fuerza. El parámetro de suavidad se ajusta en el programa y es posible cambiar valores de suavidad desde cualquier parte del programa. Es posible establecer distintos valores de suavidad para los distintos ejes, además de combinar ejes que tienen un servo normal con ejes que tienen un servo suave.

La activación y desactivación del servo suave, así como la modificación de valores de suavidad, pueden hacerse con el robot en movimiento. Si se hace, se realizará un ajuste entre los distintos modos de servo y entre los distintos valores de suavidad para conseguir unas transiciones suaves. El tiempo de ajuste puede configurarse desde el programa con el parámetro “ramp”. Con $ramp = 1$, las transiciones requieren 0,5 segundos y por lo general el tiempo será $ramp \times 0.5$ en segundos.



Nota

La desactivación del servo suave no debe realizarse si existe fuerza entre el robot y el objeto de trabajo.



Nota

Con valores de suavizado elevados, existe el riesgo de que las desviaciones de posición del servo sean tan grandes que los ejes se muevan más allá del área de trabajo del robot.

2.2.6 Paro y reanudación

Detención del movimiento

Es posible detener un movimiento de tres formas diferentes:

- *Con un paro normal*, el robot se detiene sin abandonar la trayectoria, lo que hace que la reanudación del movimiento sea sencilla.
- *Con un paro rígido*, el robot se detiene en un plazo más breve que el de un paro normal, pero la trayectoria de deceleración no sigue la trayectoria programada. Por ejemplo, se utiliza este método de paro para paros de búsqueda en los que es importante detener el movimiento lo antes posible.
- *Con un paro rápido*, se utilizan los frenos mecánicos para conseguir una distancia de deceleración, que puede ser tan breve como se especifique para garantizar la seguridad. Normalmente, la desviación de la trayectoria es mayor en los paros rápidos que en los paros rígidos.

Inicio del movimiento

Después de un paro (de cualquiera de los tipos indicados), la reanudación puede hacerse siempre sobre la trayectoria interrumpida. Si el robot se ha detenido fuera de la trayectoria programada, la reanudación comenzará con el regreso a la posición de la trayectoria en la que debería haberse detenido el robot.

La reanudación después de una caída de alimentación equivale a una reanudación tras un paro rápido. Debe recordar que el robot vuelve siempre a la trayectoria antes de que se reanude el funcionamiento del programa interrumpido, incluso en los casos en que la caída de alimentación se ha producido mientras se estaba ejecutando una instrucción lógica. Durante la reanudación, todos los tiempos se cuentan desde el principio, por ejemplo, al realizar el posicionamiento por tiempo o con una interrupción con la instrucción `WaitTime`.

2 Programación de movimiento y E/S

2.3 Sincronización con instrucciones lógicas

2.3 Sincronización con instrucciones lógicas

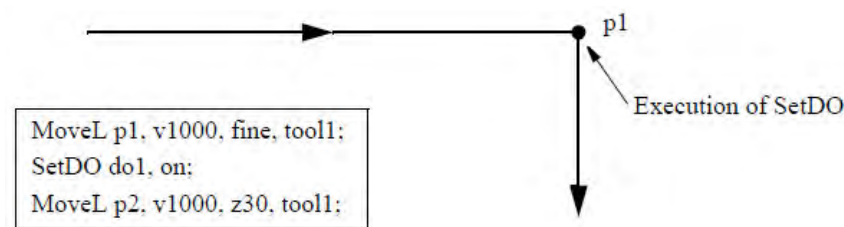
Instrucciones lógicas

Normalmente, las instrucciones se ejecutan secuencialmente en el programa. No obstante, las instrucciones lógicas también pueden ejecutarse en posiciones concretas o durante un movimiento continuo.

Una instrucción lógica es cualquier instrucción que no genere un movimiento del robot ni un movimiento de ningún eje adicional, es decir, una instrucción de E/S.

Ejecución secuencial del programa en puntos de paro

Si se ha programado una instrucción de posicionamiento como punto de paro, la instrucción posterior del programa no se ejecuta hasta que tanto el robot como los ejes adicionales se hayan parado, es decir, cuando es posible conseguir la posición programada (consulte la *Figura 35*).

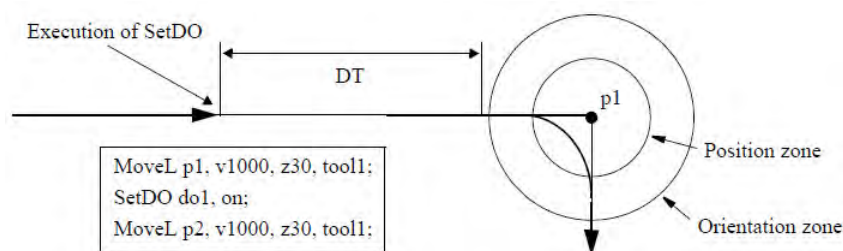


xx1100000654

Figura 35: Las instrucciones lógicas posteriores a los puntos de paro no se ejecutan hasta que se alcanza la posición de destino.

Ejecución secuencial de programas en puntos de paso

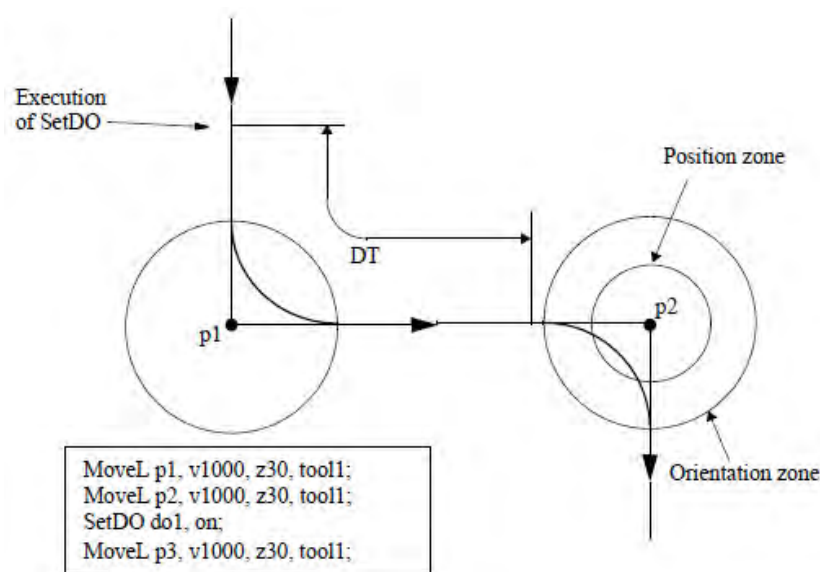
Si se programa una instrucción como punto de paso, las instrucciones lógicas siguientes se ejecutan algún tiempo antes de llegar a la zona más grande (para los ejes de posición, orientación o adicionales). Consulte la *Figura 36* y la *Figura 37*. Estas instrucciones se ejecutan en orden.



xx1100000655

Figura 36: Una instrucción lógica a continuación de un punto de paso se ejecuta antes de alcanzar la zona más grande.

Continúa en la página siguiente



xx1100000656

Figura 37: Una instrucción lógica a continuación de un punto de paso se ejecuta antes de alcanzar la zona más grande.

El tiempo en el que se ejecutan (DT) comprende los siguientes componentes de tiempo:

- El tiempo que necesita el robot para planificar el siguiente movimiento: aproximadamente 0,1 segundos.
- El retardo del robot (retardo de servo) en segundos: 0–1,0 segundos en función de la velocidad y el rendimiento real de deceleración del robot.

Ejecución simultánea de programas

La ejecución simultánea de programas puede programarse mediante el argumento `\Conc` en la instrucción de posicionamiento. Este argumento se utiliza para:

- Ejecutar una o varias instrucciones lógicas al mismo tiempo que el robot se mueve, para reducir el tiempo de ciclo (por ejemplo, se utiliza al comunicarse a través de canales serie).

Cuando se ejecuta una instrucción de posicionamiento con el argumento `\Conc`, se ejecutan también las instrucciones lógicas siguientes (en secuencia):

Si el robot no se mueve o si la instrucción de posicionamiento anterior termina con un punto de paro, las instrucciones lógicas se ejecutan tan pronto como comienza la instrucción de posicionamiento actual (al mismo tiempo que el movimiento). Consulte la *Figura 38*.

Si la instrucción de posicionamiento anterior termina en un punto de paso, las instrucciones lógicas se ejecutan en un tiempo determinado (DT) antes de alcanzar

Continúa en la página siguiente

2 Programación de movimiento y E/S

2.3 Sincronización con instrucciones lógicas

Continuación

la zona más grande (para los ejes de posicionamiento, orientación o adicionales). Consulte la *Figura 39*.

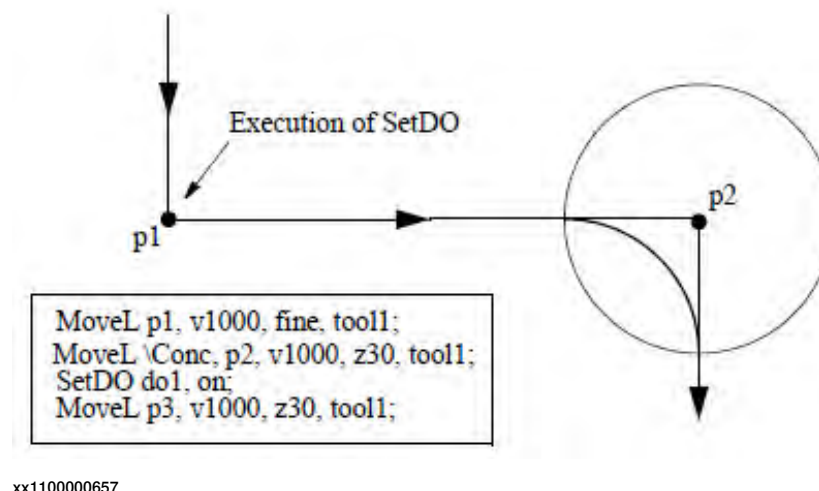


Figura 38: En el caso de la ejecución simultánea de programas después de un punto de paro, las instrucciones de posicionamiento y las instrucciones lógicas siguientes comienzan al mismo tiempo.

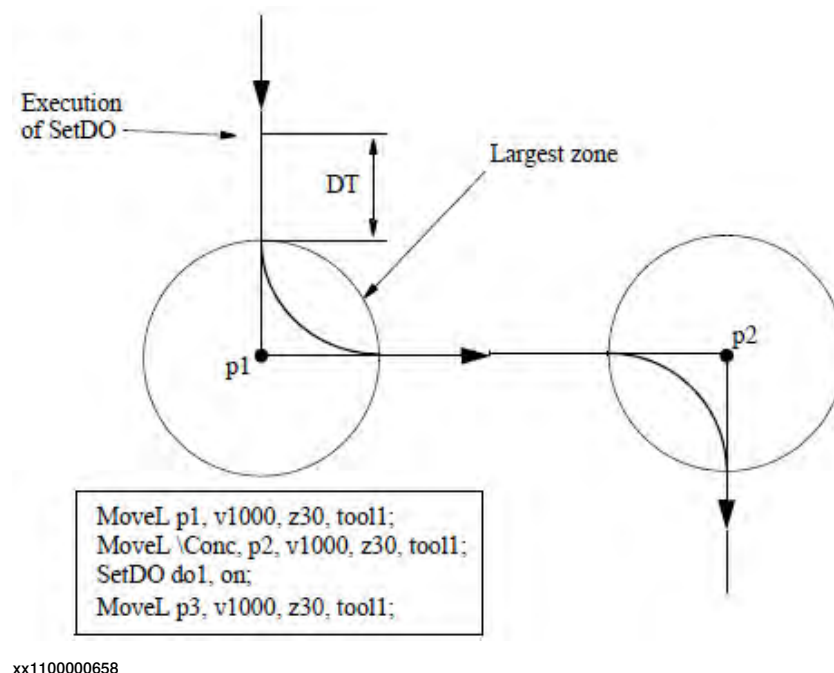


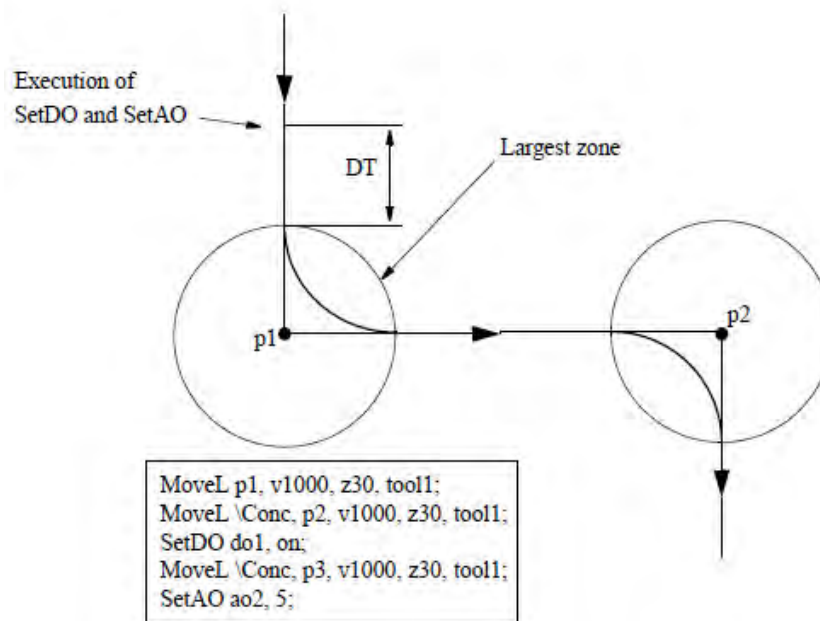
Figura 39: En el caso de la ejecución simultánea de programas después de un punto de paso, las instrucciones lógicas empiezan a ejecutarse antes de que comiencen a ejecutarse las instrucciones de posicionamiento que presentan el argumento \Conc.

Las instrucciones que afectan indirectamente a los movimientos, como `ConfL` y `SingArea`, se ejecutan de la misma forma que otras instrucciones lógicas. Sin embargo, no afectan a los movimientos solicitados por instrucciones de posicionamiento anteriores.

Continúa en la página siguiente

Si se mezclan varias instrucciones de posicionamiento que tienen el argumento `\Conc` y varias instrucciones lógicas en una secuencia larga, se aplica lo siguiente:

- Las instrucciones lógicas se ejecutan directamente en el orden en el que se programaron. Esto se realiza al mismo tiempo que el movimiento (consulte la *Figura 40*), lo que significa que las instrucciones lógicas se ejecutan en una fase anterior de la trayectoria, en lugar de en el momento en que se programaron.



xx1100000659

Figura 40: Si se programan en secuencia varias instrucciones de posicionamiento con el argumento `\Conc`, todas las instrucciones lógicas conectadas se ejecutan en el mismo momento en que se ejecuta la primera posición.

Durante la ejecución simultánea de programas, las instrucciones siguientes se programan para finalizar la secuencia y por tanto resincronizan las instrucciones de posicionamiento y las instrucciones lógicas:

- Una instrucción de posicionamiento hacia un punto de paro sin el argumento `\Conc`
- La instrucción `WaitTime` o `WaitUntil` con el argumento `\Inpos`.

Sincronización de trayectorias

Para poder sincronizar los equipos de procesamiento (para aplicaciones como aplicación de adhesivo, pintado y soldadura al arco) con los movimientos del robot, es posible generar distintos tipos de señales de sincronización.

Con los así denominados eventos de posición, se genera una señal de disparo cuando el robot atraviesa una posición predefinida de la trayectoria. Con los eventos de tiempo, se genera una señal en un tiempo predefinido antes de que el robot se detenga en la posición de paro. Además, el sistema de control también

Continúa en la página siguiente

2 Programación de movimiento y E/S

2.3 Sincronización con instrucciones lógicas

Continuación

gestiona eventos de oscilación, lo que genera pulsos en ángulos de fase predefinidos de un movimiento de oscilación.

Todas las señales de posicionamiento sincronizadas pueden conseguirse tanto antes (tiempo de avance) como después (tiempo de retardo) del momento en que el robot atraviesa la posición predefinida. La posición se define mediante una posición programada y puede ajustarse como una distancia de trayectoria antes de la posición programada.

La exactitud de repetición habitual de un conjunto de salidas digitales en la trayectoria es de +/- 2 ms.

En caso de una caída de alimentación y un reinicio con la instrucción `Trigg`, todos los eventos de disparo se generan de nuevo en la trayectoria de movimiento restante de la instrucción `Trigg`.

Información relacionada

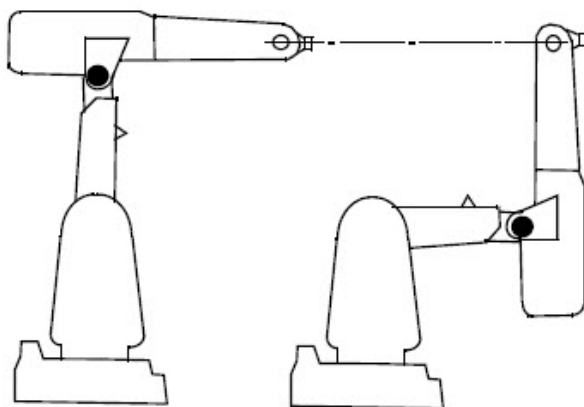
	Descrito en:
Instrucciones de posicionamiento	Movimiento en la página 57
Definición del tamaño de la zona	<i>Manual de referencia técnica - Instrucciones, funciones y tipos de datos de RAPID</i>

2.4 Configuración del robot

Distintos tipos de configuraciones de robot

Normalmente, es posible conseguir una misma posición y orientación de la herramienta de varias formas diferentes, utilizando distintos conjuntos de ángulos de ejes. Estos distintos conjuntos se denominan configuraciones de robot.

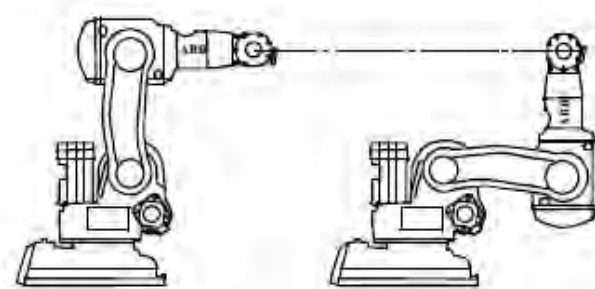
Por ejemplo, si una posición se encuentra aproximadamente en el centro de una célula de trabajo, algunos robots pueden llegar hasta la misma posición desde arriba y desde abajo, utilizando distintos sentidos del eje 1 (consulte la *Figura 41*).



xx1100000660

Figura 41: Se utilizan dos configuraciones de brazo diferentes para conseguir la misma posición y orientación. La configuración del lado derecho se consigue haciendo girar el brazo hacia atrás. El eje 1 gira 180 grados.

Algunos robots también pueden llegar a la posición desde arriba y abajo mientras utilizan el mismo sentido en el eje 1. Esto es posible con los robots que tienen un área de trabajo ampliada en el eje 3 (consulte la *Figura 42*).



xx1100000661

Figura 42: IRB 140 con dos configuraciones de brazo diferentes para conseguir la misma posición y orientación. El ángulo del eje 1 es el mismo en las dos configuraciones. La configuración del lado derecho se consigue girando el brazo inferior hacia delante y el brazo inferior hacia atrás.

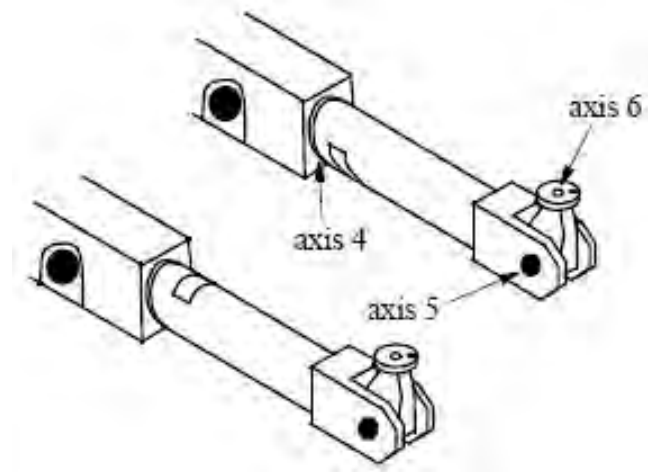
Continúa en la página siguiente

2 Programación de movimiento y E/S

2.4 Configuración del robot

Continuación

Esto también es posible haciendo girar la parte delantera del brazo superior del robot (eje 4) hacia la posición invertida a la vez que los ejes 5 y 6 giran hasta la posición y la orientación deseadas (consulte la *Figura 43*).



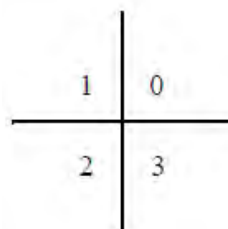
xx1100000662

Figura 43: Se utilizan dos configuraciones de muñeca diferentes para conseguir la misma posición y orientación. En la configuración en la que la parte delantera del brazo superior apunta hacia arriba (inferior), el eje 4 ha girado 180 grados, el eje 5 ha girado 180 grados y el eje 6 ha girado 180 grados para poder conseguir la configuración en la que la parte delantera del brazo superior apunta hacia abajo (superior).

Especificación de la configuración del robot

Al programar una posición del robot, se especifica también una configuración de robot con `confdata cf1`, `cf4`, `cf6`, `cfx`.

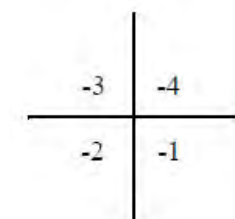
La forma de especificar la configuración del robot es distinta con los distintos tipos de robots (consulte *Manual de referencia técnica - Instrucciones, funciones y tipos de datos de RAPID - confdata* para obtener una descripción completa). Sin embargo, con la mayoría de los tipos de robots, esto incluye la definición de los cuartos de revolución adecuados para los ejes 1, 4 y 6. Por ejemplo, si el eje 1 está entre los 0 y los 90 grados, `cf1=0`. Consulte la figura siguiente.



xx1100000663

Figura 44: Cuarto de revolución para un ángulo de eje positivo

Continúa en la página siguiente



xx1100000664

Figura 45: Cuarto de revolución para un ángulo de eje negativo

Supervisión de la configuración

Normalmente, deseará que el robot tenga la misma configuración durante la ejecución del programa que la que programó. Para ello, puede usar la supervisión de la configuración del robot usando `ConfL\On` o `ConfJ\On` y, si no se consigue la configuración correcta, detener la ejecución del robot. Si la configuración no se ha supervisado, el robot puede empezar a mover inesperadamente sus brazos y muñecas, lo que podría hacer que colisione con equipos periféricos.

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

Durante un movimiento lineal, el robot se mueve siempre hacia la posición posible más cercana. Sin embargo, si la supervisión de la configuración se ha activado con `ConfL\On`, se realiza una verificación por adelantado para ver si es posible conseguir la configuración programada. Si no es posible, el programa se detiene. Cuando finaliza el movimiento (en una zona o un punto fino), también se verifica si el robot ha alcanzado la configuración programada. Si no es así, el programa se detiene. Para obtener una descripción detallada de los datos de configuración de un tipo de robot específico, consulte el tipo de dato `confdata`, *Manual de referencia técnica - Instrucciones, funciones y tipos de datos de RAPID*.

Durante el movimiento eje por eje o lineal con la supervisión de la configuración activada por `ConfL\On` o `ConfJ\On`, el robot se mueve siempre hacia la configuración de ejes programada. Si no es posible conseguir la posición y la orientación programadas, la ejecución del programa se detiene antes de empezar el movimiento. Si la supervisión de la configuración no está activada, el robot se traslada a la posición y la orientación especificadas, con la configuración más cercana.

Cuando se detiene la ejecución de una posición programada a causa de un error de configuración, con frecuencia puede deberse a uno de los motivos siguientes:

- La posición está programada fuera de línea con una configuración defectuosa.
- Se ha cambiado la herramienta del robot, haciendo que el robot tome una configuración distinta de la programada.
- La posición está sujeta a un funcionamiento con base de coordenadas activa (desplazamiento, usuario, objeto, base).
- La configuración correcta en el punto de destino puede determinarse posicionando el robot cerca de él y leyendo la configuración en el FlexPendant.

Continúa en la página siguiente

2 Programación de movimiento y E/S

2.4 Configuración del robot

Continuación

Si los parámetros de configuración cambian a causa del uso de una base de coordenadas activa, es posible desactivar la comprobación de la configuración.

Información detallada de ConfJ

MoveJ con ConfJ\Off:

- El robot se mueve hasta la posición programada, con la posición de ejes más cercana a la posición de ejes del inicio. Esto significa que no se utiliza el dato `confdata` de la instrucción. No se realiza ninguna supervisión de la configuración.

MoveJ con ConfJ\On:

- El robot se mueve hasta la posición programada, con una posición de ejes tal que la configuración correspondiente sea igual o cercana a la configuración programada en el dato `confdata`.
- Si se tiene activado un desplazamiento de programa o una corrección de trayectoria, aumenta el riesgo de error de configuración dado que los datos de configuración programados se basan en la posición original.

Información detallada de ConfL

MoveL con ConfL\Off:

- El robot se mueve en línea recta hasta la posición programada, con la posición de ejes más cercana a la posición de ejes del inicio. La configuración resultante en el punto final será la misma que la configuración real del punto de inicio. Esto significa que no se utiliza el dato `confdata` de la instrucción. No se realiza ninguna supervisión de la configuración.

MoveL con ConfL\On:

- En primer lugar se calcula la posición final con los ejes, utilizando el dato `confdata` programado para determinar la solución. A continuación, los valores de ejes para los ejes de la configuración en la posición final se comparan con los ejes correspondientes de la posición inicial.
Si los nuevos datos de configuración son aceptables en comparación con el punto inicial, el movimiento se permite. En los demás casos, el robot se detiene en la posición inicial y se emite un mensaje de error. Para obtener más detalles acerca del error de configuración permitido para los distintos tipos de robot, consulte la descripción de `confdata`, *Manual de referencia técnica - Instrucciones, funciones y tipos de datos de RAPID*.
- Si no se comunicó ningún error antes del inicio del movimiento, el sistema comprobará de nuevo la configuración al finalizar el movimiento. Si no es la misma que la configuración programada, el programa se detiene.

Información relacionada

	Descrito en:
Definición de la configuración del robot	<i>Manual de referencia técnica - Instrucciones, funciones y tipos de datos de RAPID</i>
Activación/desactivación de la supervisión de la configuración	Movimiento en la página 57

2.5 Modelos cinemáticos del robot

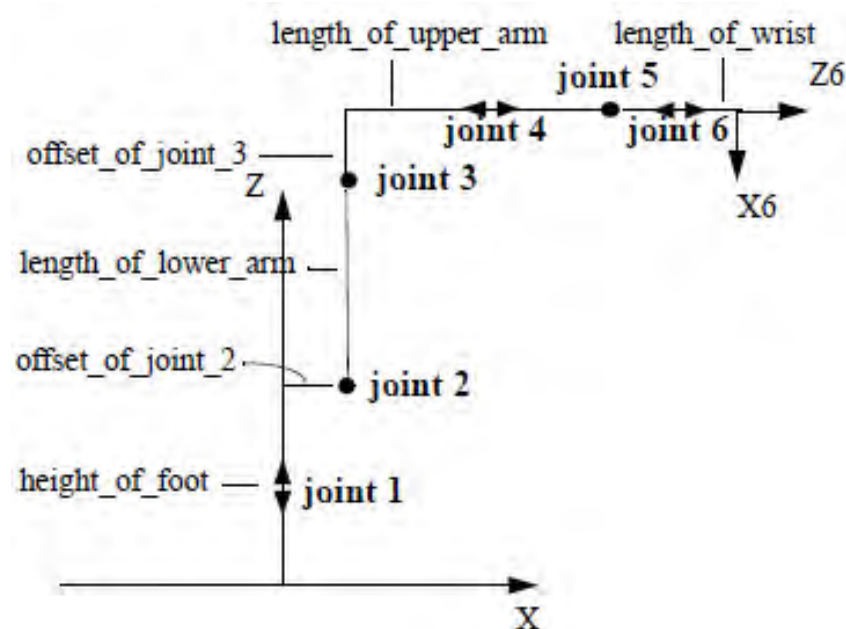
Cinemática del robot

La posición y la orientación de un robot se determinan a partir del modelo cinemático de su estructura mecánica. Es necesario definir los modelos específicos de la unidad mecánica en cada instalación. En el caso de los robots principales y exteriores ABB estándar, estos modelos están predefinidos en el controlador.

Robot principal

El modelo cinemático del robot principal modela la posición y la orientación de la herramienta del robot respecto de su base, en función de los ángulos de los ejes del robot.

Los parámetros cinemáticos que especifican las longitudes de los brazos, los offsets y las actitudes de los ejes se predefinen en el archivo de configuración de cada tipo de robot.



xx1100000666

Figura 46: Estructura cinemática de un robot IRB 1400

Continúa en la página siguiente

2 Programación de movimiento y E/S

2.5 Modelos cinemáticos del robot

Continuación

Un procedimiento de calibración apoya la definición de la base de coordenadas de la base del robot principal respecto de la base de coordenadas mundo.

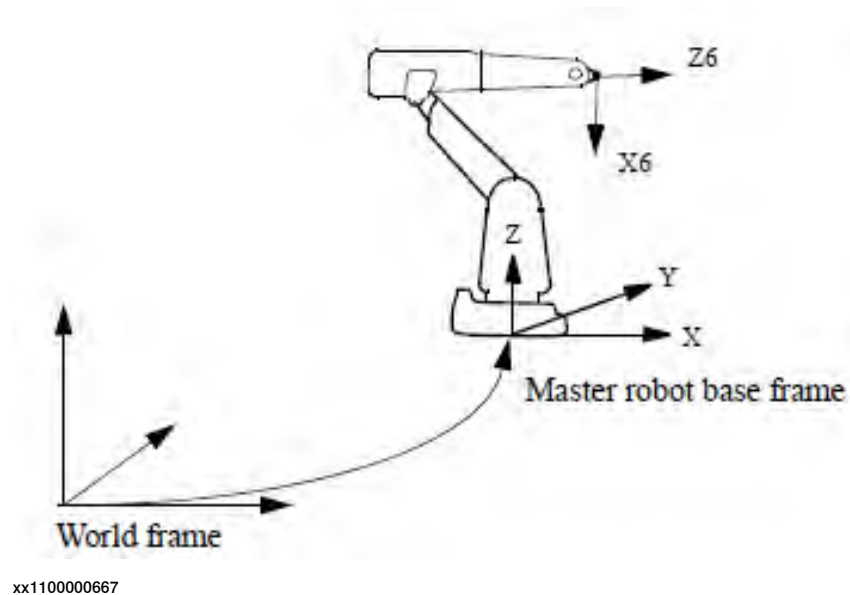


Figura 47: Base de coordenadas de la base del robot principal

Robot externo

La coordinación con un robot externo requiere también un modelo cinemático para el robot externo. Se admite un conjunto de clases predefinidas de estructuras mecánicas bidimensionales y tridimensionales.

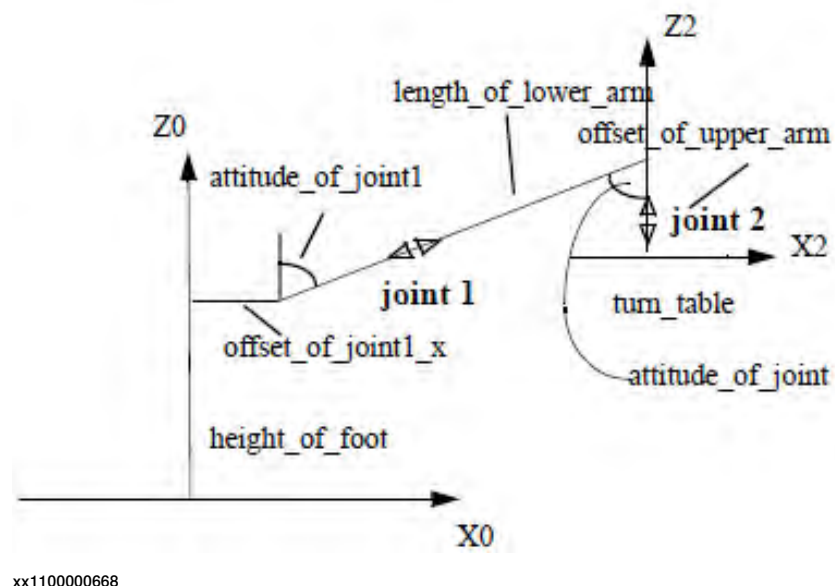


Figura 48: Estructura cinemática de un robot ORBIT 160B con un modelo predefinido

Continúa en la página siguiente

Se suministran procedimientos de calibración para definir la base de coordenadas de la base respecto de la base de coordenadas mundo para cada clase de estructura.

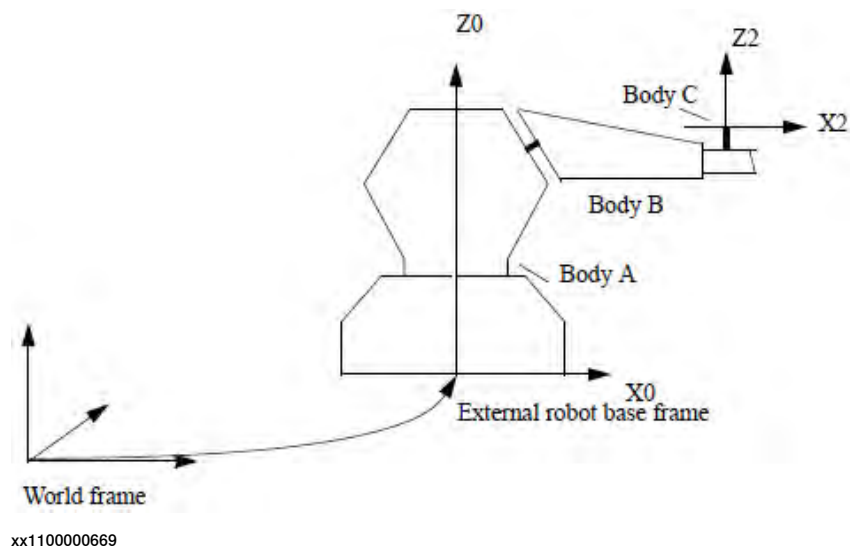


Figura 49: Base de coordenadas de la base de un robot ORBIT_160B

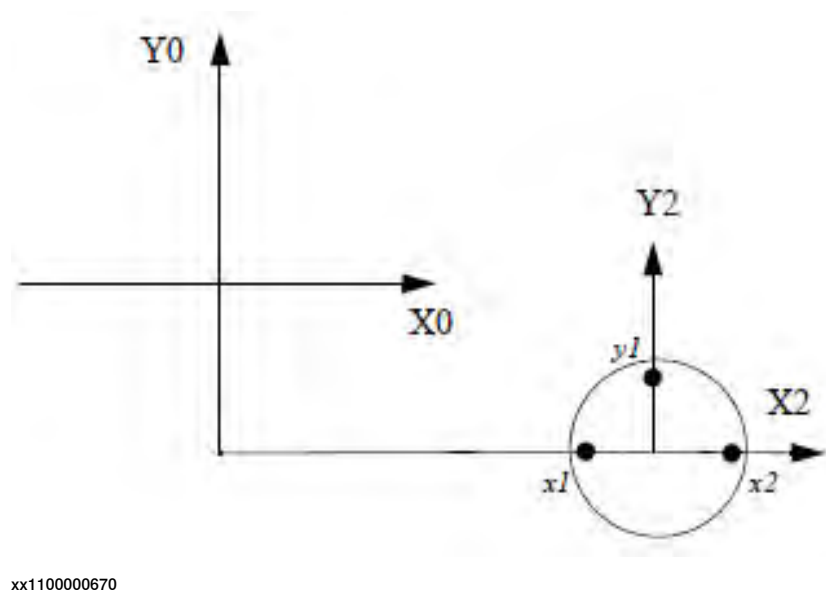


Figura 50: Puntos de referencia en la mesa giratoria para la calibración de la base de coordenadas de la base de un robot ORBIT_160B, en la posición inicial utilizando un modelo predefinido

Continúa en la página siguiente

2 Programación de movimiento y E/S

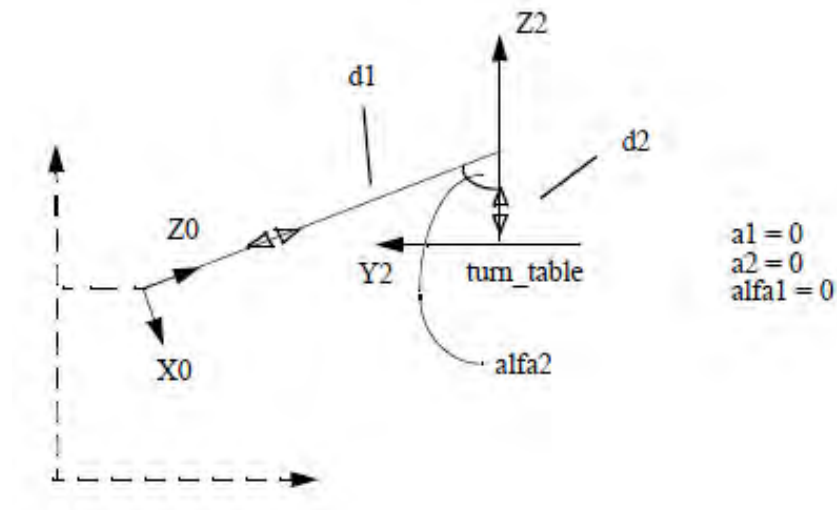
2.5 Modelos cinemáticos del robot

Continuación

Cinemática general

Las estructuras mecánicas no compatibles con las estructuras predefinidas pueden modelarse utilizando un modelo cinemático general. Esto se permite en el caso de los robots externos.

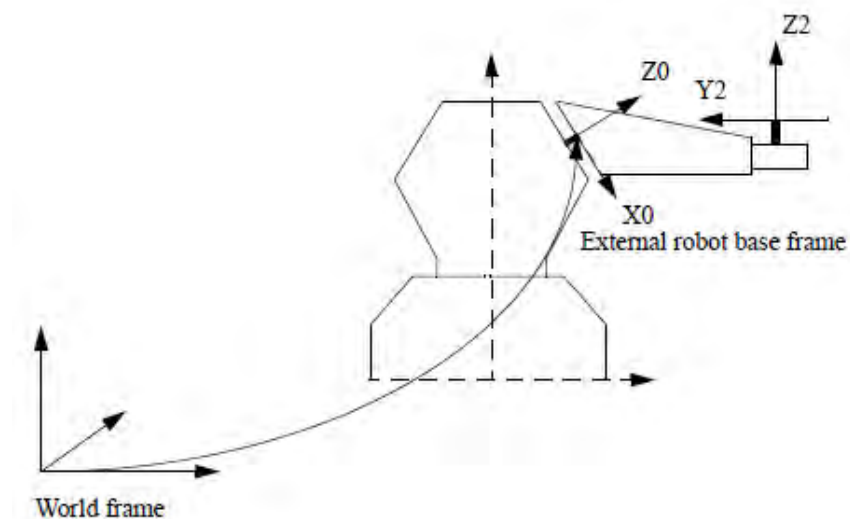
El modelado se basa en la convención *Denavit-Hartenberg* y según el libro *Introduction to Robotics, Mechanics & Control* (Introducción a la robótica, la mecánica y el control) de John J. Craig (Addison-Wesley 1986).



xx1100000671

Figura 51: Estructura cinemática de un robot ORBIT 160B con un modelo cinemático general

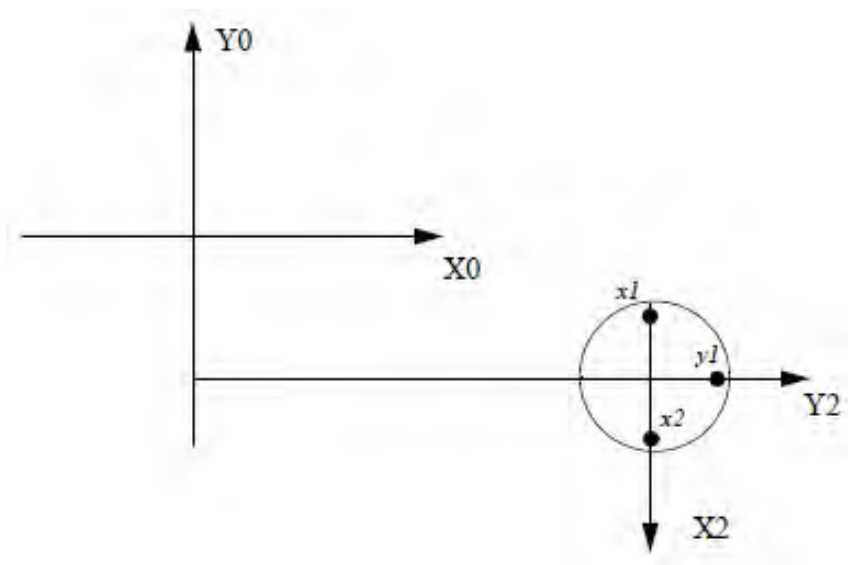
Un procedimiento de calibración apoya la definición de la base de coordenadas de la base del robot externo respecto de la base de coordenadas mundo.



xx1100000672

Figura 52: Base de coordenadas de la base de un robot ORBIT_160B con un modelo cinemático general

Continúa en la página siguiente



xx1100000673

Figura 53: Puntos de referencia en la mesa giratoria para la calibración de la base de coordenadas de la base de un robot ORBIT_160B, en la posición inicial (ejes = 0 grados)

Información relacionada

	Descrito en:
Definición de la cinemática general de un robot externo	Manual de referencia técnica - Parámetros del sistema

2.6 Supervisión del movimiento y detección de colisiones

Introducción

Se conoce como supervisión de movimiento a un conjunto de funciones que permiten supervisar los movimientos del robot con una alta sensibilidad y con bases de coordenadas. La supervisión del movimiento incluye funciones de detección de colisiones, bloqueos y cargas incorrectas. Estas funciones se conocen como funciones de detección de colisiones (opción *Collision Detection*).

La detección de colisión puede dispararse si los datos de las cargas montadas en el robot no son correctos. Esto incluye a los datos de carga de herramientas, cargas útiles y cargas de brazo. Si los datos de las herramientas o los datos de carga útil son desconocidos, puede usarse la función de identificación de cargas para definirlos. No es posible identificar los datos de carga del brazo.

Cuando se dispara la detección de colisión, se invierten los pares de motor y se aplican los frenos mecánicos para detener el robot. A continuación, el robot retrocede un poco a lo largo de la trayectoria para neutralizar las fuerzas residuales que pueden estar presentes si se ha producido una colisión o un bloqueo. A continuación, el robot se detiene de nuevo y permanece en el estado Motors ON. En la figura siguiente se ilustra una colisión típica.

La supervisión del movimiento sólo está activa si al menos un eje (incluidos los ejes adicionales) está en movimiento. Si todos los ejes están parados, la función se desactiva. Esto es así para evitar disparos innecesarios debidos a fuerzas externas del proceso.

Ajuste de los niveles de detección de colisiones

La detección de colisiones utiliza un nivel de supervisión variable. Es más sensible a velocidades bajas que a velocidades altas. Por ello, no debe ser necesario ningún ajuste de la función por parte del usuario en condiciones de funcionamiento normales. Sin embargo, es posible activar y desactivar la función para ajustar los niveles de supervisión. Existen parámetros de ajuste diferentes para el movimiento y para la ejecución del programa. Los distintos parámetros de ajuste se describen en más detalle en el *Manual de referencia técnica - Parámetros del sistema*.

RAPID cuenta con una instrucción llamada `MotionSup` que se usa para activar y desactivar la función y modificar el nivel de supervisión. Esta función resulta útil en los casos en que el proceso presenta fuerzas externas que actúan sobre el robot en determinadas partes del ciclo. La instrucción `MotionSup` se describe con más detalle en el *Manual de referencia técnica - Instrucciones, funciones y tipos de datos de RAPID*.

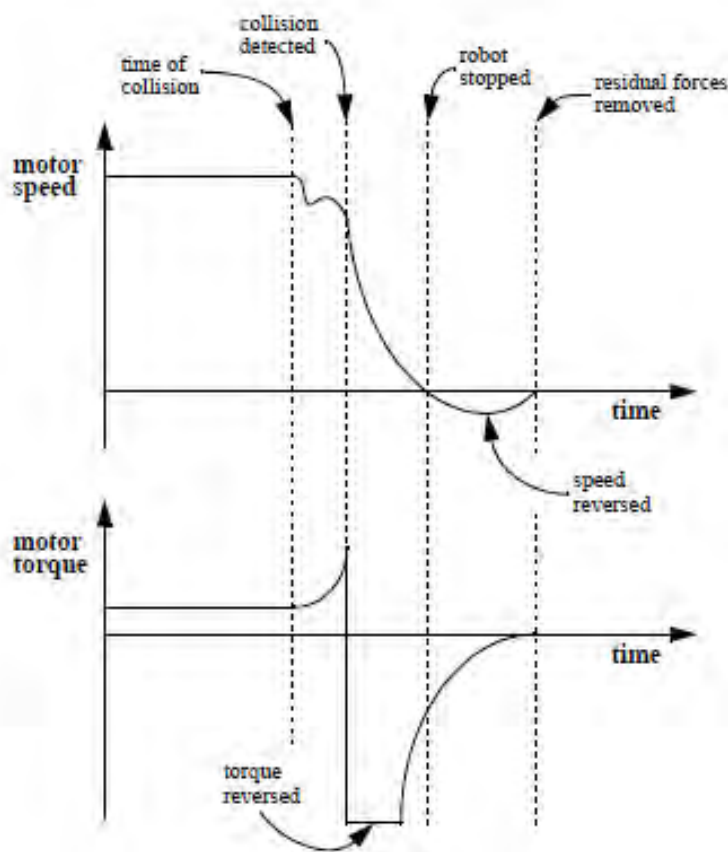
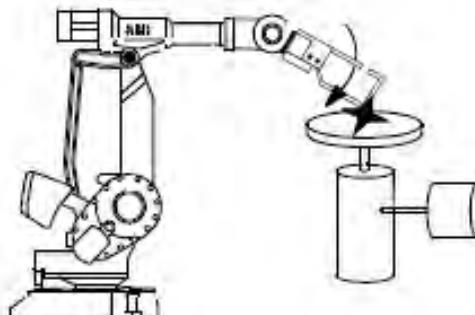
Los valores de ajuste se establecen como un porcentaje del ajuste básico. El uso del 100% supone el uso de los valores básicos. Cuando se aumenta el porcentaje, se consigue un sistema menos sensible. Al reducirlo se consigue el efecto opuesto. Es importante recordar que si se establecen valores de ajuste en los parámetros del sistema y en la instrucción de RAPID, se tienen en cuenta los dos valores. Ejemplo: Si se establece un valor de ajuste del 150% en los parámetros del sistema

y se establece un valor de ajuste del 200% en la instrucción de RAPID, el nivel de ajuste resultante será del 300%.

Figure: Typical collision

Phase 1 - The motor torque is reversed to stop the robot.

Phase 2 - The motor speed is reversed to remove residual forces on the tool and robot.



xx1100000674

Existe un nivel máximo como valor total de ajuste de la detección de colisiones que puede modificarse. Este valor es el 300% de forma predeterminada, pero puede modificarse a través del parámetro del sistema *motion_sup_max_level*.

Modificación de la supervisión del movimiento

Utilice este procedimiento en el FlexPendant para modificar la supervisión del movimiento:

- 1 En el menú **ABB**, toque **Panel de control** y a continuación **Supervisión**.

Continúa en la página siguiente

2 Programación de movimiento y E/S

2.6 Supervisión del movimiento y detección de colisiones

Continuación

- 2 Toque la lista **Tarea** y seleccione una tarea. Si tiene más de una tarea, debe configurar separadamente los valores deseados para cada tarea.
- 3 Toque **Apagado/Encendido** para eliminar o activar la supervisión de trayectoria. Toque **-/+** para ajustar la sensibilidad. La sensibilidad puede ajustarse entre 0 y 300. A no ser que tenga instalada la opción *Collision Detection*, la supervisión de trayectoria sólo afecta al robot en los modos automático y manual a máxima velocidad.
- 4 Toque **Apagado/Encendido** para eliminar o activar la supervisión de movimientos. Toque **-/+** para ajustar la sensibilidad. La sensibilidad puede ajustarse entre 0 y 300. Este valor no tiene ningún efecto a no ser que tenga instalada la opción *Collision Detection*.

Para obtener más información sobre *Collision Detection*, consulte *Application manual - Controller software IRC5*.

Salidas digitales

La salida digital `MotSupOn` está activada cuando la función de detección de colisiones está activada y desactivada cuando ésta no está activada. Recuerde que el cambio de estado de la función entra en vigor cuando comienza el movimiento. Por tanto, si la detección de colisiones está activada y el robot está en movimiento, `MotSupOn` está activada. Si se detiene el robot y se desactiva la función, `MotSupOn` sigue activada. Cuando el robot comienza a moverse, `MotSupOn` se desactiva.

La salida digital `MotSupTrigg` se activa cuando se dispara la detección de colisiones. Permanece activada hasta que se confirma el código de error, ya sea desde el *FlexPendant* o a través de la entrada digital `AckErrDialog`.

Las salidas digitales se describen con más detalle en el *Manual del operador - IRC5 con FlexPendant* y el *Manual de referencia técnica - Parámetros del sistema*.

Limitaciones

La supervisión del movimiento sólo está disponible para los ejes del robot. No está disponible para el movimiento por un track, estaciones Orbit ni otros manipuladores externos.

La detección de colisiones está desactivada cuando al menos un eje funciona en el modo de eje independiente. Éste es también el caso incluso si el eje que funciona como eje independiente es un eje adicional.

La detección de colisiones puede dispararse cuando el robot se utiliza en el modo de servo suave. Por tanto, resulta aconsejable desactivar la detección de colisiones si se utiliza el robot en el modo de servo suave.

Si se utiliza la instrucción de RAPID `MotionSup` para desactivar la detección de colisiones, el cambio sólo se aplica cuando el robot comienza a moverse. Por tanto, la salida digital `MotSupOn` puede permanecer activada temporalmente al iniciar el programa, antes de que el robot comience a moverse.

La distancia que retrocede el robot después de la colisión es proporcional a la velocidad del movimiento que se realizó antes de la colisión. Si se producen colisiones repetidas a baja velocidad, es posible que el robot no retroceda lo suficiente para anular la presión de la colisión. Como resultado, quizá no sea

Continúa en la página siguiente

posible mover el robot sin el disparo de supervisión. En este caso, utilice el menú de movimientos para desactivar temporalmente la detección de colisiones y alejar el robot del obstáculo.

En caso de una colisión rígida durante la ejecución del programa, pueden ser necesarios varios segundos antes de que el robot comience a retroceder.

Si el robot está montado sobre un track, debe desactivarse la detección de colisiones mientras se mueve la base sobre el track. Si no está desactivada, la detección de colisiones puede dispararse cuando el robot se mueve sobre el track, incluso si no se produce ninguna colisión.

Información relacionada

	Descrito en:
Instrucción de RAPID MotionSup	Movimiento en la página 57
Parámetros del sistema para ajustes	<i>Manual de referencia técnica - Parámetros del sistema</i>
Señales de E/S de supervisión del movimiento	<i>Manual de referencia técnica - Parámetros del sistema</i>
Identificación de carga	<i>Manual del operador - IRC5 con FlexPendant</i>

2.7 Singularidades

Descripción

Algunas posiciones del espacio de trabajo del robot pueden alcanzarse mediante un número infinito de configuraciones de robot en cuanto al posicionamiento y la orientación de la herramienta. Estas posiciones, conocidas como puntos singulares (singularidades), constituyen un problema a la hora de calcular los ángulos del brazo del robot a partir de la posición y la orientación de la herramienta.

En general, los robots presentan dos tipos de singularidades:

- Singularidades de brazo
- Singularidades de muñeca

Las singularidades de brazo son todas las configuraciones en las que el centro de la muñeca (la intersección de los ejes 4, 5 y 6) termina directamente sobre el eje 1 (consulte la *Figura 54*). Las singularidades de muñeca son configuraciones en las que los ejes 4 y 6 quedan en la misma línea, es decir, que el eje 5 tiene un ángulo igual a 0 (consulte la *Figura 55*).

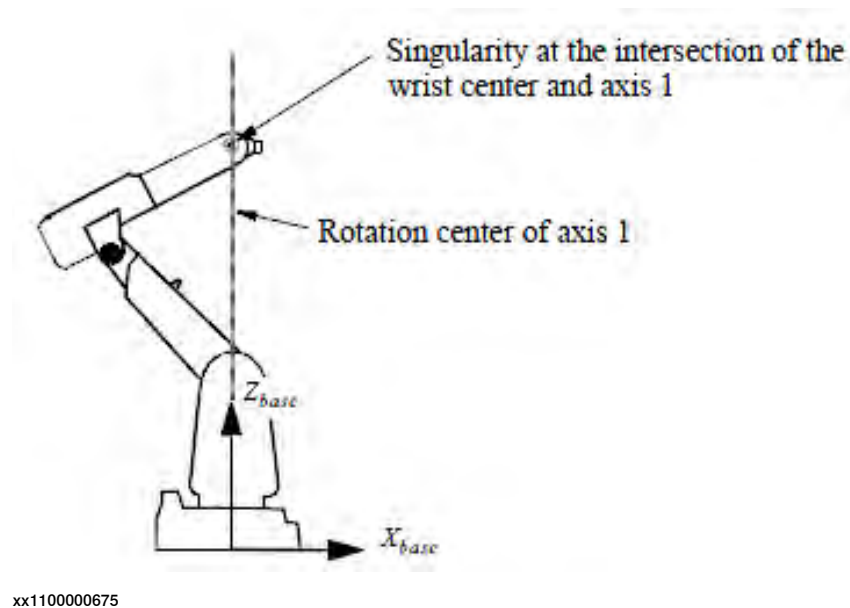
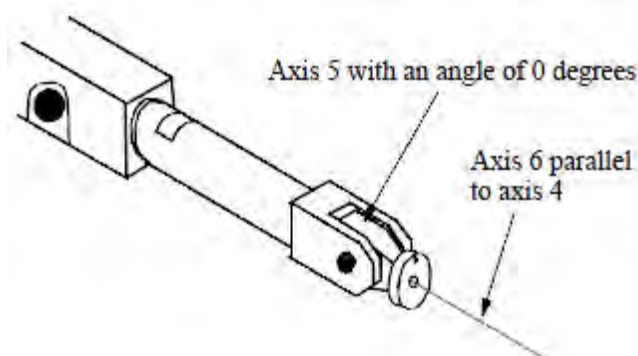


Figura 54: La singularidad del brazo se produce cuando el centro de la muñeca y el eje 1 se cortan

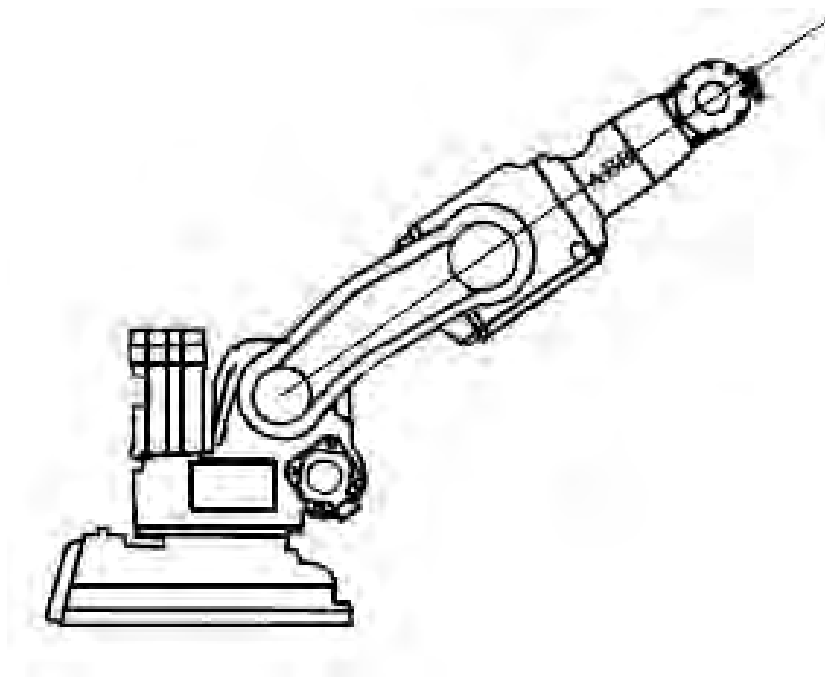


xx1100000676

Figura 55: La singularidad de la muñeca se produce cuando el eje 5 se encuentra a 0 grados

Puntos de singularidad de los robots sin barra paralela

Los robots sin barra paralela (robots con eslabones en serie) tienen la singularidad de muñeca y la singularidad de brazo, al igual que los robots de barra paralela. Además tienen un tercer tipo de singularidad. Esta singularidad se produce en las posiciones de robot en las que el centro de la muñeca y los centros de rotación de los ejes 2 y 3 se encuentran en línea recta (consulte la figura siguiente).



xx1100000677

Figura 56: El punto singular adicional del robot IRB 140

Ejecución del programa a través de singularidades

Durante la interpolación de ejes, no se produce ningún problema cuando el robot atraviesa los puntos singulares.

Continúa en la página siguiente

2 Programación de movimiento y E/S

2.7 Singularidades

Continuación

Al ejecutar una trayectoria lineal o circular cerca de una singularidad, las velocidades de algunos ejes (1 y 6/4 y 6) pueden ser muy elevadas. Para no superar las velocidades máximas de los ejes, se reduce la velocidad de la trayectoria lineal.

Es posible reducir las altas velocidades de los ejes mediante el modo (SingArea\Wrist) cuando los ejes de la muñeca están interpolados en ángulos de eje, a la vez que se mantiene la trayectoria lineal de la herramienta del robot. Sin embargo, se introduce un error de orientación en comparación con una interpolación lineal completa.

Recuerde que la configuración del robot cambia drásticamente cuando éste pasa cerca de una singularidad en una interpolación lineal o circular. Para evitar la reconfiguración, la primera posición del otro lado de la singularidad debe programarse con una orientación que haga que la reconfiguración resulte innecesaria.

Además, debe recordar que el robot no debe encontrarse en esta singularidad cuando sólo se mueven los ejes externos. Si ocurre, es posible que los ejes del robot hagan movimientos innecesarios.

Movimiento a través de singularidades

Durante la interpolación de ejes, no se produce ningún problema cuando el robot atraviesa los puntos singulares.

Durante la interpolación lineal, el robot puede atravesar puntos singulares, pero con una velocidad reducida.

Información relacionada

	Descrito en:
Control de cómo debe actuar el robot en la ejecución cerca de puntos singulares	<i>Manual de referencia técnica - Instrucciones, funciones y tipos de datos de RAPID</i>

2.8 Limitación optimizada de la aceleración

Descripción

La aceleración y la velocidad del robot se controlan continuamente de forma que no se superen los límites definidos.

Estos límites están definidos por el programa del usuario (por ejemplo, velocidad programada o `AccSet`) o definidos por el propio sistema (por ejemplo, par máximo de la caja de engranajes o del motor, par máximo o fuerza de la estructura del robot).

Datos de carga

Siempre y cuando los datos de carga (masa, centro de gravedad e inercia) se encuentren dentro de los límites del diagrama de carga y la introducción de los datos de las herramientas se realice correctamente, no se requiere ningún límite de aceleración definido por el usuario y la vida útil del robot se garantiza automáticamente.

Si los datos de carga están fuera de los límites del diagrama de carga, es posible que se requieran restricciones especiales, es decir, un valor de `AccSet` o una velocidad menor, si ABB especifica que es necesario.

Aceleración del TCP

La aceleración y la velocidad del TCP se controlan mediante el planificador de trayectorias con la ayuda de un modelo dinámico completo de los brazos del robot, incluidas las cargas definidas por el usuario.

La aceleración y la velocidad del TCP dependen de la posición, la velocidad y la aceleración de todos los ejes en cualquier instante y por tanto la aceleración real varía continuamente. De esta forma, se obtiene el tiempo de ciclo óptimo, es decir, que uno o varios de los límites se encuentran en su valor máximo en todo momento. Esto significa que los motores y la estructura del robot se utilizan a su máxima capacidad en todo momento.

2.9 Zonas mundo

Descripción de las zonas mundo

Cuando se utilizan zonas mundo (opción *World Zones*), el robot se detiene o se activa automáticamente una salida si el robot se encuentra dentro de un área especial definida por el usuario. A continuación aparecen algunos ejemplos de aplicaciones:

- Cuando dos robots comparten una parte de sus áreas de trabajo respectivas. La posibilidad de que dos robots colisionen puede eliminarse con seguridad mediante la supervisión de estas señales.
- Cuando hay equipos externos situados dentro del área de trabajo del robot. Es posible crear un área de trabajo prohibida para impedir que el robot colisione con este equipo.
- Indicación de que el robot se encuentra en una posición en la que se permite iniciar la ejecución del programa desde un PCL.



¡AVISO!

Por motivos de seguridad, esta función de software no debe utilizarse para la protección del personal. Para tal fin, utilice equipos de protección físicos.

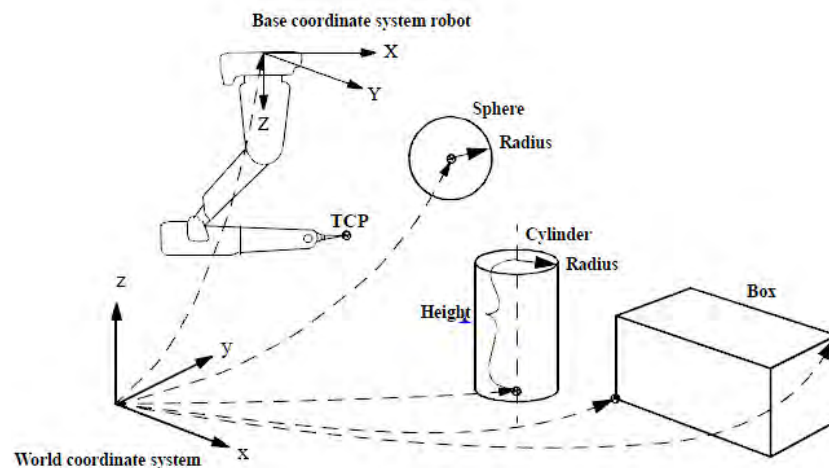
Utilización de zonas mundo

Utilice zonas mundo:

- Para indicar que el punto central de la herramienta se encuentra en una parte concreta del área de trabajo.
- Para limitar el área de trabajo del robot para evitar colisiones con la herramienta.
- Para crear un área de trabajo común para dos robots, disponibles sólo para un robot cada vez.

Definición de zonas mundo en el sistema de coordenadas mundo

Las zonas mundo deben definirse dentro del sistema de coordenadas mundo. Los lados de los cuadros son paralelos a los ejes de coordenadas y el eje de cilindro es paralelo al eje z del sistema de coordenadas mundo.

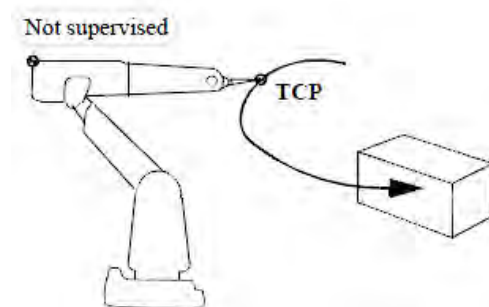


xx1100000678

Es posible definir una zona mundo de forma que esté dentro o fuera de la forma del contorno del cuadro, la esfera o el cilindro.

La zona mundo también puede definirse en el caso de los ejes. La zona debe definirse entre (dentro) o no entre (fuera) dos valores de eje para cualquier eje de robot o adicional.

Supervisión del TCP del robot



xx1100000679

Se supervisa el movimiento del punto central de la herramienta, no ninguna otra parte del robot.

El TCP se supervisa siempre con independencia del modo de funcionamiento, por ejemplo, la ejecución de programas y el desplazamiento.

Continúa en la página siguiente

2 Programación de movimiento y E/S

2.9 Zonas mundo

Continuación

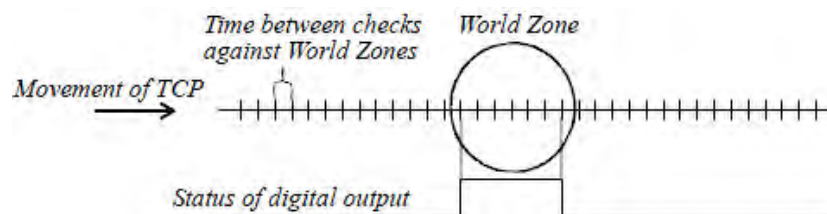
TCP estacionarios

Si el robot está sosteniendo un objeto de trabajo y trabajando con una herramienta estacionaria, se utiliza un TCP estacionario. Si la herramienta está activada, no se mueve y, si se encuentra dentro de una zona mundo, permanecerá dentro de dicha zona en todo momento.

Acciones

Activar una salida digital cuando el TCP se encuentra dentro de una zona mundo

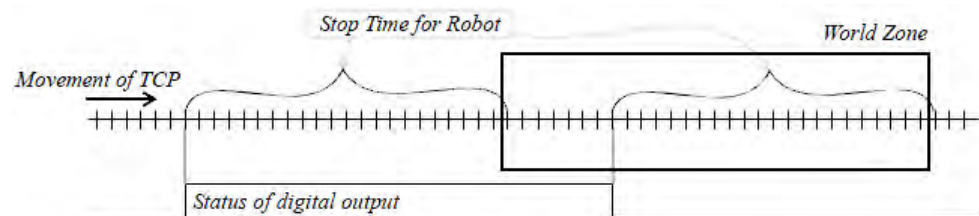
Esta acción activa una señal digital cuando el TCP se encuentra dentro de una zona mundo. Resulta útil a la hora de indicar que el robot se ha detenido dentro un área específica.



xx1100000680

Activar una salida digital antes de que el TCP alcance una zona mundo

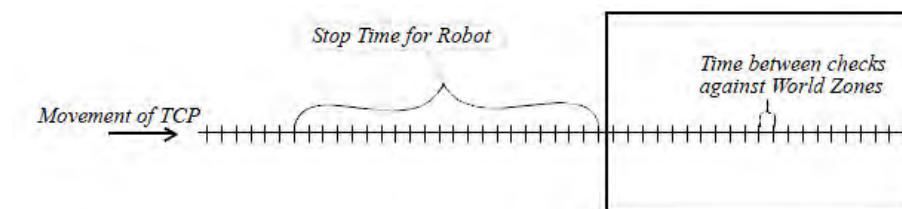
Esta acción activa una salida digital antes de que el TCP alcance una zona mundo. Puede usarse para detener el robot en cuanto entra en una zona mundo.



xx1100000681

Detener el robot antes de que el TCP alcance una zona mundo

Es posible definir una zona mundo de forma que quede fuera del área de trabajo. A continuación, el robot detendrá el punto central de la herramienta inmediatamente antes de entrar en la zona mundo en su movimiento hacia la zona mundo.



xx1100000682

Cuando el robot ha entrado en una zona mundo definida como un área de trabajo exterior, por ejemplo, cuando se liberan los frenos y se presiona manualmente, la

Continúa en la página siguiente

única forma de salir de la zona es con un movimiento o presionando manualmente con los frenos no aplicados.

Tamaño mínimo de las zonas mundo

La supervisión del movimiento de los puntos centrales de las herramientas se realiza en puntos discretos con un intervalo de tiempo entre ellos, acorde con la resolución de la trayectoria. El usuario es el responsable de hacer que las zonas tengan un tamaño suficiente como para que el robot no pueda atravesar una zona sin que se compruebe su posición dentro de ella.

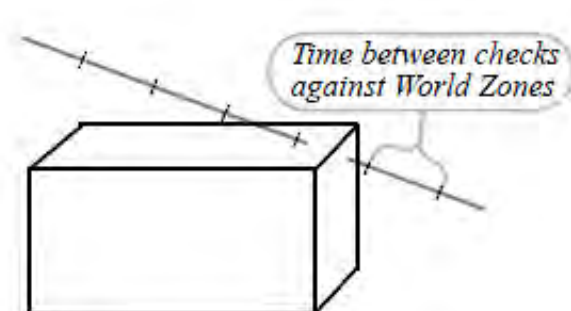
Asegúrese de que las zonas sean un poco más grandes que el tamaño mínimo.

Min. size of zone for used path_resolution and max. speed			
Speed Resol.	1000 mm/s	2000 mm/s	4000 mm/s
1	40 mm	80 mm	160 mm
2	80 mm	160 mm	320 mm
3	120 mm	240 mm	480 mm

xx1100000683

Si se utiliza la misma salida digital para más de una zona mundo, la distancia entre las zonas debe superar al tamaño mínimo indicado en la tabla anterior, con el fin de evitar un estado incorrecto de la salida.

Si el robot permanece dentro de la zona un tiempo demasiado breve, es posible que atraviese una esquina de la zona sin ser detectado. Por tanto, haga que la zona tenga un tamaño mayor que el área de riesgo.



xx1100000684

Si las zonas mundo son utilizadas en combinación con el servo suave, el tamaño de zona debe aumentarse adicionalmente para compensar el retardo del servo suave. El retardo del servo suave es la distancia existente entre el TCP del robot y la supervisión de la zona mundo en el momento de la interpolación. El retardo

Continúa en la página siguiente

2 Programación de movimiento y E/S

2.9 Zonas mundo

Continuación

de servo suave se incrementa con una mayor suavidad definida en la instrucción `SoftAct`.

Número máximo de zonas mundo

Es posible tener definido un máximo de 20 zonas mundo en un momento determinado.

Caídas de alimentación, reinicio y reanudación

Las *zonas mundo estacionarias* se eliminan al apagar el sistema y es necesario insertarlas de nuevo al encender el sistema, mediante una rutina de evento conectada al evento **POWER ON**.

Las *zonas mundo temporales* sobreviven a las caídas de alimentación, pero se eliminan cuando se carga un nuevo programa o cuando un programa se inicia desde el programa principal.

Las salidas digitales de las zonas mundo se actualizan en primer lugar a cambiar a **Motors ON**. Es decir, al reiniciar el controlador, el estado de la zona mundo cambiará a fuera durante el inicio. En la primera orden **MOTORS ON** tras un reinicio, el estado de la zona mundo se actualiza correctamente.

Si se mueve el robot durante **MOTORS OFF**, el estado de la zona mundo no se actualiza hasta la siguiente orden **MOTORS ON**.

Un paro de emergencia rígido (ni `SoftAS`, `SoftGS` ni `SoftES`) puede dar lugar a un estado incorrecto de zona mundo porque el robot puede entrar o salir de una zona durante el movimiento de paro sin que se actualicen las señales de zona mundo. Las señales de zona mundo se actualizan correctamente tras una orden **MOTORS ON**.

Información relacionada

Principios de movimiento y E/S	Sistemas de coordenadas
Tipos de datos: <ul style="list-style-type: none">• <code>wztemporary</code>• <code>wzstationary</code>• <code>shapedata</code>	<i>Manual de referencia técnica - Instrucciones, funciones y tipos de datos de RAPID</i>
Instrucciones: <ul style="list-style-type: none">• <code>WZBoxDef</code>• <code>WZSphDef</code>• <code>WZCylDef</code>• <code>WZHomeJointDef</code>• <code>WZLimJointDef</code>• <code>WZLimSup</code>• <code>WZDOSet</code>• <code>WZDisable</code>• <code>WZEnable</code>• <code>WZFree</code>	<i>Manual de referencia técnica - Instrucciones, funciones y tipos de datos de RAPID</i>

2.10 Principios de E/S

Descripción

Normalmente, el robot cuenta con una o varias placas de E/S. Cada una de las placas tiene varios canales digitales y/o analógicos que deben conectarse a señales lógicas para poder usarlos. Esto se realiza en los parámetros del sistema y suele estar ya hecho con nombres estándar antes de la entrega del robot. Siempre deben usarse señales lógicas durante la programación.

Un mismo canal físico puede conectarse a varias señales lógicas, pero también puede quedar sin ninguna conexión lógica (consulte la *Figura 57*).

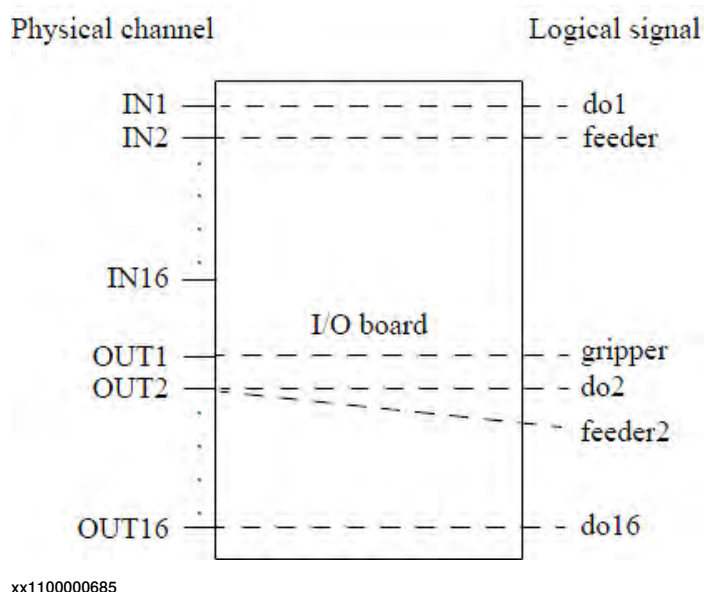


Figura 57: Para poder usar una placa de E/S, sus canales deben tener asignadas señales lógicas. En el ejemplo anterior, la salida física 2 está conectada a dos señales lógicas diferentes. Por otro lado, IN16 no tiene ninguna señal lógica y por tanto no puede usarse.

Características de las señales

Las características de una señal dependen del canal físico utilizado, así como de la forma de definir el canal en los parámetros del sistema. El canal físico determina los retardos de tiempo y los niveles de tensión (consulte la *Especificación del producto*). Las características, los tiempos de filtro y la escala entre los valores programados y los físicos se definen en los parámetros del sistema.

Cuando se enciende la fuente de alimentación del robot, todas las señales tienen el valor cero. Sin embargo, no se ven afectadas por paros de emergencia ni eventos similares.

Las salidas pueden recibir el valor uno o cero desde dentro del programa. Esto también puede hacerse utilizando un retardo o en forma de un pulso. Si se solicita un pulso o un cambio retardado para una salida, la ejecución del programa continúa. A continuación, el cambio se implementa sin que el resto de la ejecución del programa se vea afectada. Si por otro lado se solicita un nuevo cambio para la

Continúa en la página siguiente

2 Programación de movimiento y E/S

2.10 Principios de E/S

Continuación

misma salida antes de que transcurra el tiempo especificado, no se realiza el primer cambio (consulte la *Figura 58*).

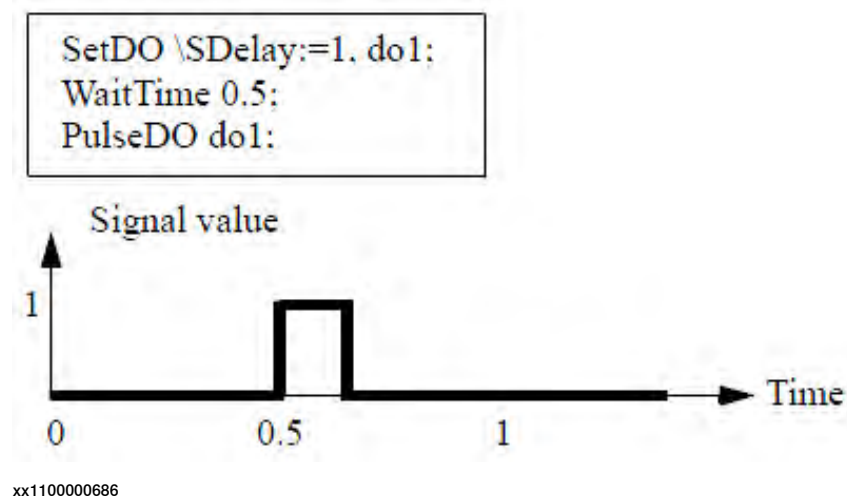


Figura 58: La instrucción SetDO no llega a realizarse porque se emite un nuevo comando antes de que transcurra el retardo especificado.

Señales conectadas a interrupciones

Las funciones de interrupciones de RAPID pueden conectarse a cambios de señales digitales. La llamada a la función puede producirse en el momento de activar o desactivar una señal. Sin embargo, si la señal digital cambia con mucha rapidez, es posible que la interrupción no se procese.

Por ejemplo, si se conecta una función a una señal denominada *do1* y el código del programa es el siguiente:

```
SetDO do1,1;
SetDO do1,0;
```

La señal se activa (1) y se desactiva (0) a continuación en unos pocos milisegundos. En este caso, la interrupción puede haberse perdido. Para asegurarse de que no se pierde la interrupción, asegúrese de que la salida está activada antes de ponerla a cero.

Por ejemplo:

```
SetDO do1,1;
WaitDO do1,1;
SetDO do1,0;
```

Con este método, no se perderá ninguna interrupción.

Señales del sistema

Es posible conectar entre sí las señales lógicas mediante funciones especiales del sistema. Por ejemplo, si se conecta una entrada a la función *Start* del sistema, se genera automáticamente un inicio de programa tan pronto como se activa esta entrada. Normalmente, estas funciones de sistema sólo están activadas en el modo automático.

Continúa en la página siguiente

Conexiones cruzadas

Las señales digitales pueden conectarse entre sí de forma que una afecte automáticamente a la otra.

- Es posible conectar una señal de salida a una o varias señales de entrada o salida.
- Es posible conectar una señal de entrada a una o varias señales de entrada o salida.
- Si se utiliza la misma señal en varias conexiones cruzadas, el valor de la señal es el mismo que el valor que se activó (modificó) en último lugar.
- Las conexiones cruzadas pueden estar vinculadas entre sí. En otras palabras, una conexión cruzada puede actuar sobre otra. Sin embargo, no deben estar conectadas de forma que creen un "círculo vicioso", por ejemplo si se hace una conexión cruzada de $di1$ a $di2$ mientras $di2$ tiene una conexión cruzada a $di1$.
- Si existe una conexión cruzada con una señal de entrada, la conexión física correspondiente se desactiva automáticamente. Por tanto, no se detectará ningún cambio que se haga en el canal físico.
- Ni los pulsos ni los retardos se transmiten a través de las conexiones cruzadas.
- Es posible definir condiciones lógicas mediante NOT, AND y OR (requiere la opción *Advanced functions*).

Ejemplos	Descripción
$di2=di1$ $di3=di2$ $do4=di2$	Si $di1$ cambia, $di2$, $di3$ y $do4$ cambiarán al valor correspondiente.
$do8=do7$ $do8=di5$	Si se cambia el valor de $do7$ a 1, $do8$ también pasa a tener el valor 1. Si a continuación se cambia el valor de $di5$ a 0, se cambia también el valor de $do8$ (a pesar de que $do7$ sigue teniendo el valor 1).
$do5 = di6 \text{ AND } do1$	$do5$ cambia a 1 si tanto $di6$ como $do1$ cambian a 1.

Limitaciones

Es posible utilizar pulsos con un máximo de 10 señales y retardos con un máximo de 20 señales en un momento determinado.



Información relacionada

	Se describe en
Definición de tarjetas y señales de E/S	<i>Manual de referencia técnica - Parámetros del sistema</i>
Instrucciones para el manejo de E/S	Señales de entrada y salida en la página 66
Manipulación manual de E/S	<i>Manual del operador - IRC5 con FlexPendant</i>

Esta página se ha dejado vacía intencionadamente

3 Glosario

Glosario

Término	Descripción
Argumento	Las partes de una instrucción que pueden cambiarse, es decir, todo menos el nombre de la instrucción.
Modo automático	El modo aplicable cuando se cambia el selector de modo a  xx1100000688
Componente	Una parte de un registro.
Configuración	La posición de los ejes del robot en un punto determinado.
Constante	Un dato cuyo valor sólo puede cambiarse manualmente.
Trayectoria de esquina	La trayectoria generada al atravesar un punto de paso.
Declaración	La parte de una rutina o de un dato en la que se definen sus propiedades.
Diálogo/Ventana de diálogo	Las ventanas de diálogo del FlexPendant deben confirmarse siempre (normalmente tocando OK o Cancelar) para poder cerrarlas.
Gestor de errores	Dentro de una rutina, la parte separada en la que se gestiona un error. Después del gestor, la ejecución se reanuda automáticamente.
Expresión	Una secuencia de datos y operandos asociados, por ejemplo <code>reg1+5</code> o <code>reg1>5</code> .
Punto de paso	Un punto al que se acerca el robot en su avance, pero sin detenerse en él. La distancia existente hasta el punto depende del tamaño de la zona programada.
Función	Una rutina que devuelve un valor.
Señal de grupo	Un número de señales digitales que se agrupan y se gestionan como una sola señal.
Interrupción	Un evento que interrumpe temporalmente la ejecución del programa y ejecuta una rutina TRAP.
E/S	Entradas y salidas eléctricas.
Rutina Main	La rutina que suele ponerse en marcha cuando se presiona el botón de inicio.
Modo manual	El modo aplicable cuando se cambia el selector de modo a  xx1100000687
Unidad mecánica	Un grupo de ejes adicionales.

Continúa en la página siguiente

Término	Descripción
Módulo	Un grupo de rutinas y datos, es decir, una parte del programa.
Motors ON/OFF	El estado del robot, es decir, si la fuente de alimentación de los motores está activada.
Panel del operador	El panel situado en la parte delantera del controlador.
Orientación	La dirección de un elemento terminal.
Parámetro	El dato de entrada de una rutina, enviado con la llamada a la rutina. Corresponde al argumento de una instrucción.
Variable persistente	Una variable cuyo valor es persistente.
Procedimiento	Una rutina que puede formar independientemente una instrucción cuando se la llama.
Programa	El conjunto de instrucciones y datos que definen la tarea del robot. Sin embargo, los programas no contienen módulos de sistema.
Datos de programa	Los datos que están disponibles en todo un módulo o en el programa completo.
Módulo de programa	Un módulo incluido en el programa del robot y que se transfiere al copiar el programa a un disquete.
Registro	Un tipo de datos compuesto.
Rutina	Un subprograma.
Datos de rutina	Datos locales que sólo pueden usarse en una rutina.
Punto de inicio	La instrucción que se ejecuta en primer lugar cuando comienza la ejecución del programa.
Punto de paro	Un punto en el que el robot se detiene antes de continuar al punto siguiente.
Módulo de sistema	Un módulo que está siempre presente en la memoria de programas. Cuando se lee un nuevo programa, los módulos de sistema permanecen en la memoria de programas.
Parámetros del sistema	Los parámetros que definen el equipo y las propiedades del robot, o los datos de configuración, en otras palabras.
Punto central de la herramienta (TCP)	El punto, normalmente en la punta de una herramienta, que se mueve siguiendo una trayectoria programada y a la velocidad programada.
Rutina TRAP	La rutina que define qué debe ocurrir cuando se produce una interrupción determinada.
Variable	Un dato cuyo valor puede cambiarse desde un programa pero que pierde su valor (vuelve a su valor inicial) cuando se pone en marcha un programa desde el principio.
Ventana	El robot se programa y maneja a través de ventanas (o vistas) en el FlexPendant, por ejemplo la ventana Editor de programas y la ventana Calibración . Es posible salir de una ventana cambiando a otra ventana o tocando el botón de cierre de la esquina superior derecha.
Zona	El espacio esférico que rodea a un punto de paso. Tan pronto como el robot entra en esta zona, comienza a trasladarse hacia la siguiente posición.

Índice

A

agregado, 28
 agregados
 expresiones, 40
 ámbito
 datos, 30
 rutina, 21
 AND, 37
 argumento, 173
 condicional, 41
 argumento condicional, 41
 argumentos
 descripción, 11
 asignación de un valor a un dato, 49

B

bool, 50

C

cadena de caracteres, 14
 calibración, 97
 carga de módulos, 49
 cinemática del robot, 151
 comentarios, 14, 49
 componente de un registro, 28, 173
 comunicación, 91
 comunicación binaria, 70
 comunicación de canal serie, 70
 comunicación de rawbytes, 71
 comunicación de zócalo, 71
 conexiones cruzadas, 171
 confdata, 148
 configuración, 173
 robot, 147
 configuración de ejes, 147
 configuración del robot, 147
 ConfJ, 150
 ConfL, 150
 CONST, 33
 constante, 30, 173
 constantes, 33
 valores de inicialización, 33
 conversión, 99
 conversiones, 51
 corrección de trayectoria, 61
 cuartos de revolución, 148

D

datos, 28
 ámbito, 30
 asignación de valores, 49
 clase de almacenamiento, 34
 constante, 30
 declaraciones, 30
 descripción, 11
 inicialización, 33
 persistente, 30
 programa, 30
 rutina, 31
 utilizados en expresiones, 39
 variable, 30
 datos de configuración, 94
 datos del sistema, 93
 datos de movimiento, 64

datos de programa, 28, 30, 174
 datos de rutina, 31, 174
 declaración, 173
 módulo, 18
 rutina, 24
 declaración de función, 24
 declaración de módulos, 18
 declaración de procedimiento, 24
 declaración de rutina, 24
 declaración de rutina TRAP, 24
 declaraciones
 constantes, 33
 variables, 31
 variables persistentes, 32
 detección de colisiones, 63, 156
 detención de la ejecución del programa, 47
 distinción entre mayúsculas y minúsculas, 12
 DIV, 36
 dnum, 50

E

E/S, 173
 E/S de posición fija, 145
 ejecución hacia atrás, 108
 ejecución simultánea, 143
 ejes adicionales, 60, 119
 ejes adicionales coordinados, 119
 ejes independientes, 61, 137
 encabezado de archivo, 15
 ERRNO, 81
 expresión, 173
 expresiones, 36
 aritméticas, 36
 cadena de caracteres, 38
 lógicas, 37
 expresiones aritméticas, 36
 expresiones de cadena de caracteres, 38
 expresiones lógicas, 37

F

función, 21, 173
 funciones aritméticas, 88
 funciones de bits, 89
 funciones de estado, 64
 funciones de posición, 63
 funciones para cadenas de caracteres, 99
 funciones para operaciones con archivos, 92

G

gestor de ejecución, 95
 gestor de ejecución hacia atrás, 108
 gestor de errores, 173
 gestores de errores, 81
 global
 datos, 30
 rutina, 21
 glosario, 173
 grabadora de trayectorias, 62

I

identificación de carga, 63
 identificadores, 13
 información de servicio, 96
 inicialización de datos, 33
 INOUT, 22
 instrucciones
 descripción, 11

- flujo de programa, 47
- listas de selección, 46
- instrucciones de archivos, 70
- instrucciones de búsqueda, 58
- instrucciones de comunicaciones, 69
- instrucciones de espera, 49
- instrucciones de flujo de programa, 47
- instrucciones de movimiento, 57
- instrucciones de posicionamiento, 57
- instrucciones de tiempo, 86
- instrucciones matemáticas, 88
- instrucciones move, 57
- interpolación, 126, 130
- interpolación circular, 127
- interpolación de ejes, 126
- interpolación lineal, 126
- interpolación lineal modificada, 129
- interrupción, 173
- interrupciones, 58, 74

L

- listas de selección, 46
- llamada a procedimiento, 25
- llamadas a funciones, 41
- local
 - datos, 30
 - rutina, 21

M

- marcadores de sustitución, 15
- matrices
 - expresiones, 39
 - variables, 31
- memoria, 94
- MOD, 37
- modelos cinemáticos, 151
- modo automático, 173
- modo manual, 173
- módulo, 174
- módulo de programa, 17, 174
- módulo de sistema, 18, 174
- módulo de sistema User, 20
- módulos, 17
 - descripción, 17
- motors ON/OFF, 174
- movimiento, 57
- movimiento circular, 57, 127
- movimiento de ejes, 57, 126
- movimiento lineal, 57, 126
- MultiMove, 60, 102
- multitarea, 101

N

- nivel de ejecución, 95
- NOT, 37
- num, 50
- números de error, 79

O

- OR, 37
- orientación, 174

P

- palabras reservadas, 13
- panel del operador, 174
- parámetro, 174
- parámetro opcional, 22

- parámetros, 22
- parámetros del sistema, 174
- parámetros de movimiento
 - instrucciones, 52
- paro, 141
- PERS, 32
- persistente, 30
- principios de E/S, 169
- prioridad
 - operadores, 43
 - tareas, 105
- prioridad de operadores, 43
- procedimiento, 21, 174
- programa, 17, 174
- punto central de la herramienta, 113, 174
- punto de inicio, 174
- punto de paro, 174
- punto de paso, 130, 142, 173

R

- RAPID Message Queues, 72
- recuperación en caso de error, 79
- registro, 28, 174
- registro de eventos, 82
- registros
 - expresiones, 39
- reglas de sintaxis, 9
- reinicio del controlador, 94
- reloj, 86
- robot con eslabones en serie, 161
- rutina, 21, 174
- rutina de finalización, 23
- rutina main, 17
- rutina Main, 173
- rutinas
 - descripción, 11
- rutinas TRAP, 74, 77
- rutina TRAP, 21, 174

S

- seguimiento de transportadores, 62
- seguimiento servo, 62
- Sensor Synchronization, 63
- señal de grupo, 173
- señales, 66, 169
- señales de E/S, 66
- señales de entrada, 66
- señales de salida, 66
- servicio, 97
- servo suave, 54, 140
- sincronización, 142
- sincronización de E/S, 142
- sincronización de trayectorias, 145
- singularidades, 129, 160
- sistema de coordenadas de desplazamiento, 118
- sistema de coordenadas de la base, 114, 120
- sistema de coordenadas de la herramienta, 121
- sistema de coordenadas del usuario, 116, 119
- sistema de coordenadas de muñeca, 121
- sistema de coordenadas de objeto, 117
- Sistema de coordenadas mundo, 114
- sistemas de coordenadas, 113, 151
- string, 50
- supervisión
 - configuración del robot, 149
- supervisión de la configuración del robot, 149
- supervisión del movimiento, 156

switch, 22, 50

T

tareas, 95, 101

TCP, 113, 174

estacionario, 123

TCP estacionario, 123

tipo de dato atómico, 28

tipo de evento, 95

tipos de datos, 28

agregados, 28

alias, 28

atómicos, 28

componentes, 28

registro, 28

semivalor, 28

sin valor, 28

tipos de datos de alias, 28

tipos de datos de igualdad, 28

tipos de datos de semivalor, 28

tipos de datos sin valor, 28

transportador de indexación, 62

trayectoria de esquina, 130, 173

trayectoria interrumpida, 64

U

UNDO, 83

unidad mecánica, 173

V

valores lógicos, 14

valores numéricos, 14

VAR, 31

variable, 30, 174

variable persistente, 174

variables, 31

matrices, 31

valores de inicialización, 33

variables persistentes, 32

valores de inicialización, 33

ventana, 174

ventana de diálogo, 173

X

XOR, 37

Z

zona, 130, 174

zonas mundo, 55, 164

Contact us

ABB AB

**Discrete Automation and Motion
Robotics**

S-721 68 VÄSTERÅS, Sweden

Telephone +46 (0) 21 344 400

ABB AS, Robotics

Discrete Automation and Motion

Nordlysvegen 7, N-4340 BRYNE, Norway

Box 265, N-4349 BRYNE, Norway

Telephone: +47 51489000

ABB Engineering (Shanghai) Ltd.

No. 4528 Kangxin Hingway

PuDong District

SHANGHAI 201319, China

Telephone: +86 21 6105 6666

www.abb.com/robotics