

SOFTWARE

KR C...

Programación por el experto

KUKA System Software (KSS)

Release 4.1

© Copyright **KUKA Roboter GmbH**

La reproducción de esta documentación – o parte de ella – o su facilitación a terceros solamente está permitida con expresa autorización del editor.

Además del volumen descrito en esta documentación, pueden existir funciones en condiciones de funcionamiento. El usuario no adquiere el derecho sobre estas funciones en la entrega de un aparato nuevo, ni en casos de servicio.

Hemos controlado el contenido del presente escrito en cuanto a la concordancia con la descripción del hardware y el software. Aún así, no pueden excluirse totalmente todas las divergencias, de modo tal, que no aceptamos responsabilidades respecto a la concordancia total. Pero el contenido de estos escritos es controlado periódicamente, y en casos de divergencia, éstas son enmendadas y presentadas correctamente en las ediciones siguientes.

Reservados los derechos a modificaciones técnicas que no tengan influencia sobre la función.

PD Interleaf

Índice

1	Generalidades sobre los programas KRL	7
1.1	Estructura y creación de programas	7
1.1.1	Superficie del programa	7
1.1.2	Concepto de archivo	9
1.1.3	Estructura de archivos	9
1.2	Crear programas y editarlos	11
1.2.1	Crear un programa nuevo	11
1.2.2	Editar, compilar y vincular un programa	12
1.3	Modificación de programas	14
1.3.1	Corrección de programa	14
1.3.2	Editor	14
1.3.2.1	Funciones de bloque	14
1.3.2.2	Copiar (CTRL-C)	14
1.3.2.3	Insertar (CTRL-V)	15
1.3.2.4	Cortar (CTRL-X)	15
1.3.2.5	Borrar	15
1.3.2.6	Buscando...	16
1.3.2.7	Reemplazar	16
1.4	Esconder partes de un programa	19
1.4.1	FOLD	19
1.4.1.1	Programa de ejemplo	20
1.5	Modos de ejecución del programa	22
1.6	Tratamiento de errores	24
1.7	Comentarios	27
2	Variables y declaraciones	29
2.1	Variables y nombres	29
2.2	Objetos de datos	31
2.2.1	Declaración e inicialización de objetos de datos	31
2.2.2	Tipos de datos simples	33
2.2.3	Campos	35
2.2.4	Cadenas de caracteres	38
2.2.5	Estructuras	38
2.2.6	Tipos de enumeración	40
2.3	Manipulación de datos	42
2.3.1	Operadores	42
2.3.1.1	Operadores aritméticos	42
2.3.1.2	Operador geométrico	43
2.3.1.3	Operadores de comparación	47
2.3.1.4	Operadores lógicos	48
2.3.1.5	Operadores de bits	49
2.3.1.6	Prioridades de los operadores	51
2.3.2	Funciones estándar	52
2.4	Variables y archivos del sistema	54
3	Programación de movimiento	59
3.1	Utilización de diferentes sistemas de coordenadas	59

3.2	Movimientos punto a punto (PTP)	66
3.2.1	General (PTP sincrónico)	66
3.2.2	Perfil de marcha más elevado	67
3.2.3	Instrucciones de movimiento	68
3.3	Movimientos sobre trayectorias (CP–Movimientos = Continuous Path)	77
3.3.1	Velocidad y aceleración	77
3.3.2	Control de la orientación	78
3.3.3	Movimientos lineales	83
3.3.4	Movimientos circulares	84
3.4	Procesamiento en avance	86
3.5	Movimientos con posicionamiento aproximado	89
3.5.1	Aproximación PTP–PTP	90
3.5.2	Aproximación LIN–LIN	93
3.5.3	Aproximación CIRC–CIRC y aproximación CIRC–LIN	96
3.5.4	Aproximación PTP – trayectoria	99
3.5.5	Cambio de herramienta con posicionamiento aproximado	102
3.6	Programación por aprendizaje de los puntos	103
3.7	Parámetros de movimiento	104
4	Asistente KRL	105
4.1	Indicaciones de posición	106
4.2	Posicionamiento [PTP]	109
4.3	Movimiento lineal [LIN]	111
4.4	Movimientos circulares [CIRC]	113
5	Control de ejecución del programa	115
5.1	Ramificaciones de programa	115
5.1.1	Instrucciones de salto	115
5.1.2	Ramificación condicionada	116
5.1.3	Distribuidor	117
5.2	Bucles	118
5.2.1	Bucles de conteo	118
5.2.2	Bucle rechazante	120
5.2.3	Bucles no rechazantes	121
5.2.4	Bucles sinfín	123
5.2.5	Terminación anticipada de la ejecución de bucles	123
5.3	Instrucciones de espera	124
5.3.1	Espera a un evento	124
5.3.2	Tiempos de espera	124
5.4	Detener el programa	125
5.5	Confirmación de mensajes	126
6	Instrucciones de entrada/salida	127
6.1	Generalidades	127
6.2	Entradas y salidas binarias	128
6.3	Entradas y salidas digitales	131

6.3.1	Declaración de la señal	131
6.3.2	Activar salidas en el punto de destino	133
6.4	Salidas de impulso	136
6.5	Entradas y salidas analógicas	138
6.5.1	Salidas analógicas	138
6.5.2	Entradas analógicas	141
6.6	Entradas digitales predefinidas	143
7	Subprogramas y funciones	145
7.1	Declaraciones	145
7.2	Llamada y transferencia de parámetros	148
8	Tratamiento de interrupciones	153
8.1	Declaración	154
8.2	Activación de interrupciones	156
8.3	Detención de movimientos en ejecución	160
8.4	Cancelación de rutinas de interrupción	161
8.5	Utilización de banderas (flags) cíclicas	164
9	Trigger – Acciones de conmutación referentes a la trayectoria	165
9.1	Acciones de conmutación en el punto de inicio o de destino de la trayectoria	165
9.2	Acciones de conmutación en cualquier lugar sobre la trayectoria	169
9.3	Sugerencias & trucos	174
9.3.1	Interacción de instrucciones de disparo (Trigger)	174
10	Listas de datos	175
10.1	Listas de datos locales	175
10.2	Listas de datos globales	176
11	Editor externo	179
11.1	Arranque del editor externo	180
11.2	Operación	182
11.3	Menú “Fichero”	185
11.3.1	Abrir	185
11.3.2	Guardar	185
11.3.3	Imprimir (“mnuPrint”)	186
11.3.4	Cerrar	186
11.3.5	Salir	187
11.4	Menú “Editar”	188
11.4.1	Cortar (“CTRL”-“X”)	188
11.4.2	Copiar (“CTRL”-“C”)	188
11.4.3	Pegar como	188
11.4.4	Borrar	189
11.4.5	Seleccionar todo	189
11.5	Menú “Utilidades”	190

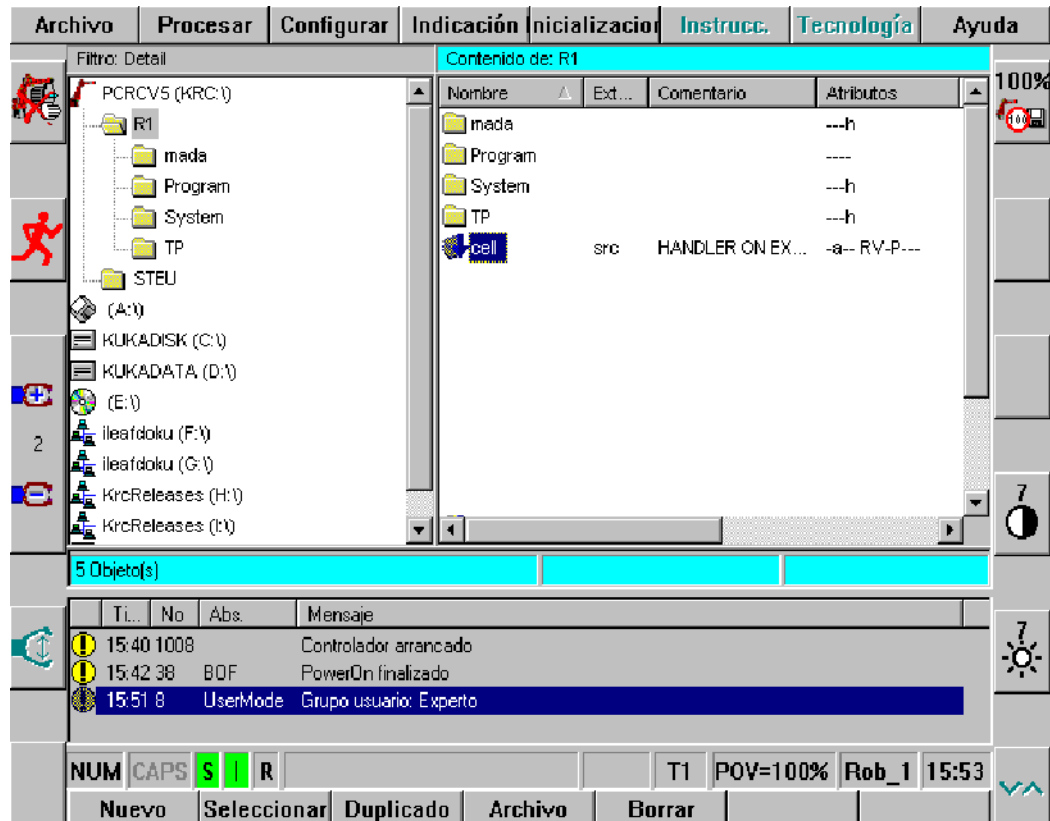
11.5.1	Espejo	190
11.5.2	Entrada manual	191
11.5.3	Cambio de bloque	195
11.5.4	Limpiar lista de ficheros	195
11.5.5	Ajuste del TCP o de BASE	196
11.6	Menú “HotEdit”	197
11.6.1	Base, TCP y World	197
11.6.2	Limits	199
11.6.3	TTS (Sistema de coordenadas de corrección)	200
11.6.3.1	Position-TTS	200
11.6.3.2	Limits-TTS	204
11.7	Menú “Ext Utilidades”	205
11.7.1	Fichero – Espejo	205
11.7.2	Fichero – Cambiar	208
11.7.2.1	Utilizar el fichero de referencia existente	209
11.7.2.2	Crear un nuevo fichero de referencia	211
11.7.3	Limite cambiado	212
11.8	Menú “Opciones”	213
11.8.1	Output setting (Declaraciones sobre la edición)	213
11.9	Menú “Ayuda”	215
11.9.1	Versión	215
11.9.2	Permanecer arriba	215

1 Generalidades sobre los programas KRL

1.1 Estructura y creación de programas

1.1.1 Superficie del programa

En el instante de haber cambiado al nivel del experto, se modifica la superficie de operación del modo representado a continuación:



Mientras que para el usuario son invisibles todos los archivos de sistema, el experto puede verlos y también editarlos en la ventana de programas. Además, junto a los nombres de archivo y comentarios, en el nivel del experto pueden mostrarse también las extensiones, atributos y tamaños de los archivos.

En la presente figura se muestran en la ventana de programas, los ficheros y los directorios del trayecto de búsqueda "R1".

De forma estándar, después de la instalación del software de la KR C1, se encuentran en el directorio "KRC:\R1\MADA\" los ficheros listados a continuación:

Fichero	Significado
\$MACHINE.DAT	Lista de datos del sistema con variables del sistema para la adaptación de la unidad de control y del robot
\$ROBCOR.DAT	Lista de datos del sistema con datos para el modelo dinámico del robot
MACHINE.UPG	Fichero del sistema para futuras actualizaciones
ROBCOR.UPG	Fichero del sistema para futuras actualizaciones

En el directorio "KRC:\R1\SYSTEM\" se encuentran los siguientes ficheros:

Fichero	Significado
\$CONFIG.DAT	Lista de datos del sistema con datos generales de configuración
BAS.SRC	Paquete básico para el comando de movimientos
IR_STOPM.SRC	Programa para el tratamiento de errores cuando se producen averías
SPS.SUB	Fichero submit para controles paralelos

En el directorio "KRC:\R1\TP\" se encuentran los siguientes ficheros:

Fichero	Significado
A10.DAT A10.SRC	Paquete tecnológico para la soldadura de trayectoria con tensiones de mando analógicas
A10_INI.DAT A10_INI.SRC	Paquete tecnológico para la inicialización de la soldadura de trayectoria con tensiones de mando analógicas
A20.DAT A20.SRC	Paquete tecnológico para la soldadura de trayectoria con números de programa digitales
A50.DAT A50.SRC	Paquete tecnológico para el uso del sensor de arco (de soldadura)
ARC_MSG.SRC	Programa para la programación de mensajes para la soldadura al arco.
ARCSPS.SUB	Archivo submit para la soldadura de trayectoria
BOSCH.SRC	Programa para la soldadura por puntos con interfaz serie para el control de soldadura por puntos Bosch PSS5200.521C
COR_T1.SRC	Programa para la corrección de la herramienta (versión antigua)
CORRTOOL.DAT CORRTOOL.SRC	Programa para la corrección de la herramienta
FLT_SERV.DAT FLT_SERV.SRC	Programa para el tratamiento de errores definido por el usuario, durante la soldadura de trayectoria
H50.SRC	Paquete para trabajos con garras
H70.SRC	Paquete para el sensor táctil
MSG_DEMO.SRC	Programa con ejemplos para los mensajes de usuario
NEW_SERV.SRC	Programa para la modificación de reacciones ante la presencia de fallos para FLT_SERV

P00.DAT P00.SRC	Paquete de programas para el acoplamiento a un PLC
PERCEPT.SRC	Programa para la llamada del protocolo PERCEPTRON
USER_GRP.DAT USER_GRP.SRC	Programa para el control de la garra, definido por el usuario
USERSPOT.DAT USERSPOT.SRC	Paquete de programas para la soldadura por puntos, definida por el usuario
WEAV_DEF.SRC	Programa para los movimientos de oscilación durante la soldadura al arco

En el directorio "KRC:\R1\" se encuentra el siguiente fichero:

CELL.SRC	Programa para el mando de robots a través de un PLC central. Aquí, se selecciona a partir de un número de programa, el programa de la pieza correspondiente
----------	---

1.1.2 Concepto de archivo

Un programa KRL puede estar compuesto de un fichero SRC y de un fichero DAT.



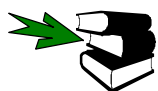
Informaciones más detalladas respecto a la creación de programas se encuentran en este capítulo, apartado **[Crear programas y editarlos]**.

Aquí, el archivo con la extensión "SRC" es el fichero con el propio código de programa. Aquí existen las variantes DEF y DEFFCT (con retorno de valor). Por el contrario, el fichero con la extensión "DAT" contiene los datos específicos para el programa. Esta partición está basada en el concepto de administración de ficheros por KRL. El programa contiene junto a las instrucciones de movimiento, distintas acciones que deben ser ejecutadas por el robot. Estas pueden ser determinadas secuencias de movimientos, por ejemplo abrir o cerrar la garra de un útil, amén de otras ejecuciones más complejas tales como el mando de una pinza de soldadura bajo consideración de condiciones de borde.

Para efectuar el test de programas es útil, y a veces necesario, poder ejecutar tareas parciales. El concepto de administración de ficheros realizado en KRL, cumple con las necesidades especiales de la programación del robot.

1.1.3 Estructura de archivos

Un archivo es la unidad que programa el programador. Con ello, un archivo corresponde a una unidad de programa en el disco duro o en la memoria (RAM). Cada programa en KRL puede constar de uno o varios archivos. Los programas simples comprenden exactamente un archivo. Las tareas más complejas se solucionan mejor con un programa que consta de varios archivos.



Informaciones detalladas acerca de los subprogramas y funciones se encuentran en el apartado **[Subprogramas y funciones]**.



La estructura interior de un archivo KRL consta de la sección de declaraciones, sección de instrucciones, así como hasta 255 subprogramas y funciones locales.

DEF

El nombre del objeto sin extensión es al mismo tiempo el nombre del archivo y, por lo tanto, se indica en la declaración con "DEF". El nombre debe constar de un máximo de 24 caracte-

res y no debe ser ninguna palabra clave (véase apartado **[Variables y declaraciones]**). Cada archivo comienza con la declaración “DEF” y finaliza con “END”.

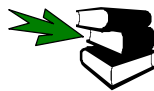
```
DEF NAME (X1:IN)
Declaraciones
Instrucciones
END
```

- | | |
|----------------|---|
| Declaración | Las declaraciones se evalúan ya antes del procesamiento del programa, es decir, durante la compilación. En la sección de declaraciones no debe haber ninguna instrucción. La primera instrucción es al mismo tiempo el comienzo de la sección de instrucciones. |
| Instrucción | Contrariamente a lo que ocurre en la sección de declaraciones, las instrucciones tienen un carácter dinámico: se ejecutan durante el procesamiento del programa. |
| Lista de datos | Un programa del robot puede constar de un solo fichero de programa o de un fichero de programa con la correspondiente lista de datos. La lista de datos y el fichero se señalizan como correspondientes mediante sus nombres comunes. Los nombres solamente se diferencian por la extensión, por ejemplo: |

```
Fichero :      PROG1.SRC
Lista de datos:  PROG1.DAT
```



En la lista de datos sólo se permiten asignaciones de valores con “=”. Cuando la lista de datos y el fichero poseen el mismo nombre, pueden utilizarse variables que están declaradas en la lista de datos, del mismo modo también como variables declaradas en el fichero SRC.



Informaciones detalladas acerca de este tema se encuentran en el apartado **[Listas de datos]**.

1.2 Crear programas y editarlos

1.2.1 Crear un programa nuevo

Nuevo

Dado que un programa para robot puede crearse también sin lista de datos, el archivo y la lista de datos en el nivel de experto no se crean automáticamente de forma simultánea. Para crear un programa, pulse la tecla del softkey “Nuevo”. Aparece la siguiente ventana:

Selección de estructura	
Nombre archivo	Comentario filtro
Cell	Automatik extern dispatch
Expert	Expert module
Expert Submit	Expert submit
Function	Function
Modul	Module
Submit	User submit

Por favor, seleccione estructura

OK

Se le requiere seleccionar un Template. Utilice para ello las teclas de flechas y confirme con el softkey “OK” o con la tecla de entrada.



Los Templates disponibles no pueden ser declarados en todos los directorios.



Informaciones más detalladas acerca de los Templates se encuentran en el **Manual de operación** en la documentación **[Instrucciones]**, capítulo **[Navegador]**, apartado **[Apéndice]**.

Los distintos Templates de forma individual:

Modul:

Se crea un fichero SRC y un fichero DAT, que contiene un programa base o tronco de programa.

Expert:

Se crea un fichero SRC y un fichero DAT, que contienen solo el encabezamiento DEF... y END.

Cell:

Aquí solo se crea un fichero SRC, que contiene un programa base o tronco de programa. Este programa sirve para el mando del robot a través de un PLC central.

Function:

Aquí se crea una función (fichero SRC), que contiene el encabezamiento DEF... y END.

Submit:

Se crea un fichero SUB con una base o tronco de programa. El fichero Submit contiene instrucciones y puede ser utilizado por ejemplo, para controles cíclicos (garras, etc.). El fichero Submit trabaja en forma paralela al servicio del robot y es ejecutado por el interpretador de la unidad de control.


Expert Submit:


Como en el caso del Template Submit, se crea un fichero SUB, pero que solo contiene el encabezamiento DEF... y END.





La línea de encabezamiento DEF... y END y las bases o troncos de programas de cada Template, se encuentran, por ejemplo, para el Template Cell bajo “C:\KRC\ROBOTER\TEMPLATE\CellVorgabe.src”.

Si Ud. ha seleccionado el correspondiente Template, se le requiere dar un nombre para el fichero generado.

Name	D...	Kommentar	Attribute	Grö...
	---		---	---


 Nombre de fichero
(máx. 24 caracteres)


 Extensión de archivo
(SRC, DAT o SUB)


 Comentario

El nombre del fichero es lo único obligatorio y puede tener, como máx., una longitud de 24 caracteres. La extensión del fichero es completada automáticamente. Si Ud. quiere adicionar un comentario, desplace el cursor con la tecla de flecha derecha sobre el campo correspondiente, y de entrada al texto deseado.

OK



La lista de datos es imprescindible cuando desee insertar también instrucciones guiadas por menú en su archivo SRC.

1.2.2 Editar, compilar y vincular un programa

Si ha creado un fichero o una lista de datos con "Nuevo", puede procesarlos con el editor. Para ello sirve el softkey "Abrir". Al cerrar el editor, se compila el código de programa completo, es decir, el código KRL textual se traduce a un lenguaje máquina comprensible para la unidad de control.



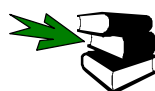
Para que el programa quede confeccionado de forma clara y entendible, es necesario, por ej. de forma ramificada, colocar las ramificaciones en distintos niveles. En el editor puede efectuarse ésto con espacios en blanco.

Compilador

El compilador comprueba el código para verificar que la sintaxis y la semántica son correctas. En caso de existencia de errores, se emite el correspondiente mensaje de fallos y se genera un archivo de fallos con la extensión ".ERR".



Sólo se permite seleccionar y ejecutar programas libre de errores.



Informaciones adicionales respecto al tratamiento de errores o fallos de edición, se encuentran en el apartado **[Tratamiento de errores]**.

Vínculo

En la carga de un programa a través del softkey "Seleccionar", se vinculan todos los ficheros y listas de datos necesarios para formar un programa. Al vincular, se comprueba si todos los módulos existen, se han analizado y están libres de errores. Además, el vínculo comprueba cuando se efectúa una transferencia de parámetros, la compatibilidad de los tipos entre los parámetros transmitidos. Si durante este proceso aparece algún error, del mismo modo que en el caso de la compilación, se genera un fichero de error, con la extensión ".ERR".



Ud. también puede escribir un programa KRL con cualquier editor de textos usual, y a continuación cargarlo en la memoria del sistema con ayuda del softkey "Cargar". En este caso, debe prestar atención que se lleven a cabo todas las inicializaciones necesarias (por ejemplo, velocidades de los ejes).

A continuación se presenta un ejemplo sencillo de programa para la definición de velocidades y aceleraciones de ejes:



DEF PROG1()

;----- Sección de declaraciones -----
INT J

;----- Sección de instrucciones -----

\$VEL_AXIS[1]=100 ;Especificación de las velocidades de ejes
\$VEL_AXIS[2]=100
\$VEL_AXIS[3]=100
\$VEL_AXIS[4]=100
\$VEL_AXIS[5]=100
\$VEL_AXIS[6]=100

\$ACC_AXIS[1]=100 ;Especificación de las aceleraciones de ejes
\$ACC_AXIS[2]=100
\$ACC_AXIS[3]=100
\$ACC_AXIS[4]=100
\$ACC_AXIS[5]=100
\$ACC_AXIS[6]=100

PTP {A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}

FOR J=1 TO 5
 PTP {A1 4}
 PTP {A2 -7,A3 5}
 PTP {A1 0,A2 -9,A3 9}
ENDFOR

PTP {A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}
END

1.3 Modificación de programas

Básicamente, existen dos posibilidades para modificar un programa en la superficie de operación del nivel del experto:

- Corrección de programa (PROKOR)
- Editor

1.3.1 Corrección de programa

Corrección de programa es el método normal. Si se selecciona un programa o se detiene un programa en curso, automáticamente se encuentra en el modo PROKOR.

Aquí pueden darse entradas o modificarse instrucciones por medio del formulario Inline o en código ASCII (en el nivel del experto) que únicamente afectan a **una** línea de programa. Por lo tanto, ninguna estructura de control (bucles, etc.) ni declaraciones de variables.



En el estado seleccionado, las entradas erróneas son borradas inmediatamente al abandonar la línea del programa, y aparece un mensaje de fallos en la ventana de mensajes.

1.3.2 Editor

Por ello, si se desean procesar o insertar determinadas instrucciones KRL o estructuras de programas, debe realizarse esto a través del editor. Dado que al cerrar el editor se compila el código completo, pueden identificarse también errores que se producen en relación con otras líneas (por ejemplo, variables declaradas incorrectamente).

1.3.2.1 Funciones de bloque



Las funciones de bloque se encuentran solamente a disposición en el editor a partir del nivel del "Experto". Abra Ud. un programa cuyo contenido desea modificar con las funciones de bloque, mediante la tecla del softkey "Editar". Como cambiar previamente al nivel del "Experto" se describe en la documentación **Configuración** en el capítulo **[Configurar el sistema]**, apartado **"Grupo de usuario"**.

Coloque primeramente el cursor intermitente del editor sobre el inicio o final de la parte de programa a ser desplazada. Mantenga entonces oprimida la tecla "Shift" en el teclado mientras que mueve el cursor hacia arriba o abajo. De esta manera marca una parte del programa que en el paso siguiente debe ser procesado con las funciones de bloque. La parte ahora marcada se reconoce por estar resaltada en color.

Procesar

Pulse la tecla del menú "Procesar", y del menú que se abre, seleccione la función deseada.



Si para las funciones de bloque se han de utilizar el bloque numérico y el teclado, la función NUM debe estar desactivada. Pulse para ello la tecla "Num" que se encuentra en el teclado. La indicación correspondiente en la línea de estado está desactivada.

1.3.2.2 Copiar (CTRL-C)

La sección marcada del programa es memorizada en el portapapeles para su posterior procesamiento. Esta sección puede ser insertada a continuación en otro lugar.

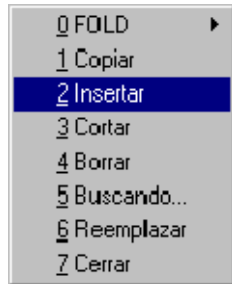




De forma alternativa, puede Ud. mantener pulsada la tecla CTRL sobre el teclado numérico, y pulsar adicionalmente la tecla C sobre el teclado. A continuación suelte ambas teclas.

1.3.2.3 Insertar (CTRL-V)

Coloque el cursor del editor ahora en el punto en el que deba ser insertada la parte de programa “copiada” o “cortada”.



Seleccione ahora la opción “Insertar”. El sector de programa antes marcado es insertado debajo del cursor de edición.



De forma alternativa, puede Ud. mantener pulsada la tecla CTRL sobre el teclado numérico, y pulsar adicionalmente la tecla V sobre el teclado. A continuación suelte ambas teclas.

1.3.2.4 Cortar (CTRL-X)

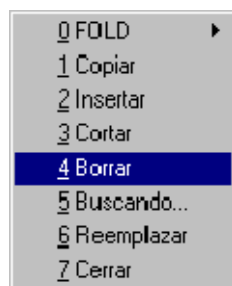
Si selecciona del menú la opción “Cortar”, la parte de programa marcada se guarda en la memoria intermedia y se borra del listado del programa.



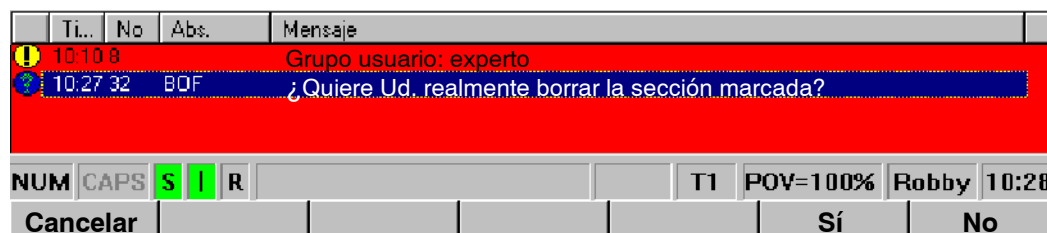
De forma alternativa, puede Ud. mantener pulsada la tecla CTRL sobre el teclado numérico, y pulsar adicionalmente la tecla X sobre el teclado. A continuación suelte ambas teclas.

1.3.2.5 Borrar

El sector marcado puede ser eliminado del programa. En este caso no se efectúa ninguna memorización en el portapapeles. Este sector recortado del programa queda definitivamente perdido.



Por este motivo, se efectúa una pregunta de seguridad dentro de la ventana de mensajes, a contestar a través de la barra de softkeys.

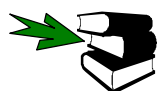


- **Cancelar** La acción “Borrar” es cancelada;
- **Sí** El sector marcado es borrado definitivamente;
- **No** La función “Borrar” es cancelada;



Del menú, seleccione la opción “Borrar”, y la sección marcada del programa es borrada del listado sin memorización intermedia en el portapapeles.

1.3.2.6 Buscando...



Informaciones más detalladas se encuentran en el **Manual de operación**, en la documentación **Programación por el usuario**, capítulo **[Confección y modificación de un programa]**, apartado **[Trabajar con el editor de programa]**.

1.3.2.7 Reemplazar

La función “Buscar y reemplazar” solo se tiene a disposición en el nivel del experto y allí, exclusivamente en el editor. Esta aplicación efectúa una búsqueda en el programa, de una secuencia de caracteres indicada, dentro de la parte visible (se excluyen líneas de folds o folds abiertas) permitiendo además el reemplazo por otra secuencia definida de caracteres.

Para ello seleccione del menú “Procesar” la opción “Reemplazar”.



Se abre la siguiente ventana:

Buscar	<input type="text"/>
Reemplaz	<input type="text"/>

```

1  DEF error( )
2  int i
3  INI{PE}%U3.2.0,%MKUKATPBASIS,%CINIT,%UCOMMON,%P
4  PTP HOME Vel= 100 % DEFAULT
5
6  FOR i=1 To 4
7  $OUT[I]=TRUE
8  END FOR
9
10 I=I+1
11
12 PTP HOME Vel= 100 % DEFAULT
13 END

```

KRC:\R1\ERROR.SRC Ln 11, Col 0

La barra de softkeys se modifica.

Buscar	reemplazar	Reemplazar todo				Cancelar
--------	------------	-----------------	--	--	--	----------

En la línea de búsqueda, indique una secuencia de caracteres, y cambie la posición hacia abajo a la línea de reemplazo con ayuda de la tecla de flecha. De entrada aquí a la secuencia de caracteres que ha de reemplazar los buscados.

Buscar	FOR
Reemplaz	IF

```

1  DEF ERROR( )
2  int i
3  INI
4  PTP HOME Vel= 100 % DEFAULT
5
6  FOR i=1 TO 4
7  $OUT[I]=TRUE
8  ENDFOR
9
10 I=I+1
11
12 PTP HOME Vel= 100 % DEFAULT
13 END

```

KRC:\R1\PROG_01.SRC Ln 11, Col 0

Buscar

reemplazar

Si la secuencia de caracteres aparece repetidas veces en el documento, pero Ud. quiere reemplazarla solamente una vez en una determinada posición, pulse la tecla del softkey "Buscar" tantas veces, hasta acceder a la posición buscada. A continuación pulse "Reemplazar". La secuencia de caracteres buscada es reemplazada por la secuencia de caracteres indicada.

Programación por el experto

Si Ud. desea reemplazar la secuencia de caracteres buscada en todas las posiciones o en una zona marcada anteriormente en el programa, entonces, después de la entrada en el formulario búsqueda/reemplazo arriba descrita, pulse la tecla del softkey “Reemplazar todas”.

Cancelar

Ud. recibe en la ventana de mensajes, el mensaje “Se ha encontrado la region especificada o marcada”. (Confirmación que el programa completo o la zona marcada fue revisada). Después de pulsar la tecla del softkey “Cancelar”, abandona Ud. el modo de reemplazo y en la ventana de mensajes se visualiza la cantidad de reemplazos efectuados desde el momento de activada la función.

	Ti...	Nº	Abs.	Mensaje
!	17:46 40	BOF		Se ha encontrado la region especificada o marcada.
!	17:47 39	BOF		2 reemplazamientos realizados

1.4 Esconder partes de un programa

De forma diferente a otros editores normales, el editor KCP permite una visualización de partes específicas del contenido de un programa. De esta manera, por ejemplo, el usuario ve solamente los contenidos esenciales de un programa, mientras que el experto tiene la oportunidad de contemplar el programa completo.

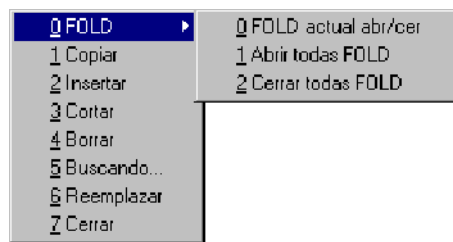
1.4.1 FOLD

La superficie de operación KUKA utiliza una técnica especial para una clara visualización de un programa. Instrucciones ejecutadas como comentarios KRL permiten suprimir la visualización de partes siguientes del programa. De esta manera, el programa se divide en apartados adecuados que se denominan de acuerdo a su carácter clasificador como "FOLDS".

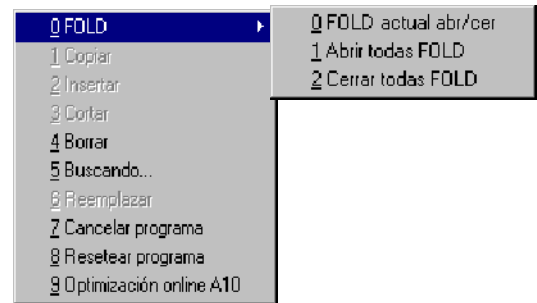
"FOLDS", de forma estándar, se encuentran "cerradas", y sólo pueden ser "abiertas" en el nivel del experto. Contienen informaciones que para el usuario, sobre la superficie de operación de KUKA (KUKA-BOF) son invisibles. En el nivel del experto Ud. tiene la posibilidad de hacer invisible para el usuario un bloque KRL. Para ello, se deben encerrar las correspondientes declaraciones o instrucciones por medio de las denominaciones "**;FOLD**" y "**;END-FOLD**".

Procesar

En un programa, folds pueden ser presentadas en pantalla o escondidas, pulsando la tecla del menú "Procesar" y a continuación seleccionando "FOLD" y el comando deseado.



Programa en el editor



Programa seleccionado

Se dispone de las siguientes opciones a seleccionar:

- **FOLD actual abr/cer** abre o cierra la FOLD en la línea en la que se encuentra el editor de cursor
- **Abrir todas FOLD** abre todas las FOLDS del programa
- **Cerrar todas FOLD** cierra todas las FOLDS del programa



Si se resetea un programa seleccionado, en el cual se han abierto Folds, entonces éstas se cierran automáticamente.

De la secuencia...

```
;FOLD RESET OUT

FOR I=1 TO 16
    $OUT[I]=FALSE
ENDFOR

;ENDFOLD
```

...de una fold cerrada, se visualiza en la superficie de operación solamente las palabras **"RESET OUT"**. Con esta instrucción se puede, por ejemplo, hacer invisible al usuario la sección de declaraciones e inicialización.

1.4.1.1 Programa de ejemplo



```
DEF FOLDS()

;FOLD DECLARATION;% informaciones adicionales
;----- Sección de declaraciones -----
EXT BAS (BAS_COMMAND :IN,REAL :IN )
DECL AXIS HOME
INT I
;ENDFOLD

;FOLD INITIALISATION
;----- Inicialización -----
INTERRUPT DECL 3 WHEN $STOPMESS==TRUE DO IR_STOPM ( )
INTERRUPT ON 3
BAS (#INITMOV,0 ) ;inicialización de las velocidades,
                  ;aceleraciones, $BASE, $TOOL, etc.
FOR I=1 TO 16
    $OUT[I]=FALSE
ENDFOR
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 30,A6 0}
;ENDFOLD

;----- Sección principal -----
PTP HOME ;desplazamiento COI
LIN {X 540,Y 630,Z 1500,A 0,B 90,C 0}
PTP HOME

END
```

En la superficie de operación se ve el programa del ejemplo de la siguiente manera:

```
1
2  DECLARATION
3
4  INITIALISATION
5
6  ;-----Sección principal-----
```

El mismo programa con folds abiertas:

```

1
2  DECLARATION
3  ;----- Sección de declaraciones -----
4  EXT BAS (BAS_COMMAND :IN,REAL :IN )
5  DECL AXIS HOME
6  INT I
7
8  INITIALISATION
9  ;----- Inicialización -----
10 INTERRUPT DECL 3 WHEN $STOPMESS==TRUE DO IR_STOPM ( )
11 INTERRUPT ON 3
12 BAS (#INITNOV,0 ) ; Inicialización de velocidades;
13 ; aceleraciones , $BASE, $TOOL, etc.
14 FOR I=1 TO 16
15 $OUT[I]=FALSE
16 ENDFOR
17 HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}
18
19 ;----- Sección principal -----

```

Con FOLD cerrada, sólo se ve la expresión detrás de la palabra clave "FOLD". Contrariamente a esto, con FOLD abierta, se visualizan todas las instrucciones y declaraciones.








"FOLD" es solamente una instrucción para el editor. El compilador interpreta las instrucciones FOLD debido al signo de punto y coma colocado delante, como un comentario normal.

1.5 Modos de ejecución del programa

Los modos de ejecución de un programa especifican si el procesamiento del mismo debe realizarse

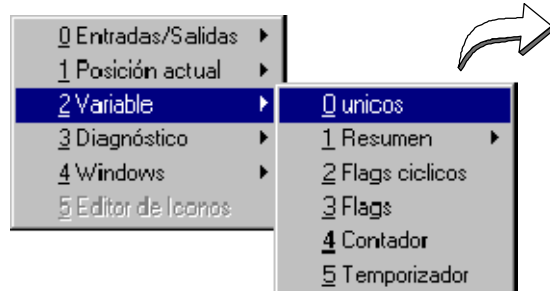
- sin detención del programa
- por pasos de movimiento
- por pasos individuales (línea).

En la siguiente tabla se describen todos los tipos de ejecución de programas.

Modo de ejecución	Descripción
GO 	Todas las instrucciones se procesan en el programa sin PARADA hasta el final del programa.
MSTEP 	Motion Step (paso de movimiento) El programa se ejecuta de a pasos, es decir, con una PARADA después de cada paso de movimiento. El programa es ejecutado sin procesamiento en avance.
ISTEP 	Paso incremental (paso individual) El programa se ejecuta por pasos programados, es decir, con una PARADA después de cada instrucción (también líneas en blanco). El programa es ejecutado sin procesamiento en avance.
PSTEP 	Program Step (paso de programa) Subprogramas son ejecutados completamente. El programa es ejecutado sin procesamiento en avance.
CSTEP 	Continuous Step (paso de movimiento) El programa se ejecuta de a pasos, es decir, con una PARADA después de cada paso de movimiento con parada exacta. El programa se ejecuta con procesamiento en avance, es decir, la ejecución se efectúa con posicionamiento aproximado.

Los modos de ejecución de programas GO, MSTEP y ISTEP pueden seleccionarse en el KCP a través de una función de estado o por medio de la variable "\$PRO_MODE". En cambio, PSTEP y CSTEP pueden ser declarados solamente a través de la variable "\$PRO_MODE". Para la modificación de cada estado en particular de esta variable, llame la función en el menú "Indicación" -> "Variable" -> "Únicos". A continuación, en el campo de entrada "Nombre" indique la variable "\$PRO_MODE" y en el campo "Valor nuevo" el valor deseado.

Indicación



Nombre:

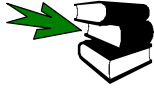
Valor actual:

Valor nuevo:

Módulo:



Los modos de ejecución de programa “#PSTEP” y “#CSTEP” sólo pueden seleccionarse a través de la corrección de variables y no por medio de funciones de estado.



Informaciones adicionales se encuentran en el capítulo **[Variables y declaraciones]**, en el apartado **[Objetos de datos]** bajo **(Tipos de enumeración)**.

1.6 Tratamiento de errores



Si se produce un error al compilar o al vincular, en la ventana de mensajes se emite un mensaje de error y en el navegador se marca el fichero con el error.



Como ejemplo sirve el fichero "ERROR.SRC", que fue creada (con errores):

```



1  DEF  ERROR ( )
2  INI
3  PTP HOME  Vel= 100 % DEFAULT
4
5  FOR I=1 TO 4
6  $OUT[I]=TRUE
7  ENDFOR
8
9  I == I + 1
10
11 PTP HOME  Vel= 100 % DEFAULT
12 END

```

Cerrado el editor, aparece ahora en la ventana de mensajes una indicación acerca de la cantidad de errores.

	Ti..	No	Abs.	Mensaje
!	14:49 8		UserMode	Grupo usuario: Experto
!	14:51 1421	UMP		/R1/ERROR : 3 Error de compilación

Al mismo tiempo, durante este procedimiento, los ficheros afectados son marcados con una cruz roja.

Nombre	Ext...	Comentario	Atributos	Ta
 error	src		-a-- R/V---E-	3 k
 error	dat		-a-- R/V---E-	2 k

Ud. dispone de la siguiente barra de softkeys:

Nuevo	Editar ERR	Abrir	Editar DAT	Borrar		
-------	------------	-------	------------	--------	--	--

El softkey "Abrir" carga el fichero en el editor, y al pulsar la tecla del softkey "EditarDat" (Lista de datos) se abre con el editor el fichero de extensión Dat. Si Ud. quiere borrar los ficheros erróneos, entonces pulse "Borrar" y a través de "Nuevo" puede Ud. crear un fichero nuevo.

Editar ERR

Pulsando la tecla del softkey “Editar ERR” (Lista de fallos), se abre la misma.

ErrorView(error.SRC)				Línea de título con el nombre del fichero
Lí...	Col.	Error ...	Descripción	
52	5	2263	TIPO DE VARI...	Breve descripción
53	6	2249	EXPRESION D...	
56	6	2309	SE ESPERA "("	
*1				Número de fallo
				Número de línea y columna errónea
FOR I=1 TO 4				Línea del texto fuente errónea
TIPO DE VARIABLE DE BUCLE DIFERENTE A				Descripción del fallo = Breve descripción
INT				

Con ello cambia la barra de softkeys:

				->Editor	Actualiza	Cerrar
--	--	--	--	----------	-----------	--------



OBSERVACION

- *1 Los números indicados de las líneas corresponden a la enumeración absoluta de las líneas en el programa, del mismo modo que se efectuaría la indicación en un editor ASCII normal. Para conseguir una concordancia entre el número de línea en el protocolo de errores y el número de línea en el KCP, deberían estar abiertas todas las Folds, “Visualización Detalles” y la “DEF Línea” deberían estar activas. Pero esta representación es algo confusa, dado que pone a disposición todas las informaciones, a pesar de no necesitarlas. Informaciones adicionales sobre la vista en detalle y la línea DEF se encuentran en el apartado **[Esconder partes de un programa]**.

Del protocolo de errores puede verse que han aparecido los siguientes fallos:

- 3 líneas en el fichero SRC contienen errores;
- los números de las líneas con errores son 51, 52 y 55;
- en la línea 51, los números de error
 - 2263: tipo de variable de bucle diferente a INT
- en la línea 52 el número de error
 - 2249: expresión diferente a INT
- en la línea 55 el mensaje de error
 - 2309: se espera “(“

De los mensajes de error “2263” puede deducirse rápidamente, que la variable *I* no fue declarada como entera. El mensaje de fallo 2249 es un resultado también de la declaración faltante, dado que en un bucle de conteo el contador siempre debe ser del tipo INT. El mensaje “2309” significa: el compilador interpreta la línea como un llamado de subprograma, pero en donde faltan los paréntesis.



El significado de los números de error puede ser presentado en pantalla en línea (online) con ayuda de la función del menú "Indicación" -> "Variable" -> "Unicos". Para ello, indique en la ventana de estado en el campo de entrada "Nombre" el signo "&" seguido del número indicativo de error. En este caso, por ejemplo "&2263" y pulse la tecla de entrada.

Indicación

0 Entradas/Salidas ▶

1 Posición actual ▶

2 Variable ▶

3 Diagnóstico ▶

4 Windows ▶

5 Editor de Iconos

0 unicos

1 Resumen ▶

2 Flags ciclicos

3 Flags

4 Contador

5 Temporizador

Nombre:

Valor actual:

Valor nuevo:

Módulo:

->Editor

Si ahora Ud. carga el fichero SRC (en este caso "ERROR.SRC") en el editor, puede efectuar las modificaciones correspondientes. Para la facilitación, el cursor se posiciona en forma intermitente sobre la primera línea errónea. Preste atención, que la visibilidad limitada esté desactivada, y también que sea visible la línea DEF. Explicaciones más detalladas se encuentran en el apartado **[Esconder partes de un programa]**.

En el siguiente ejemplo las Folds no necesitan ser abiertas. Pero si se desea hacerlo, utilice la instrucción del menú "Procesar" -> "FOLD" -> "Abrir todas FOLD".



La línea faltante "INT I" en el programa creado originalmente debe ser insertada **delante** de la línea "INI". Esto sólo es posible, cuando se tiene visible la línea "DEF ERROR ()".

Entonces, inserte la línea

INT I

delante de la línea INI, y elimine en la línea 10 un signo de igualdad.

I = I + 1

<pre> 1 DEF ERROR() 2 INT I 3 INI 4 PTP HOME Vel= 100 % DEFAULT 5 6 FOR I=1 TO 4 7 \$OUT[I]=TRUE 8 ENDFOR 9 10 I=I+1 11 12 PTP HOME Vel= 100 % DEFAULT 13 END </pre>	<p>Está línea insertarla aquí</p> <p>Eliminar un signo de igualdad</p>
--	--

Actualiza

Después de haber cerrado el editor y memorizado el fichero corregido, puede Ud. con la lista de fallos, pulsar la tecla del softkey "Actualiza", y la lista de fallos desaparece cuando todos los fallos han sido subsanados.

1.7 Comentarios

Los comentarios son una componente importante de cada programa de computación. Con ellos puede Ud. clarificarse la estructura de un programa y hacerla comprensible a terceros. Los comentarios no influyen en la velocidad de procesamiento del programa.

Los comentarios pueden ser colocados en cualquier parte de un programa. Se inician siempre por medio del signo punto y coma “;”, por ej.:

```
...
PTP P1                ;Desplazamiento al punto de partida
...
;--- Resetear salidas ---
FOR I = 1 TO 16
    $OUT[I] = FALSE
ENDFOR
...
```


2 Variables y declaraciones

2.1 Variables y nombres

Junto al uso de constantes, es decir, la indicación directa del valor en forma de números, símbolos, etc., pueden utilizarse en KRL también variables y otras formas de datos en el programa.

Para la programación de robots industriales son necesarias variables, por ejemplo, para el procesamiento del sensor. Con ellas puede memorizarse el valor leído por el sensor y evaluarse en diferentes lugares del programa. Además, pueden llevarse a cabo operaciones aritméticas para calcular, por ejemplo, una nueva posición.

Una variable se representa en el programa mediante un nombre, donde la denominación del nombre puede seleccionarse libremente con determinados límites.

Nombres

Nombres en KRL

- deben tener una longitud máxima de 24 caracteres,
- deben contener letras (A–Z), cifras (0–9) así como los símbolos “_” y “\$”,
- no deben empezar con cifras,
- no deben ser palabras clave.



Dado que todas las variables del sistema (véase el apartado 2.4) comienzan con el símbolo “\$”, no debe utilizar este símbolo como primer carácter en los nombres definidos por el usuario.

Son nombres KRL válidos, por ejemplo

SENSOR_1

BOQUILLA13

P1_HASTA_P12

Una variable debe contemplarse como un área de memoria fija, a cuyo contenido puede accederse a través del nombre de variable. Por ello, la variable está realizada para el tiempo de procesamiento del programa mediante un lugar de memoria y un contenido de memoria (valor).

Asignación de valor

Con el símbolo de igualdad (=) se asignan los valores a las variables. La instrucción

CANTIDAD = 5

significa entonces, que en el área de memoria con la dirección de CANTIDAD se registra el valor 5. Cual sea exactamente el aspecto de la dirección, no es interesante para el programador, dado que la asigna automáticamente el compilador. Solamente es importante que el contenido de memoria pueda accederse en cualquier momento en el programa con la ayuda de su nombre.

Dado que los diferentes objetos de datos (véase el apartado 2.2) tienen también diferentes necesidades de lugar de memoria, el tipo de datos de una variable debe acordarse (declararse) antes de su uso (véase el apartado 2.2.1).

Duración

La duración de una variable es el tiempo durante el cual la variable tiene asignado un puesto en la memoria. Depende si la variable está declarada en un archivo SRC o en una lista de datos:

- **Variable declarada en un archivo SRC**

La duración se limita al tiempo de funcionamiento del programa. Después de finalizar el procesamiento, se libera de nuevo el lugar de la memoria. Con ello, se pierde el valor de la variable.

- **Variable declarada en una lista de datos (véase el apartado Listas de datos)**

La duración no depende del tiempo de funcionamiento del programa. La variable existe mientras exista la lista de datos. Dichas variables son, por lo tanto, permanentes. (Hasta la próxima conexión / desconexión).

2.2 Objetos de datos

Como objetos de datos deben entenderse unidades de memoria nombrables de un tipo de datos determinado. Las unidades de memoria pueden constar de múltiples células de memoria diferentes (bytes, palabras, etc.). Si el programador acuerda un objeto de datos de este tipo bajo un nombre, se obtiene una variable. La variable ocupa a continuación una o varias células de memoria en las que el programa puede grabar y leer datos. Mediante la denominación simbólica de las células de memoria con nombres libremente seleccionables, la programación se hace más fácil y clara, y el programa puede leerse mejor.

Para la aclaración del concepto tipo de datos, puede servir el ejemplo siguiente: en una célula de memoria con 8 bits se encuentra la combinación de bits

00110101

¿Cómo debe interpretarse esta combinación de bits? ¿Se trata de una representación binaria del número 53 o del carácter ASCII "5", que se representa con la misma muestra de bits?

Tipo de datos

Para responder inequívocamente a esta pregunta, falta todavía una información importante, como es la indicación del tipo de datos de un objeto de datos. En el caso anterior podría ser el tipo "número entero" (INTEGER) o "carácter" (CHARACTER).

Junto a este motivo técnico de datos para la introducción de los distintos tipos de datos es también cómodo para el usuario el uso de tipos de datos, ya que puede trabajarse exactamente con aquellos que son especialmente indicados para la aplicación en particular.

2.2.1 Declaración e inicialización de objetos de datos

DECL

La asignación de un nombre de variable a un tipo de datos y la reserva del lugar de memoria se produce en KRL con la ayuda de la declaración DECL. Con

```
DECL INT CANTIDAD, NUMERO
```

identifica, por ejemplo, dos variables CANTIDAD y NUMERO del tipo de datos "número entero" (INTEGER).

Con ello, el compilador conoce estas dos variables, así como el tipo de datos correspondiente y puede comprobar si utilizando estas variables, este tipo de datos permite la operación prevista.

La declaración comienza, como se muestra en el ejemplo, con la palabra clave DECL, seguida por el tipo de datos y la lista de variables que deben contener este tipo de datos.



En la declaración de variables y campos de un tipo de datos predefinido, puede omitirse la palabra clave DECL. Junto a los tipos de datos simples INT, REAL, CHAR y BOOL (véase el apartado 2.2.2) están predefinidos, entre otros, los tipos de datos de estructura POS, E6POS, FRAME, AXIS y E6AXIS (véase el apartado 2.2.5).

Para las variables (¡ningún campo!) del tipo de datos POS, puede suprimirse la declaración completa. El tipo de datos POS vale como tipo de datos estándar para las variables.

La palabra clave DECL es imprescindible para la declaración de los tipos de estructura o de enumeración libremente definibles (véase los apartados 2.2.5 y 2.2.6).

Inicialización

Después de la declaración de una variable, su valor se define en primer lugar como no válido, ya que de lo contrario dependería de la ocupación ocasional de memoria. Para poder trabajar con la variable, debe prefijarse un determinado valor. Esta primera asignación de un valor a una variable, se denomina inicialización.



Para la creación de nuevos archivos a través del softkey "Nuevo", en la superficie de operación KUKA se genera también automáticamente, una secuencia INI. La declaración de las variables debe efectuarse siempre antes de esta secuencia.

La asignación de un valor a una variable es una instrucción, y por ello, no debe estar en la sección de declaraciones. Sin embargo, la inicialización puede efectuarse en cualquier momento en la sección de instrucciones. Pero todas las variables acordadas deben inicializarse convenientemente en una sección de inicialización directamente después de la sección de declaración (véase Fig. 1).

Solamente en listas de datos se admite inicializar variables directamente en la línea de declaraciones.

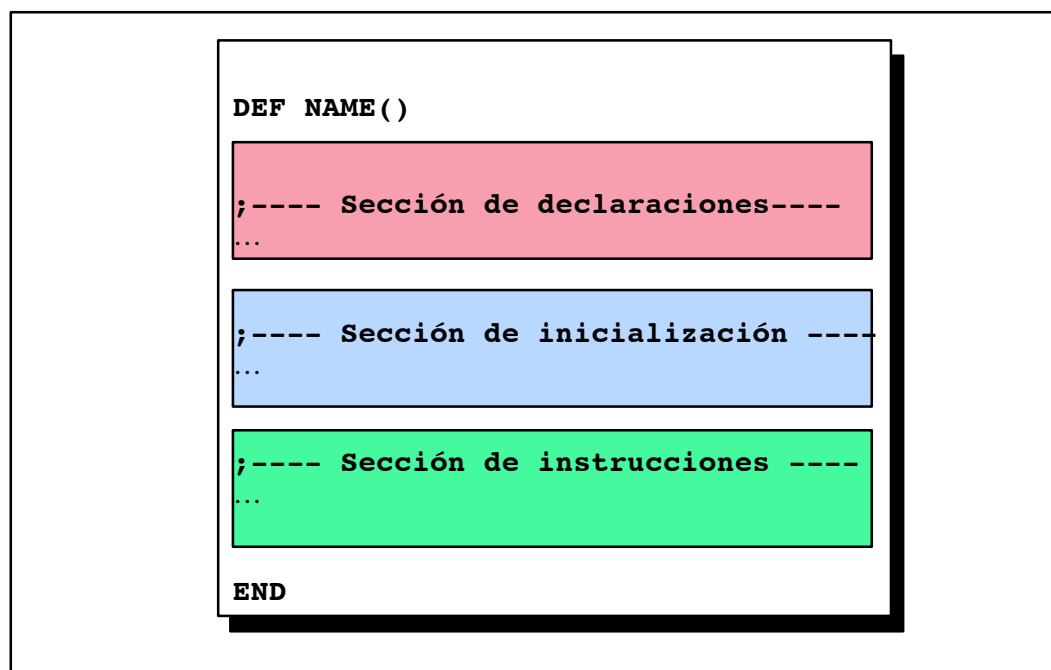
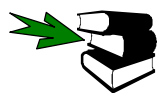


Fig. 1 Estructura básica de un programa del robot



Informaciones adicionales se encuentran en el capítulo **[Listas de datos]**.

2.2.2 Tipos de datos simples

Entre los tipos de datos simples se entienden algunos tipos de datos básicos que existen en la mayoría de lenguajes de programación. Los tipos de datos simples, al contrario que los tipos de datos de estructura (véase el apartado 2.2.3–2.2.6) solamente contienen un único valor. Los tipos de datos conocidos en KRL se presentan en una lista junto con su rango de valores correspondiente en la Tab. 1.

Tipo de datos	Entero	Real	Booleano	Carácter
Palabra clave	INT	REAL	BOOL	CHAR
Significado	número entero	número de coma flotante	estado lógico	1 carácter
Rango de valores	$-2^{31} \dots 2^{31}-1$	$\pm 1.1\text{E}-38 \dots \pm 3.4\text{E}+38$	TRUE, FALSE	carácter ASCII

Tab. 1 Tipo de datos simple

INT

El tipo de datos Integer (entero) es una cantidad parcial de la cantidad de números enteros. Por lo tanto, solamente puede existir una cantidad parcial cuando ninguna calculadora puede representar la cantidad teóricamente infinita de números enteros. Los 32 bits previstos para los tipos enteros en KRL proporcionan, por lo tanto, 2^{31} números enteros más el signo correspondiente. El número 0 se cuenta entre los números positivos.

Con

```
NUMERO = -23456
```

se asigna a la variable NUMERO el valor -23456.

Si asigna un valor REAL a una variable INTEGER, el valor se redondea según las normas generales (x.0 hasta x.49 redondeado hacia abajo, x.5 hasta x.99 redondeado hacia arriba). Mediante la instrucción

```
NUMERO = 45.78
```

la variable INTEGER NUMERO obtiene el valor 46.



Excepción: en la división de enteros, se corta la posición decimal, por ejemplo:
 $7/4 = 1$

Sistema
binario
Sistema
hexadecimal

Mientras que el ser humano calcula y piensa en el sistema decimal, un ordenador solamente reconoce unos y ceros, que representan los dos estados Desactivado y Activado. Un estado (desconectado o conectado) se representa así con un bit. Por motivos de velocidad, el ordenador accede, en general, a un paquete completo de ceros y unos de este tipo. Los tamaños normales de paquete son 8 bits (=1 byte), 16 bits o 32 bits. Por ello, en las operaciones al nivel de máquina, con frecuencia es una ayuda la representación en sistema binario (sistema de 2 dígitos: cifras 0 y 1) o en sistema hexadecimal (sistema de 16 dígitos: cifras 0–9 y A–F). Los valores enteros binarios o hexadecimales puede indicarlos en KRL utilizando comilla simple (') y una B que indica representación binaria o H para hexadecimal.

D	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
H	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10

Tab. 2 Los primeros 17 números en sistema decimal y hexadecimal

El número 90 puede asignarlo por tanto, en KRL, a tres tipos diferentes de una variable entera:

```
NUMENT = 90           ;Sistema decimal
NUMENT = 'B1011010'   ;Sistema binario
NUMENT = 'H5A'         ;Sistema hexadecimal
```

Bin → Dec

La conversión de los números binarios al sistema decimal se lleva a cabo del modo siguiente:

1	0	1	1	0	1	0	$= 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 90$
2^6	2^5	2^4	2^3	2^2	2^1	2^0	

Hex → Dec

Para transferir los números desde el sistema hexadecimal al sistema decimal, proceda del modo siguiente:

5	A	$= 5 \cdot 16^1 + 10 \cdot 16^0 = 90$
16^1	16^0	

REAL

En el concepto de la representación de coma flotante se trata de la separación de un número en mantisa y exponente y su representación de forma normalizada. Así se representa, por ejemplo:

```
5.3      como  0.53000000 E+01
-100     como -0.10000000 E+03
0.0513   como  0.51300000 E-01
```

Al calcular con valores reales, debido a la cantidad limitada de posiciones de coma flotante y la inexactitud que conlleva, debe tenerse en cuenta que las leyes algebraicas habituales ya no son válidas en todos los casos. Así, por ejemplo, en el álgebra vale:

$$\frac{1}{3} \times 3 = 1$$

Si se deja calcular al ordenador, puede obtenerse que el resultado es sólo 0.99999999 E+00. Una comparación lógica de este número con el número 1 proporcionaría el valor FALSE. Sin embargo, para las aplicaciones prácticas en el campo de los robots en general suele ser suficiente esta exactitud, si se tiene en cuenta que la prueba lógica solamente puede llevarse a cabo convenientemente en igualdades con magnitudes reales, dentro de un pequeño rango de tolerancia.

Asignaciones admisibles en variables reales son, por ejemplo:

```
NUMREAL1 = -13.653
NUMREAL2 = 10
NUMREAL3 = 34.56 E-12
```



Si se asigna un valor ENTERO (INTEGER) a una variable REAL, se lleva a cabo una conversión de tipo automática según REAL. La variable NUMREAL2 tiene también el valor 10.0 según la asignación superior.

BOOL

Las variables booleanas sirven para la descripción de los estados lógicos (por ejemplo, entradas/salidas). Solamente pueden asumir los valores TRUE (verdadero) y FALSE (falso):

```
ESTADO1 = TRUE
ESTADO2 = FALSE
```

CHAR

Las variables de carácter pueden representar exactamente 1 carácter del juego de caracteres ASCII. Para la asignación de un carácter ASCII a una variable CHAR, el carácter asignado debe estar incluido en las comillas ("):

```
CARACTER1 = "G"
```

```
CARACTER2 = "?"
```

Para guardar todas las series de caracteres, véase el apartado 2.2.4.

2.2.3 Campos

Se entienden como campos los agrupamientos de objetos del mismo tipo de datos para formar un objeto de datos, donde los componentes individuales de un campo pueden accederse a través de índices. Mediante la declaración

```
DECL INT OTTO[ 7 ]
```

Índice
campo

de

puede archivar, por ejemplo, 7 números enteros diferentes en el campo OTTO[]. A cada componente individual del campo puede accederse indicando el índice correspondiente (el primer índice es siempre el 1):

```
OTTO[ 1 ] = 5 ; al 1º elemento se le asigna el número 5
OTTO[ 2 ] = 10 ; al 2º elemento se le asigna el número 10
OTTO[ 3 ] = 15 ; al 3º elemento se le asigna el número 15
OTTO[ 4 ] = 20 ; al 4º elemento se le asigna el número 20
OTTO[ 5 ] = 25 ; al 5º elemento se le asigna el número 25
OTTO[ 6 ] = 30 ; al 6º elemento se le asigna el número 30
OTTO[ 7 ] = 35 ; al 7º elemento se le asigna el número 35
```

Gráficamente, el campo con el nombre OTTO[] puede presentarse como una estantería con 7 compartimentos. Ahora, la ocupación de los compartimentos, según las asignaciones anteriores, ofrecería el aspecto siguiente:

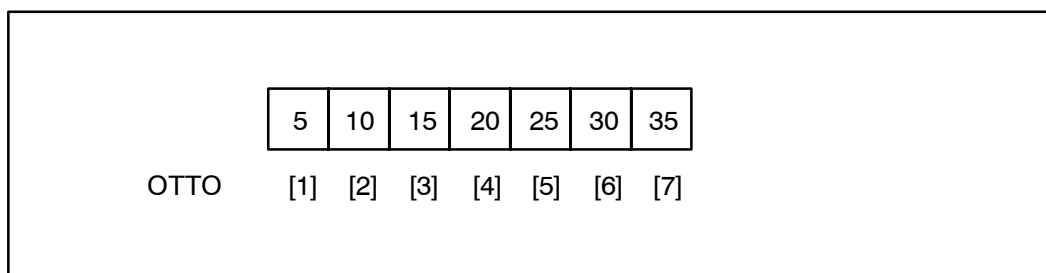


Fig. 2 Representación de un campo monodimensional

Ahora, si deben inicializarse todos los elementos de un campo con el mismo número, por ejemplo 0, no se tiene que programar explícitamente cada asignación, sino que se puede "automatizar" la preasignación con la ayuda de un bucle y de una variable numérica:

```
FOR I = 1 TO 7
    OTTO[ I ] = 0
ENDFOR
```



Informaciones adicionales se encuentran en el capítulo **[Control de ejecución del programa]**, apartado **[Bucles]**.

La variable de conteo es en este caso la variable entera I. Debe estar declarada antes de su utilización como entero.



- El tipo de datos de un campo es opcional. Con ello, los elementos individuales pueden constar a su vez de una composición de tipos de datos (por ejemplo: campo de campos)
- Para el índice solamente se admiten tipos de datos enteros
- Junto a las constantes y variables, se admiten también para el índice, expresiones aritméticas (véase apartado 2.3.1)
- El índice cuenta siempre desde 1

Bidimensional

Junto a los campos unidimensionales tratados hasta ahora, es decir, campos con un solo índice, en KRL puede utilizarse también campos bidimensionales o tridimensionales. Con

```
DECL REAL MATRIX[ 5, 4 ]
```

declara un campo bidimensional 5×4 con $5 \times 4 = 20$ elementos del tipo REAL. Gráficamente, este campo puede representarse como una matriz con 5 columnas y 4 filas. Con la secuencia de programa

```
I[ 3 ] = 0
FOR FILA = 1 TO 5
  FOR COLUMNA = 1 TO 4
    I[ 3 ] = I[ 3 ] + 1
    MATRIX[FILA,COLUMNA] = I[ 3 ]
  ENDFOR
ENDFOR
```

los elementos de la matriz se ocupan con un valor según el orden de su asignación. Por ello, se obtiene la siguiente ocupación de matriz:

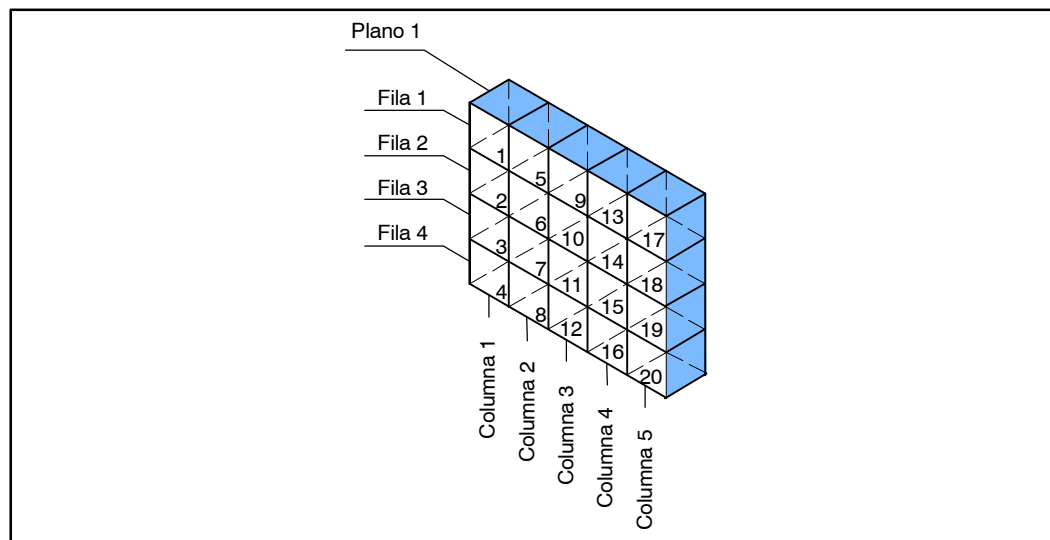


Fig. 3 Representación de un campo bidimensional

Tridimensional

Finalmente, los campos tridimensionales pueden representarse como varias matrices bidimensionales unas tras otras. La tercera dimensión indica, por así decirlo, el nivel en el que está la matriz (véase Fig. 4). Un campo tridimensional se declara de forma análoga a los campos unidimensionales o bidimensionales, por ejemplo:

```
DECL BOOL FELD_3D[ 3, 5, 4 ]
```

La secuencia de inicialización podría tener el aspecto siguiente:

```
FOR PLANO = 1 TO 3
  FOR COLUMNA = 1 TO 5
    FOR FILA = 1 TO 4
      CAMPO_3D[ PLANO, COLUMNA, FILA ] = FALSE
    ENDFOR
  ENDFOR
ENDFOR
```

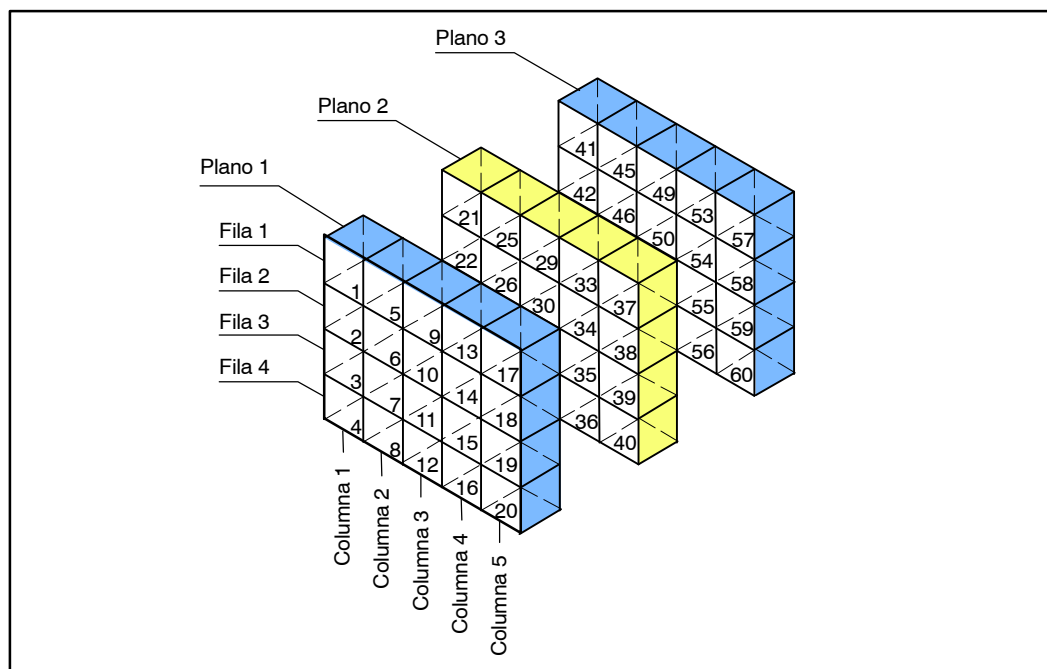


Fig. 4 Representación de un campo tridimensional

2.2.4 Cadenas de caracteres

Con el tipo de datos `CHAR`, como se ha descrito, solamente pueden guardarse caracteres individuales. Para utilizar cadenas enteras de caracteres, por ejemplo, palabras, simplemente se define un campo unidimensional del tipo `CHAR`:

```
DECL CHAR NAME[ 8 ]
```

Como en todos los otros casos, Ud. puede acceder a cada elemento individual del campo `NAME[]`, por ejemplo:

```
NAME[ 3 ] = "G"
```

		G					
--	--	---	--	--	--	--	--

Puede introducir también cadenas enteras de caracteres:

```
NAME[ ] = "ABCDEFGG"
```

ocupa los siete primeros elementos del campo `NAME[]` con las letras A, B, C, D, E, F y G:

A	B	C	D	E	F	G	
---	---	---	---	---	---	---	--

2.2.5 Estructuras

STRUC

Si se tienen que agrupar diferentes tipos de datos, el campo no es apropiado y debe recurrirse a la forma general de la composición. Con la instrucción de declaración `STRUC` pueden agruparse diferentes tipos de datos definidos previamente o tipos de datos predefinidos, para un nuevo tipo de datos compuesto. En especial, pueden formar parte también de un compuesto otros compuestos y campos.

El ejemplo típico para el uso de compuestos es el tipo de datos estándar `POS`. Consta de 6 valores `REAL` y 2 valores `INT` y se definió en el archivo `$OPERATE.SRC` del modo siguiente:

```
STRUC POS REAL X, Y, Z, A, B, C, INT S, T
```

Punto
separador

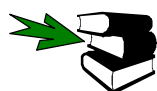
Por ejemplo, si ahora utiliza la variable `POSICION` del tipo de datos estructurales `POS`, así puede ocupar los elementos en forma individual con la ayuda del punto separador, por ejemplo:

```
POSICION.X = 34.4
POSICION.Y = -23.2
POSICION.Z = 100.0
POSICION.A = 90
POSICION.B = 29.5
POSICION.C = 3.5
POSICION.S = 2
POSICION.T = 6
```

Agregados

o conjuntamente, por medio de un denominado agregado

```
POSICION={X 34.4,Y -23.2,Z 100.0,A 90,B 29.5,C 3.5,S 2,T 6}
```



Informaciones adicionales se encuentran en el capítulo **[Variables y declaraciones]**, apartado **[Declaración e inicialización de objetos de datos]**.

Para los agregados se aplican las disposiciones siguientes:



- Los valores de un agregado pueden ser constantes simples o incluso agregados.
- En un agregado no tienen que indicarse todos los componentes de la estructura.
- Los componentes no necesitan indicarse en el orden en que se definieron.
- Cada componente debe estar incluido en un agregado una sola vez.
- En los campos de estructuras, un agregado describe el valor de un elemento de campo individual.
- Al comienzo de un agregado puede estar indicado el nombre del tipo de estructura, separado por un signo de dos puntos.

Las asignaciones siguientes son admisibles también, por ejemplo, para las variables POS:

```
POSICION={B 100.0,X 29.5,T 6}
POSICION={A 54.6,B -125.64,C 245.6}
POSICION={POS: X 230,Y 0.0,Z 342.5}
```

En los tipos de estructura POS, E6POS, AXIS, E6AXIS y FRAME no se modifican los componentes que faltan. En todos los demás agregados se definen como no válidos los componentes no existentes.

A través del ejemplo siguiente se explica el procedimiento de creación de algunas variables estructurales:

En un subprograma para la soldadura al arco, debe transferirse a una variable S_PARA la información siguiente:

REAL	V_HILO	Velocidad del hilo
INT	CARACT	Línea característica 0...100%
BOOL	ARCO	con/sin arco (para simulación)

La variable S_PARA debe constar de 3 elementos de diferentes tipos de datos. En primer lugar tiene que definirse un nuevo tipo de datos que cumpla con estos requisitos:

```
STRUC TIPOSOLD REAL V_HILO, INT CARACT, BOOL ARCO
```

Con ello se ha creado un nuevo tipo de datos con la denominación TIPOSOLD (TIPOSOLD no es ninguna variable). TIPOSOLD consta por lo menos de 3 componentes V_HILO, CARACT y ARCO. A continuación, puede declararse una variable opcional del nuevo tipo de datos, por ejemplo:

```
DECL TIPOSOLD S_PARA
```

De este modo ha creado una variable S_PARA del tipo de datos TIPOSOLD. A los elementos individuales se puede tener acceso – como descrito – con la ayuda del punto separador o del agregado:

```
S_PARA.V_HILO = 10.2
S_PARA.CARACT = 66
S_PARA.ARCO = TRUE
```

O

```
S_PARA = {V_HILO 10.2,CARACT 50, ARCO TRUE}
```



Para poder diferenciar mejor los tipos de datos autodefinidos de las variables, los nombres de los nuevos tipos de datos deberían empezar o finalizar con ...TIPO.

Estructuras predefinidas

En el archivo \$OPERATE.SRC existen ya predefinidas las estructuras siguientes:

```
STRUC AXIS      REAL  A1,A2,A3,A4,A5,A6
STRUC E6AXIS    REAL  A1,A2,A3,A4,A5,A6,E1,E2,E3,E4,E5,E6
STRUC FRAME     REAL  X,Y,Z,A,B,C
STRUC POS       REAL  X,Y,Z,A,B,C, INT S,T
STRUC E6POS     REAL  X,Y,Z,A,B,C,E1,E2,E3,E4,E5,E6, INT S,T
```

Los componentes A1...A6 de la estructura **AXIS** son valores angulares (ejes rotatorios) o valores de translación (ejes translatorios) para el desplazamiento específico de los ejes del robot 1...6.

Con los componentes adicionales E1...E6 en la estructura **E6AXIS**, puede tenerse acceso a los ejes adicionales.

En la estructura **FRAME** puede especificar 3 valores de posición en el espacio (X,Y y Z) así como 3 orientaciones en el espacio (A, B y C). Un punto en el espacio está definido de este modo unívocamente en posición y orientación.

Dado que existen robots que pueden desplazarse hasta uno y el mismo punto en el espacio con varias posiciones de eje, las variables enteras S y T en la estructura **POS** sirven para la determinación de una posición unívoca del eje.

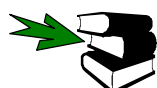


Informaciones adicionales se encuentran en el capítulo **[Programación de movimiento]**, apartado **[Instrucciones de movimiento]** Status (S) y Turn (T).

Con los componentes E1...E6 en el tipo de estructura **E6POS**, puede tenerse acceso nuevamente a los ejes adicionales.

Tipos de datos geométricos

Los tipos AXIS, E6AXIS, POS, E6POS y FRAME se denominan también tipos de datos geométricos, ya que el programador puede describir con ellos fácilmente relaciones geométricas.



Informaciones adicionales se encuentran en el capítulo **[Programación de movimiento]**, apartado **[Utilización de diferentes sistemas de coordenadas]**.

2.2.6 Tipos de enumeración

Un tipo de datos de enumeración es un tipo de datos que se compone de una cantidad limitada de constantes. Las constantes son nombres libremente definibles y pueden ser especificadas por el usuario. Una variable de este tipo (variable de enumeración) solamente puede aceptar el valor de una de estas constantes.

Como explicación sirve la variable de sistema \$MODE_OP. En ella se memoriza el tipo de modo de servicio seleccionado. Pueden seleccionarse los modos de servicio T1, T2, AUT y EX.

Podría declararse \$MODE_OP como variable entera, asignar un número a cada modo de servicio y memorizarlo en \$MODE_OP. Sin embargo, esto sería muy complicado.

ENUM

Una solución mucho más conveniente es la que ofrece el tipo de enumeración. En el archivo \$OPERATE.SRC se generó para ello un tipo de datos de enumeración con el nombre MODE_OP:

```
ENUM MODE_OP T1, T2, AUT, EX, INVALID
```

La instrucción para declaración de los tipos de enumeración se denomina ENUM. Las variables del tipo de enumeración MODE_OP solamente pueden aceptar los valores T1, T2, AUT, EX o INVALID. La declaración de las variables se efectúa nuevamente con la palabra clave DECL:

```
DECL MODE_OP $MODE_OP
```


– símbolo

La variable de enumeración `$MODE_OP` puede ocuparla ahora mediante asignación normal con uno de los cuatro valores del tipo de datos `MODE_OP`. Para diferenciarlas respecto a las constantes simples, delante de las constantes de enumeración que se definan, en las inicializaciones o consultas, se antepone un símbolo “#”, por ejemplo:

```
$MODE_OP = #T1
```

Con `ENUM` puede crear a continuación, opcionalmente, tantos tipos de datos de enumeración que quiera, y que Ud. mismo defina.

2.3 Manipulación de datos

Para la manipulación de diferentes objetos de datos, existen una gran cantidad de operadores y funciones, con cuya ayuda pueden crearse fórmulas. El poderío de un lenguaje de programación de robots depende, por igual, de los objetos de datos admitidos y de sus posibilidades de manipulación.

2.3.1 Operadores

Operando Deben entenderse como operadores, los operadores matemáticos normales, en contraposición a funciones tales, como por ejemplo, $\text{SIN}(30)$, que proporciona el seno del ángulo de 30° . En la operación $5+7$, se denominan como operandos el 5 y el 7 y el signo + es el operador.

En cada operación, el compilador comprueba la admisibilidad de los operandos. Así, por ejemplo, $7 - 3$ como sustracción de dos números enteros, es una operación admisible, $5 + "A"$ como adición de un valor entero a un carácter, es una operación no permitida.

En algunas operaciones, como $5 + 7.1$, es decir, la adición de valores enteros con reales, se lleva a cabo una adaptación del tipo, y el valor entero 5 se convierte en el valor real 5,0. Este problema se tratará más a fondo al ocuparnos de los operadores individuales.

2.3.1.1 Operadores aritméticos

Los operadores aritméticos corresponden a los tipos de datos INTEGER (ENTERO) y REAL. En KRL se admiten los 4 tipos de cálculo básico (véase Tab. 3).

Operador	Descripción
+	Adición o signo positivo
-	Sustracción o signo negativo
*	Multiplicación
/	División

Tab. 3 Operadores aritméticos

El resultado de una operación aritmética solamente es INT, cuando los dos operandos son del tipo INTEGER (entero). Si el resultado de una división entera no es entero, se desecha la parte decimal. Cuando, como mínimo, uno de los dos operandos es REAL, entonces también el resultado es de tipo REAL (véase Tab. 4).

Operandos	INT	REAL
INT	INT	REAL
REAL	REAL	REAL

Tab. 4 Resultado de una operación aritmética

Como explicación puede utilizarse el ejemplo de programa siguiente:



```

DEF ARITH()

;----- Sección de declaraciones -----
INT A,B,C
REAL K,L,M

;----- Inicialización -----
;antes de la inicialización, todas las variables son no válidas
A = 2           ;A=2
B = 9.8         ;B=10
C = 7/4         ;C=1
K = 3.5         ;K=3.5
L = 0.1 E01     ;L=1.0
M = 3           ;M=3.0

;----- Sección principal -----
A = A * C       ;A=2
B = B - 'HB'    ;B=-1
C = C + K       ;C=5
K = K * 10      ;K=35.0
L = 10 / 4      ;L=2
L = 10 / 4.0    ;L=2.5
L = 10 / 4.     ;L=2.5
L = 10./ 4      ;L=2.5
C = 10./ 4.     ;C=3
M = (10/3) * M  ;M=9.0

END

```

2.3.1.2 Operador geométrico

El operador geométrico se simboliza en KRL con el símbolo de dos puntos ":". Ejecuta un combinación lógica entre los tipos de datos **FRAME** y **POS**.

La combinación lógica entre dos frames es la transformación normal de sistemas de coordenadas. Por ello, la combinación lógica de una estructura **FRAME** con una estructura **POS** solamente actúa sobre el frame dentro de la estructura **POS**. Los componentes **S** y **T** no se ven afectados por la transformación y por ello tampoco deben estar ocupados con un valor. Sin embargo, los valores **X**, **Y**, **Z**, **A**, **B** y **C** deben estar ocupados siempre con un valor, tanto en los operandos **POS** como también en los operandos **FRAME**.

Combinación
lógica de
frames

Una combinación lógica de frames se evalúa de izquierda a derecha. El resultado tiene siempre el tipo de datos del operando que se encuentra en el extremo de la derecha (véase Tab. 5).

Operando izquierdo (SC referencia)	Operador	Operando derecho (SC destino)	Resultado
POS	:	POS	POS
POS	:	FRAME	FRAME
FRAME	:	POS	POS
FRAME	:	FRAME	FRAME

Tab. 5 Combinaciones de tipos de datos en el operador geométrico



Cuando el operando izquierdo tiene el tipo de datos POS, se produce una adaptación del tipo. La posición indicada por la estructura POS es convertida en un frame. Esto significa, que el sistema determina el frame de la herramienta para esta posición.

La forma de actuación del operador geométrico puede explicarse a través de un ejemplo simple (véase Fig. 5):

En una habitación hay una mesa. El sistema de coordenadas del ESPACIO está definido como sistema de coordenadas fijo en la esquina delantera izquierda de la habitación.

La mesa se encuentra paralela a las paredes de la habitación. La esquina delantera izquierda de la mesa se encuentra exactamente a 600 mm de la pared delantera y a 450 mm de la pared izquierda de la habitación. La mesa tiene 800 mm de altura.

Sobre la mesa hay una pieza de forma cuadrangular. Debe colocar el sistema de coordenadas de la PIEZA como se muestra en Fig. 5, en una esquina de la pieza. Para poder manipular convenientemente la pieza más tarde, el eje Z muestra del sistema de coordenadas de la PIEZA hacia abajo. La pieza está girada 40° respecto al eje Z del sistema de coordenadas MESA. La posición del sistema de coordenadas PIEZA respecto al sistema de coordenadas MESA es $X=80$ mm, $Y=110$ mm y $Z=55$ mm.

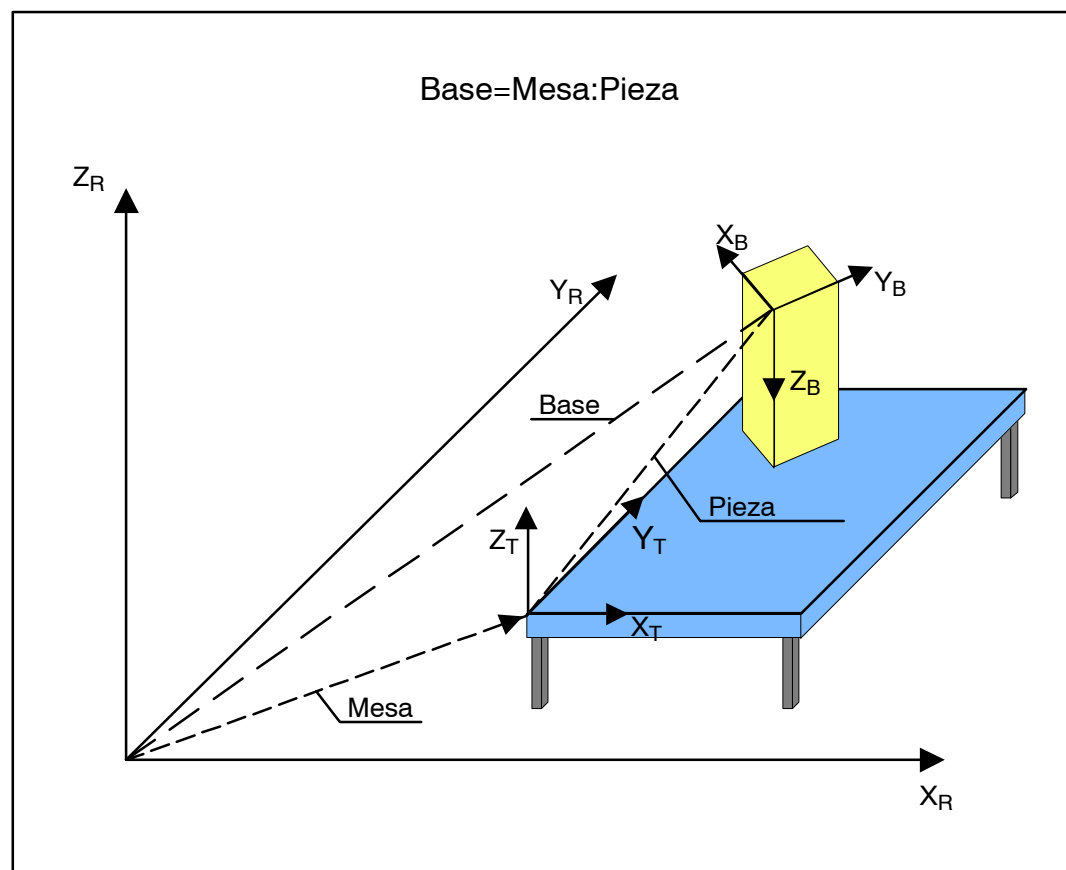


Fig. 5 Modo de actuación del operador geométrico

Ahora se describen las tareas a realizar del sistema de coordenadas PIEZA respecto al sistema de coordenadas ESPACIO. Para ello, debe declarar, en primer lugar, las variables de frame siguientes:

FRAME MESA, PIEZA, BASE

El sistema de coordenadas ESPACIO está ya especificado por el sistema. Los sistemas de coordenadas MESA y PIEZA se inician ahora de acuerdo con las condiciones de borde:

MESA = {X 450, Y 600, Z 800, A 0, B 0, C 0}

PIEZA = {X 80, Y 110, Z 55, A -40, B 180, C 0}

El sistema de coordenadas `PIEZA` respecto al sistema de coordenadas `ESPACIO` se obtiene ahora con la ayuda del operador geométrico, para

`BASE = MESA:PIEZA`

En nuestro caso, `BASE` está ocupado ahora del modo siguiente:

`BASE = {X 530,Y 710,Z 855,A 140,B 0,C -180}`

Otra posibilidad adicional sería:

`BASE = {X 530,Y 710,Z 855,A -40,B 180,C 0}`



Solamente en este caso especial se obtienen los componentes de `BASIS` como adición de los componentes de `MESA` y `PIEZA`. Esto se debe a que el sistema de coordenadas `MESA` no está girado respecto al sistema de coordenadas `ESPACIO`.

¡Sin embargo, en general, no resulta posible una adición simple de los componentes!

Una combinación lógica de frame tampoco es conmutativa, lo que significa que mediante el intercambio del frame de referencia y del frame destino, normalmente se modificará también el resultado.



Informaciones adicionales se encuentran en el capítulo **[Programación de movimiento]**, apartado **[Utilización de diferentes sistemas de coordenadas]**.

Para la aplicación del operador geométrico, otro ejemplo: en ello se desplazan diferentes sistemas de coordenadas y combinaciones lógicas de sistemas de coordenadas. Para aclarar las modificaciones de orientación, la punta de la herramienta desplaza en primer lugar en cada sistema de coordenadas una pieza en la dirección `X`, después una pieza en la dirección `Y` y finalmente una pieza en la dirección `Z`.



```

DEF  GEOM_OP ( );

----- Sección de declaraciones -----
EXT  BAS (BAS_COMMAND :IN,REAL :IN )
DECL AXIS HOME           ;Variable HOME del tipo AXIS
DECL FRAME MYBASE[2]     ;Campo del tipo FRAME

;----- Inicialización -----
BAS (#INITMOV,0 ) ;inicialización de las velocidades,
                  ;aceleraciones, $BASE, $TOOL, etc.
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 30,A6 0}; Activar Sistema
de coordenadas base
$BASE={X 1000,Y 0,Z 1000,A 0,B 0,C 0}REF_POS_X={X 100,Y 0,Z 0,A
0,B 0,C 0}           ;Pos. referencia
REF_POS_Y={X 100,Y 100,Z 0,A 0,B 0,C 0}
REF_POS_Z={X 100,Y 100,Z 100,A 0,B 0,C 0}; definir sistemas de
                                         coordenadas propios
MYBASE[1]={X 200,Y 100,Z 0,A 0,B 0,C 180}
MYBASE[2]={X 0,Y 200,Z 250,A 0,B 90,C 0}

;----- Sección principal -----
PTP  HOME ; desplazamiento COI; Movimiento referente al sistema
de coordenadas $BASE
PTP  $NULLFRAME           ;desplazarse directamente hasta el origen
de $BASE-SC
WAIT SEC 2                ;esperar 2 segundos
PTP  REF_POS_X             ;recorrer 100 mm en la dirección x
PTP  REF_POS_Y             ;recorrer 100 mm en la dirección y
PTP  REF_POS_Z             ;recorrer 100 mm en la dirección z
                           ;movimiento respecto del $BASE-SC
                           ;desplazado para MYBASE[1]

PTP  MYBASE[1]
WAIT SEC 2
PTP  MYBASE[1]:REF_POS_X
PTP  MYBASE[1]:REF_POS_Y
PTP  MYBASE[1]:REF_POS_Z; Movimiento respecto del $BASE-SC
desplazado para MYBASE[2]
PTP  MYBASE[2]
WAIT SEC 2
PTP  MYBASE[2]:REF_POS_X
PTP  MYBASE[2]:REF_POS_Y
PTP  MYBASE[2]:REF_POS_Z; Movimiento respecto del $BASE-SC
desplazado para MYBASE[1]:MYBASE[2]
PTP  MYBASE[1]:MYBASE[2]
WAIT SEC 2
PTP  MYBASE[1]:MYBASE[2]:REF_POS_X
PTP  MYBASE[1]:MYBASE[2]:REF_POS_Y
PTP  MYBASE[1]:MYBASE[2]:REF_POS_Z; Movimiento respecto del
$BASE-SC desplazado para MYBASE[2]:MYBASE[1]
PTP  MYBASE[2]:MYBASE[1]
WAIT SEC 2
PTP  MYBASE[2]:MYBASE[1]:REF_POS_X
PTP  MYBASE[2]:MYBASE[1]:REF_POS_Y
PTP  MYBASE[2]:MYBASE[1]:REF_POS_Z
PTP  HOME
END

```

2.3.1.3 Operadores de comparación

Con los operadores de comparación indicados en Tab. 6 pueden formarse expresiones lógicas. Por ello, el resultado de una comparación es siempre del tipo de datos `BOOL`, ya que una comparación solamente puede ser siempre verdadera (`TRUE`) o falsa (`FALSE`).

Operador	Descripción	Tipos de datos admisibles
<code>==</code>	igual	<code>INT</code> , <code>REAL</code> , <code>CHAR</code> , <code>ENUM</code> , <code>BOOL</code>
<code><></code>	desigual	<code>INT</code> , <code>REAL</code> , <code>CHAR</code> , <code>ENUM</code> , <code>BOOL</code>
<code>></code>	mayor que	<code>INT</code> , <code>REAL</code> , <code>CHAR</code> , <code>ENUM</code>
<code><</code>	menor que	<code>INT</code> , <code>REAL</code> , <code>CHAR</code> , <code>ENUM</code>
<code>>=</code>	mayor o igual que	<code>INT</code> , <code>REAL</code> , <code>CHAR</code> , <code>ENUM</code>
<code><=</code>	menor o igual que	<code>INT</code> , <code>REAL</code> , <code>CHAR</code> , <code>ENUM</code>

Tab. 6 Operadores de comparación

Las comparaciones pueden utilizarse en instrucciones de ejecución de programa (véase el capítulo 5), el resultado de una comparación puede asignarse a una variable booleana.

La prueba de la igualdad o desigualdad solamente tiene un sentido relativo para los valores reales, ya que debido al error de redondeo durante el cálculo de los valores a comparar, fórmulas algebraicamente idénticas pueden proporcionar valores desiguales (véase 2.2.2).



- Son posibles las combinaciones de operandos de `INT`, `REAL` y `CHAR`.
- Un tipo `ENUM` solamente puede compararse con el mismo tipo `ENUM`.
- Un tipo `BOOL` solamente puede compararse con un tipo `BOOL`.

Por lo tanto, es posible la comparación de valores numéricos (`INT`, `REAL`) con caracteres alfabéticos (`CHAR`), debido a que cada carácter ASCII tiene asignado un código ASCII. Este código es un número que determina la secuencia de los caracteres en el juego de caracteres.

Las constantes individuales de un tipo de enumeración, se numeran durante su declaración en el orden en que aparecen. Los operadores de comparación hacen referencia a estos números.

Junto a las comparaciones simples se admiten también las múltiples. He aquí algunos ejemplos:

```

...
BOOL A,B
...
B = 10 < 3                                ;B=FALSE
A = 10/3 == 3                             ;A=TRUE
B = ((B == A) <> (10.00001 >= 10)) == TRUE ;B=TRUE
A = "F" < "Z"                             ;A=TRUE
...

```

2.3.1.4 Operadores lógicos

Sirven para la combinación lógica de las variables booleanas, constantes y expresiones lógicas simples, tal como se forman con la ayuda de los operadores de comparación. Así, por ejemplo, la expresión

$(A > 5) \text{ AND } (A < 12)$

solamente tiene el valor TRUE, cuando A se sitúa en el rango entre 5 y 12. Dichas expresiones se utilizan con frecuencia en instrucciones para el control de ejecución (véase el capítulo 5). Los operadores lógicos se listan en la Tab. 7.

Operador	Nº operandos	Descripción
NOT	1	Inversión
AND	2	Y lógica
OR	2	O lógico
EXOR	2	O exclusivo

Tab. 7 Operadores lógicos

Los operandos de una combinación lógica deben ser del tipo BOOL, y el resultado es siempre del tipo BOOL. En Tab. 8 están representados los posibles resultados de las combinaciones lógicas respectivas en función del valor de los operandos.

Operación		NOT A	A AND B	A OR B	A EXOR B
A = TRUE	B = TRUE	FALSE	TRUE	TRUE	FALSE
A = TRUE	B = FALSE	FALSE	FALSE	TRUE	TRUE
A = FALSE	B = TRUE	TRUE	FALSE	TRUE	TRUE
A = FALSE	B = FALSE	TRUE	FALSE	FALSE	FALSE

Tab. 8 Tabla de veracidad para combinaciones lógicas

He aquí algunos ejemplos de combinaciones lógicas:

...

BOOL A,B,C

...

A = TRUE

B = NOT A

C = (A AND B) OR NOT (B EXOR NOT A)

A = NOT NOT C

...

;A=TRUE

;B=FALSE

;C=TRUE

;A=TRUE

2.3.1.5 Operadores de bits

Con la ayuda de los operadores de bits (véase Tab. 9), pueden combinarse lógicamente números enteros, combinando lógicamente entre sí los bits individuales de los números. Los operadores de bits combinan bits individuales del mismo modo que los operadores lógicos, dos valores booleanos, cuando el valor de bit 1 se considera como **TRUE** y el valor 0 como **FALSE**.

Una combinación lógica **Y** por bits de los números 5 y 12, proporciona de este modo, por ejemplo, el número 4, una combinación lógica **O** por bits el número 13 y una combinación **O** exclusivo por bits, el número 9:

	0	1	0	1	= 5
	1	1	0	0	= 12
AND	0	1	0	0	= 4
OR	1	1	0	1	= 13
EXOR	1	0	0	1	= 9

Durante la inversión por bits no se complementan simplemente todos los bits. En lugar de ello, cuando se use **B_NOT** se añade 1 al operando y se invierte el signo, por ejemplo:

B_NOT 10 = -11
B_NOT -10 = 9

Los operadores de bits se utilizan, por ejemplo, para combinar entre sí señales de entrada / salida digitales (véase 6.3).

Operador	Nº operandos	Descripción
B_NOT	1	Complementación por bits
B_AND	2	Combinación lógica Y por bits
B_OR	2	Combinación lógica O por bits
B_EXOR	2	Combinación lógica O exclusivo por bits

Tab. 9 Operadores de bits lógicos



Dado que a los caracteres ASCII se tiene acceso también a través de los CÓDIGOS ASCII de números enteros, el tipo de datos de los operandos, además de **INT** puede ser también **CHAR**. El resultado es siempre del tipo **INT**.

Ejemplos para el uso de operadores de bits:

```
...
INT A
...
A = 10 B_AND 9           ;A=8
A = 10 B_OR 9            ;A=11
A = 10 B_EXOR 9          ;A=3
A = B_NOT 197            ;A=-198
A = B_NOT 'HC5'          ;A=-198
A = B_NOT 'B11000101'    ;A=-198
A = B_NOT "E"            ;A=-70
...
```

Suponiendo que haya definido una salida digital de 8 bits de ancho, puede Ud. acceder a la salida a través de la variable INTEGER DIG. Para activar los bits 0, 2, 3 y 7 simplemente puede programar

Intercalar bits `DIG = 'B10001101' B_OR DIG`

Todos los demás bits no se ven influenciados, independientemente del valor que tengan.

Si, por ejemplo, desea enmascarar los bits 1, 2 y 6, debe programar

Enmascarar bits `DIG = 'B10111001' B_AND DIG`

Con ello, todos los demás bits no se modifican.

Con la misma facilidad puede comprobar con los operadores de bits, si bits individuales de la salida se encuentran activados. La expresión

Filtrado de bits `('B10000001' B_AND DIG) == 'B10000001'`

es TRUE, cuando los bits 0 y 7 están activados, de lo contrario es FALSE.

Si solamente quiere comprobar si como mínimo hay activado uno de los dos bits 0 o 7, la combinación lógica Y por bits, únicamente debe ser mayor que cero:

`('B10000001' B_AND DIG) > 0`

2.3.1.6 Prioridades de los operadores

Prioridad

Si utiliza expresiones más complejas con varios operadores, debe tenerse en cuenta las diferentes prioridades de los operadores individuales (véase Tab. 10), ya que las expresiones individuales se ejecutan en el orden de sus prioridades.

Prioridad	Operador
1	NOT B_NOT
2	* /
3	+ -
4	AND B_AND
5	EXOR B_EXOR
6	OR B_OR
7	== <> < > >= <=

Tab. 10 Prioridades de operadores

Básicamente vale:

- Las expresiones entre paréntesis se procesan en primer lugar.
- En las expresiones sin paréntesis se evalúa por el orden de la prioridad.
- Las combinaciones lógicas con operadores de la misma prioridad, se ejecutan de izquierda a derecha.

Ejemplos:

```

...
INT A,B
BOOL E,F
...
A = 4
B = 7
E = TRUE
F = FALSE
...
E = NOT E OR F AND NOT (-3 + A * 2 > B) ;E=FALSE
A = 4 + 5 * 3 - B_NOT B / 2 ;A=23
B = 7 B_EXOR 3 B_OR 4 B_EXOR 3 B_AND 5 ;B=5
F = TRUE == (5 >= B) AND NOT F ;F=TRUE
...

```

2.3.2 Funciones estándar

Para el cálculo de determinados problemas matemáticos, en KRL hay predefinidas algunas funciones estándar (véase Tab. 11). Pueden utilizarse directamente sin otra declaración.

Descripción	Función	Tipo de datos Argumento	Rango de valores Argumento	Tipo de datos Función	Rango de valores Resultado
Cantidad	ABS (X)	REAL	$-\infty \dots +\infty$	REAL	$0 \dots +\infty$
Raíz	SQRT (X)	REAL	$0 \dots +\infty$	REAL	$0 \dots +\infty$
Seno	SIN (X)	REAL	$-\infty \dots +\infty$	REAL	$-1 \dots +1$
Coseno	COS (X)	REAL	$-\infty \dots +\infty$	REAL	$-1 \dots +1$
Tangente	TAN (X)	REAL	$-\infty \dots +\infty^*$	REAL	$-\infty \dots +\infty$
Arco coseno	ACOS (X)	REAL	$-1 \dots +1$	REAL	$0^\circ \dots 180^\circ$
Arco tang.	ATAN2 (Y, X)	REAL	$-\infty \dots +\infty$	REAL	$-90^\circ \dots +90^\circ$
* ningún múltiplo impar de 90° , es decir, $X \neq (2k-1) \cdot 90^\circ$, $k \in \mathbb{N}$					

Tab. 11 Funciones matemáticas estándar

Cantidad	<p>La función ABS (X) calcula la cantidad del valor X, por ejemplo:</p> <pre>B = -3.4 A = 5 * ABS(B) ;A=17.0</pre>
Raíz	<p>SQRT (X) calcula la raíz cuadrada del número X, por ejemplo:</p> <pre>A = SQRT(16.0801) ;A=4.01</pre>
Seno Coseno Tangente	<p>Las funciones trigonométricas SIN (X), COS (X) y TAN (X) calculan el seno, el coseno o la tangente del ángulo X, por ejemplo:</p> <pre>A = SIN(30) ;A=0.5 B = 2 * COS(45) ;B=1.41421356 C = TAN(45) ;C=1.0</pre> <p>Las tangentes de $\pm 90^\circ$ y de los múltiplos impares de $\pm 90^\circ$ ($\pm 270^\circ$, $\pm 450^\circ$, $\pm 630^\circ \dots$) son infinitas. Por lo tanto, el intento de cálculo de uno de estos valores produce un mensaje de fallo.</p>
Arco coseno	<p>ACOS (X) es la función inversa para COS(X):</p> <pre>A = COS(60) ;A=0.5 B = ACOS(A) ;B=60</pre>
Arco seno	<p>Para el Arco seno, la función inversa para SIN (X), no es una función estándar predefinida. Sin embargo, en base a la relación $\text{SIN}(X) = \text{COS}(90^\circ - X)$ puede calcularla también muy fácilmente:</p> <pre>A = SIN(60) ;A=0.8660254 B = 90 - ACOS(A) ;B=60</pre>
Arco tangente	<p>La tangente de un ángulo está definida como cateto opuesto (Y) dividido por el cateto adyacente (X) en un triángulo rectángulo. Si se tiene la longitud de ambos catetos, puede calcularse el ángulo entre el cateto adyacente y la hipotenusa con el arco tangente.</p> <p>Si tomamos ahora un círculo entero, resultará decisivo el signo que tengan los componentes X e Y. Si solamente se han tenido en cuenta los cocientes, con el arco tangente solamente han podido calcularse el ángulo entre 0° y 180°. Esto ocurre también con todas las calculadoras de bolsillo normales: el arco tangente de valores positivos proporciona un ángulo entre 0° y 90°, el arco tangente de valores negativos un ángulo entre 90° y 180°.</p> <p>Mediante la indicación explícita de X e Y el cuadrante está unívocamente especificado a través de su signo, en el que se encuentra el ángulo (véase Fig. 6). Por ello, puede calcular</p>

también el ángulo en los cuadrantes III y IV. Por lo tanto, para el cálculo del arco tangente, en la función **ATAN2 (Y, X)** son necesarios también estos dos datos, por ejemplo:

A = ATAN2 (0.5, 0.5)	;A=45
B = ATAN2 (0.5, -0.5)	;B=135
C = ATAN2 (-0.5, -0.5)	;C=225
D = ATAN2 (-0.5, 0.5)	;D=315

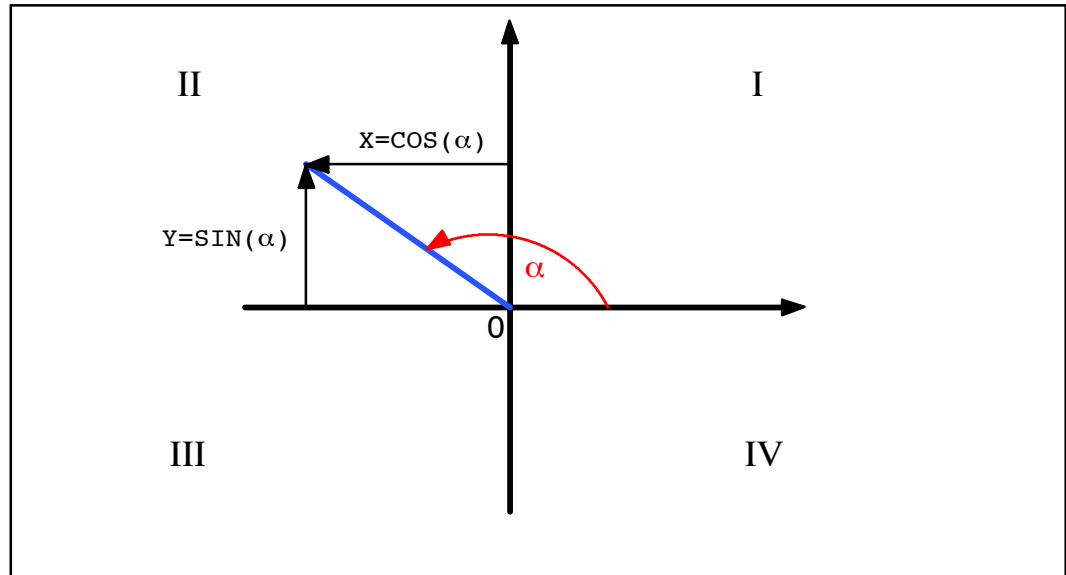
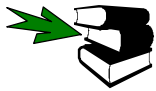


Fig. 6 Utilización de x e y en la función ATAN2 (Y, X)



Informaciones adicionales se encuentran en el capítulo **[Subprogramas y funciones]**.

2.4 Variables y archivos del sistema

Un requisito importante para el procesamiento de aplicaciones complejas en la técnica de robots, es una unidad de control programable en forma libre y cómoda.

Para ello, debe poderse programar fácilmente la funcionalidad de la unidad de control del robot en el lenguaje del robot. Recién cuando la integración de los parámetros de control en un programa de robot resulta fácil y está completa, podrá aprovechar la totalidad de las funcionalidades de una unidad de control del robot. En la KR C..., ésto se soluciona de forma excelente a través del concepto de las variables y archivos predefinidos del sistema.

Son ejemplos de variables predefinidas \$POS_ACT (posición actual del robot), \$BASE (sistema básico de coordenadas) o \$VEL_CP (velocidad de trayectoria).



Un listado de todas las variables predefinidas las encuentra Ud. en la documentación por separado **[Variables del sistema]**.

Las variables del sistema están totalmente integradas en el concepto de variables de la KRL. Poseen un tipo de datos correspondiente y pueden ser leídas y escritas como cualquier otra variable en el programa, siempre y cuando no existan limitaciones debido al tipo de datos. La posición actual del robot, solamente puede leerse, por ejemplo, no escribirse. La unidad de control verifica estas limitaciones.

En la medida en que desde el punto de vista de la técnica de seguridad sea posible, dispone Ud. incluso de acceso de escritura a datos del sistema. De este modo, se cuenta con múltiples posibilidades para el diagnóstico, ya que desde la KCP y desde el sistema de programación, pueden cargarse una gran variedad de datos del sistema o influirse en ellos.

Variables del sistema útiles con acceso de escritura son, por ejemplo \$TIMER[] y \$FLAG[].

Temporizador

Las 16 variables de temporizador \$TIMER[1]...\$TIMER[16], sirven para medir los procesos temporales y de este modo pueden utilizarse como "cronómetro". El inicio y la parada del proceso de medición se efectúa con las variables del sistema

\$TIMER_STOP[1]...\$TIMER_STOP[16]:

\$TIMER_STOP[4] = FALSE

arranca, por ejemplo, el temporizador 4,

\$TIMER_STOP[4] = TRUE

detiene de nuevo el temporizador 4. A través de una asignación normal de valores, puede resetearse en cualquier momento la variable de temporizador afectada, por ejemplo:

\$TIMER[4] = 0

Si cambia el valor de una variable de temporizador de menos a más, se coloca un flag correspondiente en TRUE (condición Timer-Out), por ejemplo:

\$TIMER_FLAG[4] = TRUE

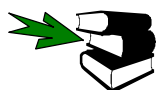
Al arrancar la unidad de control, todas las variables de temporizador se predefinen con 0, las señales

\$TIMER_FLAG[1]...\$TIMER_FLAG[16] con FALSE y las variables

\$TIMER_STOP[1]...\$TIMER_STOP[16] con TRUE.

La unidad de las variables de temporizador es el milisegundo (ms). La actualización de \$TIMER[1]...\$TIMER[16] así como de \$TIMER_FLAG[1]...\$TIMER_FLAG[16], se efectúa con un intervalo de 12 ms.

Flags	Las 1024 señales <code>\$FLAG[1]...\$FLAG[1024]</code> se utilizan como marcas globales. Estas variables booleanas se predefinen con <code>FALSE</code> . Ud. puede interrogar, en cualquier momento, el valor actual de los flags sobre la superficie de operación, en la opción de menú "Visualización".
Flags cíclicos	<p>Además, en el KR C... hay 32 flags cíclicos <code>\$CYCFLAG[1]...\$CYCFLAG[32]</code> disponibles. Después del arranque de la unidad de control están todas predefinidas con <code>FALSE</code>.</p> <p>Las señales solamente están activas cíclicamente en el nivel de robot. En un archivo submit son admisibles los flags cíclicos, pero no se efectúa ninguna evaluación cíclica.</p> <p>Los flags cíclicos pueden definirse y activarse también en subprogramas, funciones y subprogramas de interrupción.</p> <p><code>\$CYCFLAG[1]...\$CYCFLAG[32]</code> tienen el tipo de datos <code>BOOL</code>. Cuando se efectúa una asignación a un flag cíclico puede utilizarse la expresión booleana que se desee.</p> <p>Son admisibles</p> <ul style="list-style-type: none"> variables booleanas del sistema y variables booleanas, que se declaran e inician en una lista de datos. <p>No son admisibles</p> <ul style="list-style-type: none"> funciones que reenvían un valor booleano. <p>La instrucción</p> <pre>\$CYCFLAG[10] = \$IN[2] AND \$IN[13]</pre> <p>por ejemplo, hace que la expresión booleana "<code>\$IN[2] AND \$IN[13]</code>" se evalúe cíclicamente. Es decir, que siempre que se modifique la entrada 2 o la entrada 13, se modifica también <code>\$CYCFLAG[10]</code>, no importa el lugar en que se encuentre el puntero de ejecución del programa después de la ejecución de la expresión anterior.</p> <p>Todos los flags cíclicos definidos siguen siendo válidos hasta que se selecciona un módulo o se efectúa una selección de paso por reset. Si se alcanza el final del programa, todos los flags cíclicos se siguen manteniendo activos.</p>



Informaciones adicionales se encuentran en el capítulo **[Tratamiento de interrupciones]**, apartado **[Utilización de banderas (flags) cíclicas]**.

\$ – símbolo

En general, los nombres de las variables predefinidas se seleccionan de forma que pueden memorizarse fácilmente. Todos comienzan con un símbolo \$ y constan de una abreviatura inglesa fácil de recordar. Dado que se tratan como variables habituales, no es necesario que recuerde instrucciones extraordinarias ni opciones extrañas.

Para evitar confusiones, no debería Ud. declarar ninguna variable que comience con un símbolo \$.

Una parte de las variables predefinidas se refiere a toda la unidad de control KR C... (por ejemplo, `$ALARM_STOP` para la definición de la salida para la señal de parada de emergencia hacia el PLC). Sin embargo, otras solamente son importantes para el robot (por ejemplo, `$BASE` para el sistema de coordenadas base).

Sobre la superficie de operación de KUKA aparece la estación de discos del robot sobre el cual se han memorizado los datos relevantes de control en el directorio “Steu” y los datos relevantes del robot en el directorio “R1”.

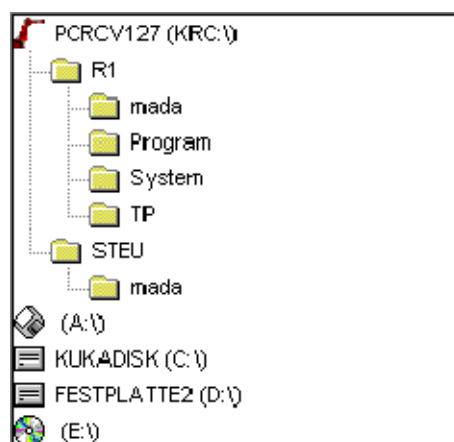


Fig. 7 Diferentes niveles en la superficie de operación KUKA

Para la programación de la KR C... puede Ud. crear archivos de programa y listas de datos. En los archivos de programa hay definiciones de datos e instrucciones ejecutables, las listas de datos solamente contienen definiciones de datos y eventualmente inicializaciones.



Informaciones adicionales se encuentran en el capítulo **[Listas de datos]**.

Junto a las listas de datos que Ud. crea durante la programación, en el KR C... existen todavía listas de datos definidas por KUKA y que se suministran con el software de control. Estas listas de datos se llaman listas de datos predefinidas y contienen principalmente las variables predefinidas.

Las listas de datos predefinidas no puede borrarlas ni crearlas por sí mismo. Se crean durante la instalación del software y existen siempre. También los nombres de las listas de datos predefinidas comienzan, como los nombres de los datos predefinidos, con un símbolo \$.

En la KR C... existen las siguientes listas predefinidas de datos:

- **\$MACHINE.DAT**
es una lista de datos predefinida que contiene exclusivamente variables de sistema predefinidas. Con los datos de la máquina, se adapta la unidad de control a los robots conectados (cinemática, parámetros de regulación, etc.). Existe un \$MACHINE.DAT tanto en el sistema de control como en el sistema de robot. Ud. no puede crear ninguna nueva variable ni borrar las existentes.

Ejemplos:

\$ALARM_STOP	Señal para parada de emergencia (específica de la unidad de control)
\$NUM_AX	Número de ejes del robot (específico del robot)

- **\$CUSTOM.DAT**
es una lista de datos que solamente existe en el sistema de control. Contiene datos con los que puede configurar o parametrizar determinadas funciones de control. El programador tiene la posibilidad de modificar los valores de las variables predefinidas. Ud. no puede generar ninguna nueva variable ni borrar las existentes.

Ejemplos:

\$PSER_1	Parámetro de protocolo del interfaz serie 1
\$IBUS_ON	Conexión de grupos alternativos de interbus

▪ **\$CONFIG.DAT**

es una lista de datos predefinida por KUKA, pero que no contiene ninguna variable de sistema predefinida. Para ello existe un \$CONFIG.DAT en el nivel de la unidad de control y en el nivel de robot. En ella pueden definirse variables, estructuras, canales y señales que son válidas durante mucho tiempo y que son de la máxima importancia para muchos programas.

La lista de datos se subdivide en los bloques siguientes:

- BAS
- AUTOEXT
- GRIPPER
- PERCEPT
- SPOT
- A10
- A50
- A20
- TOUCHSENSE
- USER

Las declaraciones globales del usuario deben registrarse necesariamente en el bloque USER. Solamente aquí se aceptan las declaraciones para una actualización de software posterior.

▪ **\$ROBCOR.DAT**

El archivo \$ROBCOR.DAT contiene datos específicos del robot para el modelo dinámico del robot. Estos datos son necesarios para la planificación de la trayectoria. Tampoco, en este caso, Ud. puede generar ninguna nueva variable ni borrar las existentes.

Tab. 12 proporciona un resumen de las listas de datos predefinidas.

Lista de datos	Sistema		Asignación de valor	
	Control	Robot	con	por
\$MACHINE.DAT	✓	✓	Puesta en servicio	KUKA/usuario
\$CUSTOM.DAT	✓		Puesta en servicio	Usuario/KUKA
\$CONFIG.DAT	✓	✓	Montaje o reequipamiento de células	Usuario/KUKA
\$ROBCOR.DAT		✓	Suministro	KUKA

Tab. 12 Listas de datos predefinidas en la KR C...

3 Programación de movimiento

Instrucciones de movimiento

Entre las tareas más importantes de una unidad de control de robot se incluye el movimiento del robot. El programador controla los movimientos del robot industrial con la ayuda de instrucciones de movimiento especiales. Estas constituyen también la característica principal que diferencia los lenguajes de robot de los lenguajes de programación comunes para un ordenador, como el C o el Pascal.

Según el tipo de control, estas instrucciones de movimiento pueden subdividirse en instrucciones para movimientos simples de punto a punto y en instrucciones para movimientos de trayectoria. Mientras que en los movimientos de trayectoria, el efector final (por ejemplo, garra o herramienta) describe una trayectoria geométrica exactamente definida en el espacio (recta o arco de circunferencia), en los movimientos de punto a punto, la trayectoria de movimiento depende de la cinemática del robot y por ello no es exactamente previsible. Ambos tipos de movimiento tienen en común, que la programación se efectúa desde la posición actual a una nueva posición. Por lo tanto, en general, en una instrucción de movimiento solamente es necesario indicar la posición destino (excepción: movimientos circulares, véase 3.3.4).

Las coordenadas de las posiciones pueden especificarse textualmente mediante la introducción de valores numéricos o mediante el desplazamiento con el robot hasta el punto en el espacio y la memorización de los valores actuales (aprendizaje). Para ello, existe la posibilidad de referir las indicaciones a diferentes sistemas de coordenadas.

Otras propiedades de movimiento, tales como velocidades y aceleraciones, así como el guiado, pueden ser ajustados por medio de variables del sistema. El posicionamiento aproximado de los puntos auxiliares se inicia por medio de los parámetros opcionales en la instrucción de movimiento. Para el posicionamiento aproximado debe estar activado un procesamiento en avance.

3.1 Utilización de diferentes sistemas de coordenadas

Sistema de coordenadas

Para poder indicar en el espacio la posición u orientación de un punto, se utilizan diferentes sistemas de coordenadas. Puede efectuarse una diferenciación básica entre el sistema de coordenadas específico del eje y cartesiano:

Específico del eje

En el **sistema de coordenadas específico del eje** se indican los corrimientos (para los ejes de traslación) o giros (para los ejes rotatorios) de cada eje del robot. En un robot de brazo articulado de 6 ejes, deben indicarse los 6 ángulos de articulación del robot, para especificar unívocamente la posición y la orientación (véase Fig. 8).

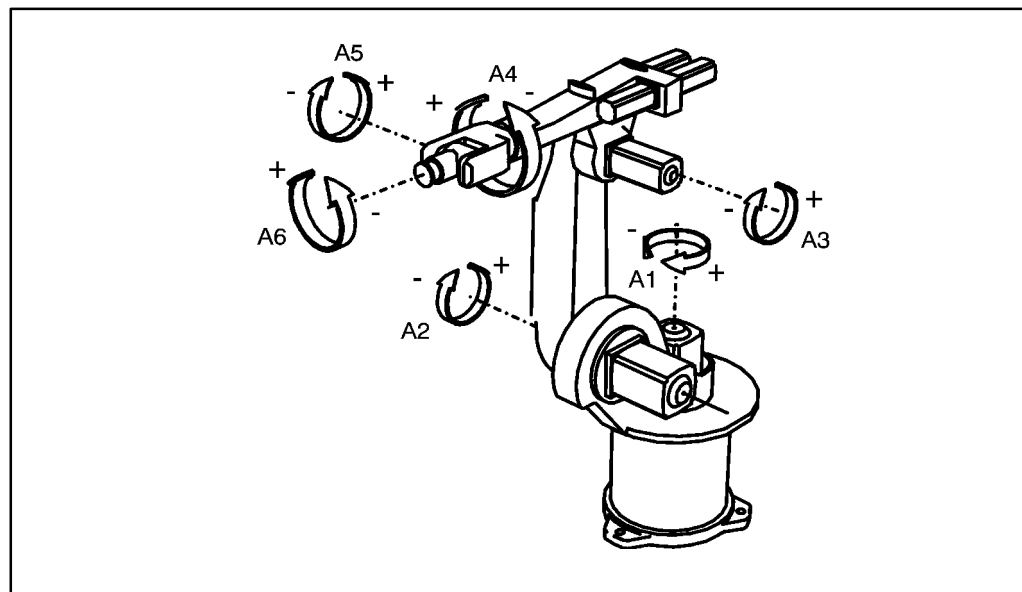
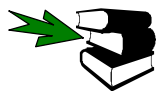


Fig. 8 Sistema de coordenadas específico del eje de un robot de brazo articulado de 6 ejes

La descripción de una posición específica del eje se efectúa en la KR C... mediante el tipo de estructura predefinida **AXIS**, cuyos componentes tienen el significado de los ángulos o longitudes según el tipo de eje.



Informaciones adicionales se encuentran en el capítulo **[Variables y declaraciones]**, apartado **[Estructuras]**.

Transformación de las coordenadas

Las posiciones específicas de los ejes solamente pueden alcanzarse en combinación con los pasos de movimiento PTP. Si un movimiento de trayectoria se programa con una posición de robot específica del eje, esto da lugar a un fallo.

Dado que el ser humano como programador piensa en coordenadas cartesianas, la programación en el sistema de coordenadas específicas del eje suele ser poco práctica. La unidad de control ofrece por ello, varios sistemas de coordenadas cartesianas para la programación, cuyas coordenadas, antes de la ejecución del movimiento, se transforman automáticamente en el sistema específico del eje (véase Fig. 9).

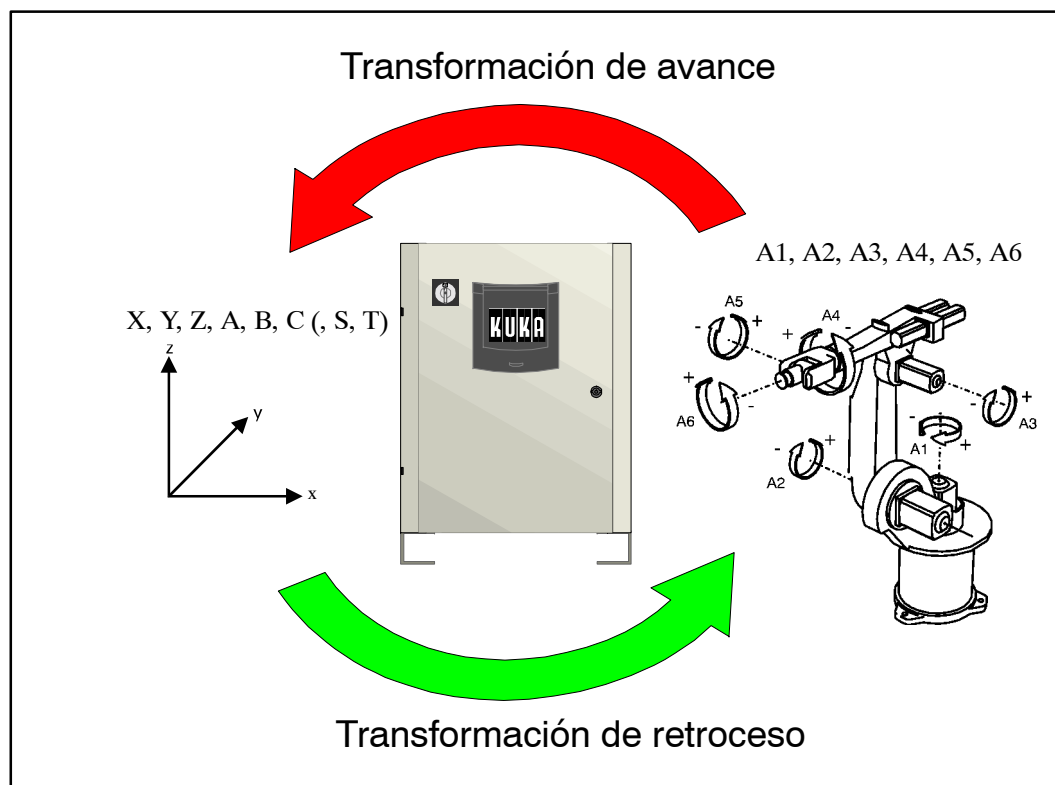


Fig. 9 Transformación de coordenadas

Cartesiano

En un **sistema de coordenadas cartesiano** los 3 ejes de coordenadas X, Y y Z se encuentran perpendiculares unos respecto a otros y forman, en este orden, un sistema rectangular.

La posición de un punto en el espacio está determinada unívocamente en el sistema de coordenadas cartesiano mediante la indicación de las coordenadas X, Y y Z. Estas se obtienen de las distancias traslatorias de cada valor de coordenadas hacia el origen de las coordenadas (véase Fig. 10).

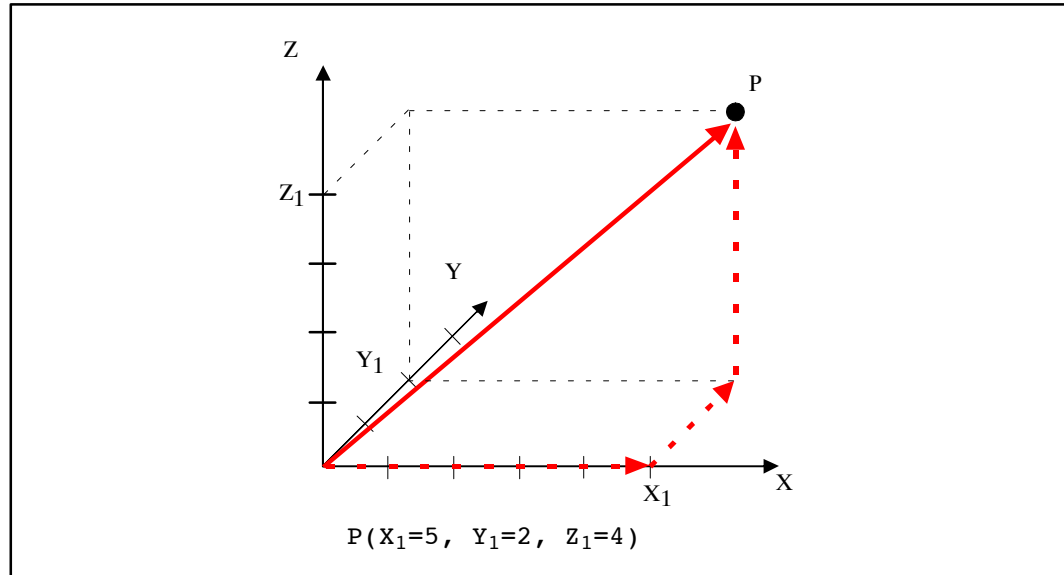


Fig. 10 Descripción traslatoria de la posición de un punto

Para poder alcanzar cualquier punto en el espacio con la orientación que se desee, junto a los tres valores traslatorios, son necesarios además, tres datos rotatorios:

Los ángulos denominados en el KR C... con A, B y C, describen giros en torno a los ejes de coordenadas Z, Y y X. Para ello, debe mantenerse el orden de las rotaciones:

1. Giro en torno al eje Z en el ángulo A
2. Giro en torno al nuevo eje Y en el ángulo B
3. Giro en torno al nuevo eje X en el ángulo C

Este orden de giro corresponde a los ángulos roll-pitch-yaw (rodar-cabecear-oscilar) conocidos de la aeronáutica. El ángulo C corresponde al rodar, el ángulo B al cabecear y el ángulo A al oscilar (véase Fig. 11).

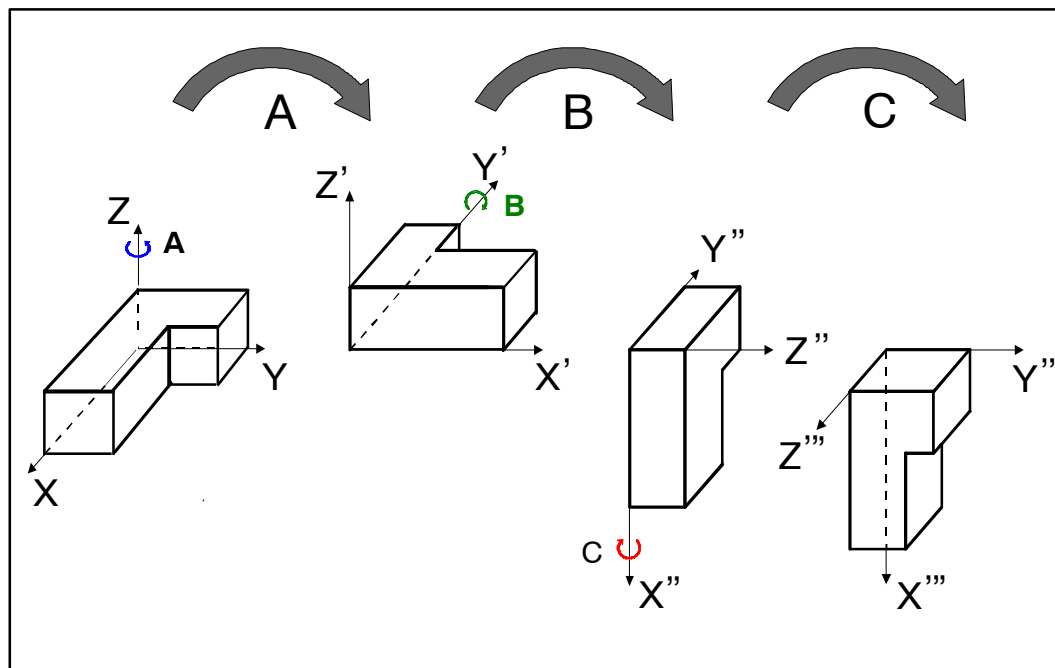
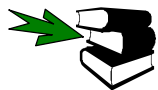


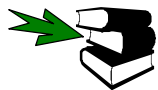
Fig. 11 Descripción por medio de las rotaciones de la orientación de un punto

Con las translaciones X , Y y Z , y con las rotaciones A , B y C puede describirse unívocamente la posición y orientación de un punto en el espacio. En la KR C..., esto se produce con la estructura predefinida `FRAME`.



Informaciones adicionales se encuentran en el capítulo **[Variables y declaraciones]** apartado **[Estructuras]**.

Para los movimientos de trayectoria, la indicación de las coordenadas `FRAME` es siempre unívoca y suficiente. Sin embargo, en el desplazamiento punto a punto, con determinadas cinemáticas de robots (por ejemplo, brazo articulado de 6 ejes) puede alcanzarse uno y el mismo punto (posición y orientación) en el espacio con varias posiciones de eje. Con los otros dos datos "S" y "T", puede solucionarse esta ambigüedad. En la KR C... está prevista la estructura `POS` para un frame ampliado con "S" y "T".



Informaciones adicionales se encuentran en el capítulo **[Variables y declaraciones]** apartado **[Estructuras]** y apartado **[Instrucciones de movimiento]**.

En la KR C..., están predefinidos los siguientes sistemas de coordenadas cartesianos (Tab. 13 y Fig. 12):

Sistema de coordenadas	Variable del sistema	Estado
Sistema de coordenadas universal	\$WORLD	protegida contra escritura
Sistema de coordenadas del robot	\$ROBROOT	protegida contra escritura (modificable en R1\MADA\ \$MA-CHINE.DAT)
Sistema de coordenadas de la herramienta	\$TOOL	a describir
Sistema de coordenadas base (de la pieza)	\$BASE	a describir

Tab. 13 Sistemas de coordenadas predefinidos

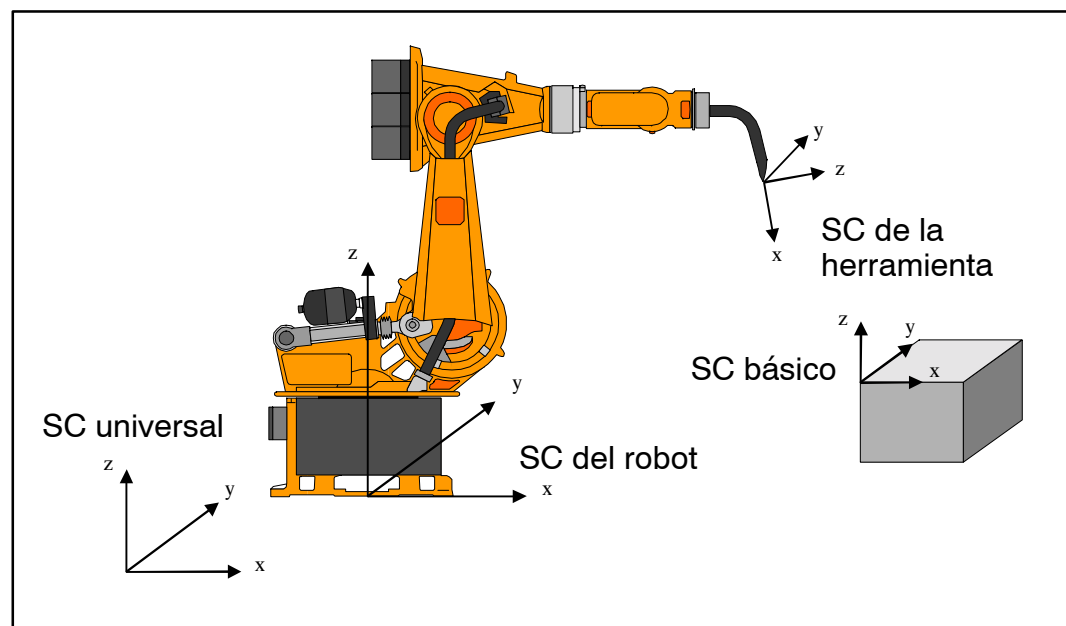


Fig. 12 Sistemas de coordenadas cartesianos para robots

Sistema de coordenadas universal

El sistema de coordenadas universales es un sistema de coordenadas fijo en un lugar (= no se mueve con el movimiento del robot), y que sirve como sistema de coordenadas genérico para un sistema de robot (robot, útil de recepción de una pieza, o bien, una herramienta). De este modo, representa el sistema de referencia tanto para el sistema de robot como también para los periféricos de la célula.

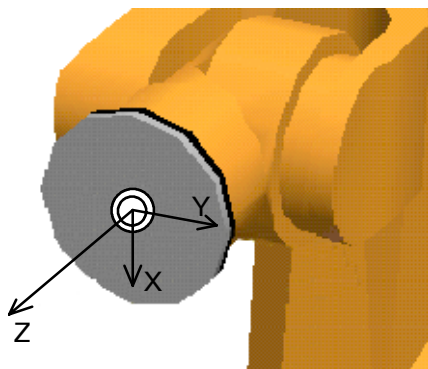
Sistema de coordenadas del robot

Este sistema de coordenadas se encuentra en el pie del robot y sirve como sistema de coordenadas de referencia para la estructura mecánica del robot. Se refiere también al sistema de coordenadas universal e idéntico al mismo en el suministro. Con \$ROBROOT puede definirse un corrimiento del robot hacia \$WORLD.

Sistema de coordenadas de la herramienta

El sistema de coordenadas de la herramienta tiene su origen en la punta de la herramienta. La orientación puede seleccionarse de modo tal, que su eje X sea idéntico con la dirección

de avance de la herramienta, mostrando fuera de la misma. Con el movimiento de la punta de la herramienta, se mueve también el sistema de coordenadas de la herramienta.



En el suministro, el sistema de coordenadas de la herramienta se encuentra en la brida del robot (el eje Z es idéntico al eje 6). Se refiere, a través de la transformación, al sistema de coordenadas del robot.

Si se efectúa un cambio de herramienta, después de la nueva medición, puede seguirse utilizando el programa original, ya que el ordenador conoce las coordenadas de la punta de la herramienta.

Sistema de coordenadas base

El sistema de coordenadas base se utiliza como sistema de referencia para la descripción de la posición de la pieza a trabajar. La programación del robot se efectúa en el sistema de coordenadas base. Como sistema de coordenadas de referencia utiliza el sistema de coordenadas universal. En el suministro del robot es \$BASE=\$WORLD.

Mediante una modificación de \$BASE, por ejemplo, pueden procesarse varias piezas a trabajar iguales en diferentes lugares, con el mismo programa.

Interpolación referida a la base

Para la interpolación de la trayectoria de movimiento, la unidad de control del robot calcula normalmente (herramienta en la brida del robot) la posición actual (\$POS_ACT) con referencia al sistema de coordenadas \$BASE (véase Fig. 13).

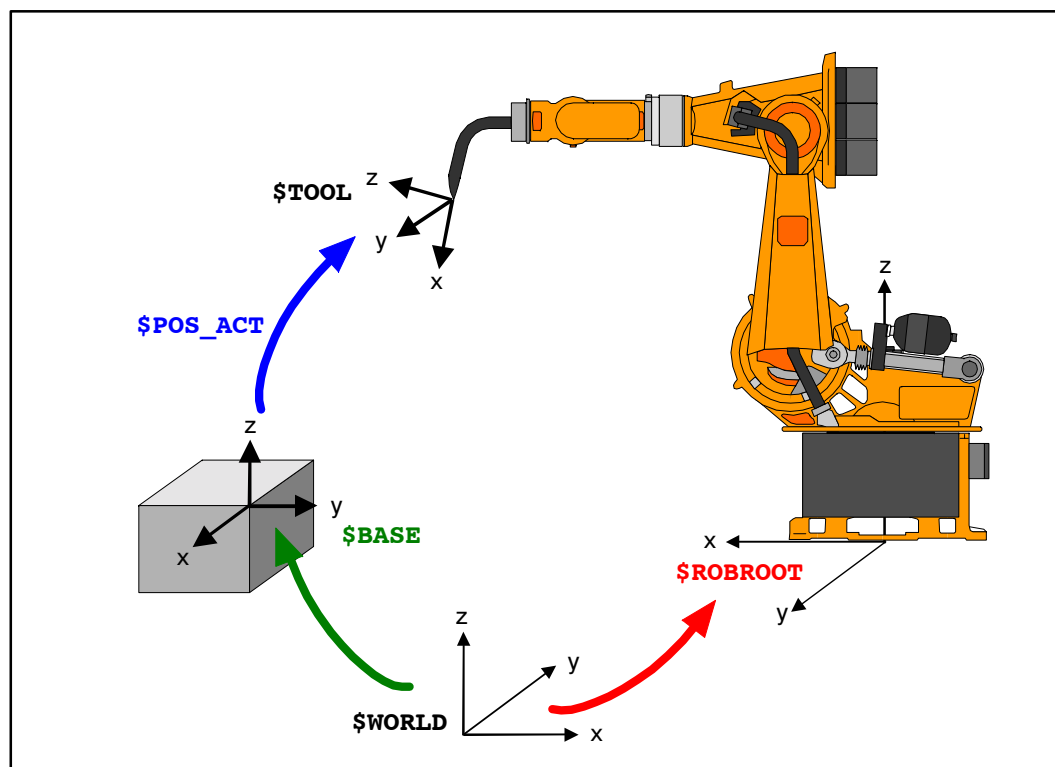


Fig. 13 Cadena cinemática para la interpolación referida a la base

Herramienta fija

Sin embargo, en la práctica industrial, es cada vez más usual fijar la herramienta (por ejemplo, antorcha de soldadura) en el espacio y conducir la pieza a trabajar por medio de una garra adecuada a lo largo de la herramienta fija.



La variable **\$TOOL** hace referencia siempre a la herramienta o la pieza que se encuentra en el robot. Contrariamente a ello, la variable **\$BASE** se refiere siempre a una herramienta o pieza externa.

Interpolación referida a la garra

Dado que ahora pieza y herramienta intercambiaron su posición, pero el movimiento debe seguir referido a la pieza, la interpolación de la trayectoria del movimiento debe efectuarse entonces a través del sistema de coordenadas **\$TOOL**. Esta asignación del tipo de interpolación se efectúa en forma implícita en la utilización de un TCP normal o externo. Con las variables del sistema **\$IPO_MODE** puede Ud. definir este tipo de interpolación. La línea de programa

```
$IPO_MODE = #TCP
```

permite una interpolación referida a la garra en el sistema de coordenadas **\$TOOL**. La posición actual **\$POS_ACT** se calcula ahora en referencia a **\$TOOL** (véase Fig. 14). Con

```
$IPO_MODE = #BASE
```

define Ud. nuevamente el tipo de interpolación en la interpolación referida a la base para el caso normal. Este es también el ajuste estándar al arrancar la unidad de control.

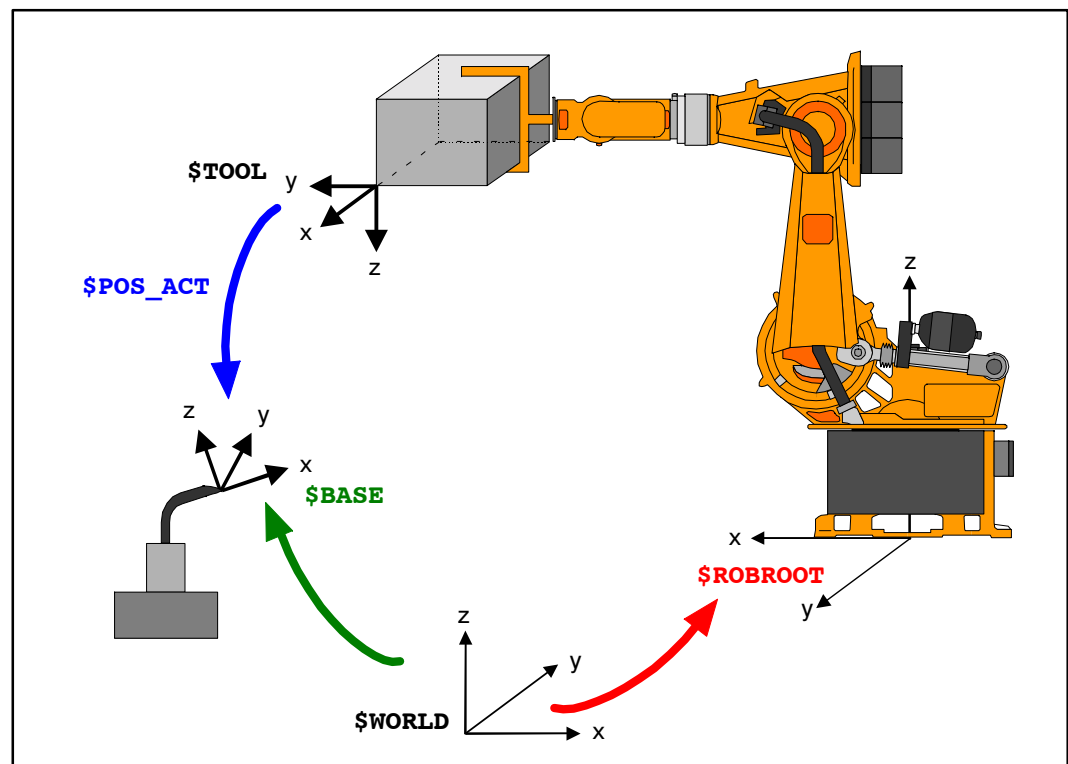


Fig. 14 Cadena cinemática para la interpolación referida a la garra



Un ejemplo de corrimiento del sistema de coordenadas se encuentra en el capítulo [Variables y declaraciones], apartado [Operador geométrico].

3.2 Movimientos punto a punto (PTP)

3.2.1 General (PTP sincrónico)

PTP

El movimiento punto a punto (PTP) es la posibilidad más rápida de mover la punta de la herramienta (Tool Center Point: TCP) desde la posición actual hasta una posición destino programada. La unidad de control calcula para ello las diferencias de ángulo necesarias para cada eje.

Con la ayuda de la variable del sistema

- **\$VEL_AXIS**[*número de eje*] se programan las velocidades máximas específicas del eje,

y con

- **\$ACC_AXIS**[*número de eje*] las aceleraciones del eje máximas específicas.

Todos los datos se indican en porcentaje, en referencia a un valor máximo definido en los datos de la máquina. En el caso de que estas dos variables del sistema no se hubiesen programado para todos los ejes, al ejecutarse el programa se emite el mensaje de fallo correspondiente.

Los movimientos de los ejes individuales se sincronizan para ello (PTP sincrónico) de forma que todos los ejes inician y finalizan simultáneamente el movimiento. Esto significa que sólo el eje con el recorrido más largo, el llamado eje guía, se desplaza con el valor límite programado para aceleración y velocidad. Todos los demás ejes se mueven solamente con las aceleraciones y velocidades necesarias para alcanzar el punto final del movimiento en el mismo instante, independientemente de los valores programados en **\$VEL_AXIS**[*Nro*] y **\$ACC_AXIS**[*Nro*].

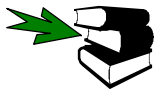
Si la adaptación de aceleración o la característica de velocidad superior está activada (**\$ADAP_ACC**=#STEP1, **\$OPT_MOVE**=#STEP1) los ejes se desplazan además de forma sincrónica en sus fases, es decir, todos los ejes se encuentran al mismo tiempo en la fase de aceleración, de movimiento constante y de retardo.



Dado que en general, en los movimientos PTP con coordenadas destino cartesianas, no se conoce cuál es el eje guía, en este caso suele ser conveniente equiparar los valores de aceleración y de velocidad para todos los ejes.

El mando de los movimientos sincrónicos en tiempo disminuye la carga mecánica del robot, dado que los momentos de los motores y reductores de los ejes con trayectos de desplazamiento menor se reducen.

El mando de movimiento sincrónico de fases lleva (adicionalmente) a una trayectoria de movimiento, que independientemente de la velocidad y aceleración programada, siempre de igual ejecución en el espacio.



Informaciones adicionales se encuentran en el capítulo **[Programación de movimiento]**, apartado **[Movimientos con posicionamiento aproximado]**.

3.2.2 Perfil de marcha más elevado

Por ello, en forma estándar, se utiliza un perfil de marcha más elevado para los movimientos PTP. Con este modelo en **pasos PTP individuales** y en **pasos PTP con programación aproximada**, el desplazamiento se efectúa desde el punto inicial al de destino optimizado en tiempo. Esto significa, que con los reductores y motores existentes no es posible desplazarse más rápidamente. Los momentos admitidos se aprovechan siempre óptimamente en cada punto de la trayectoria, lo cual resulta especialmente aplicable también en la fase de velocidad constante. En todos los casos, la velocidad es adaptada de modo tal que en ningún momento se sobrepasan los momentos.

Una modificación de los valores de velocidad o de aceleración produce también, para las instrucciones de posicionamiento aproximado, sólo una modificación del perfil de la característica de velocidad en la trayectoria. La curva geométrica en el espacio se mantiene sin modificar.

Las asignaciones de velocidad y los valores límites de las aceleraciones (declaración en porcentaje) pueden ser definidas para cada eje de forma individual. Pero este valor límite no actúa directamente sobre la aceleración, sino sobre el momento de aceleración del eje. Es decir, un valor de aceleración en el eje de 50% no reduce, necesariamente, la aceleración a la mitad.

3.2.3 Instrucciones de movimiento

El ejemplo de programa siguiente, **PTP_AXIS.SRC** representa básicamente, el programa KRL ejecutable más pequeño:



```

DEF PTP_AXIS()           ;el nombre del programa es PTP_AXIS

$VEL_AXIS[1]=100         ;Especificación de las velocidades de eje
$VEL_AXIS[2]=100
$VEL_AXIS[3]=100
$VEL_AXIS[4]=100
$VEL_AXIS[5]=100
$VEL_AXIS[6]=100

$ACC_AXIS[1]=100         ;Especificación de las aceleraciones de eje
$ACC_AXIS[2]=100
$ACC_AXIS[3]=100
$ACC_AXIS[4]=100
$ACC_AXIS[5]=100
$ACC_AXIS[6]=100

PTP {AXIS: A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}

END

```



En primer lugar se especifican en este programa las velocidades y aceleraciones de los ejes. Antes de que pueda ejecutarse un movimiento punto a punto, tienen que haberse realizado estas asignaciones.

Posteriormente, el robot desplaza cada eje a las posiciones angulares especificadas en la estructura AXIS. Por ejemplo eje 1 a 0°, eje 2 a -90°, eje 3 a 90°, eje 4 a 0°, eje 5 a 0° y eje 6 a 0°.

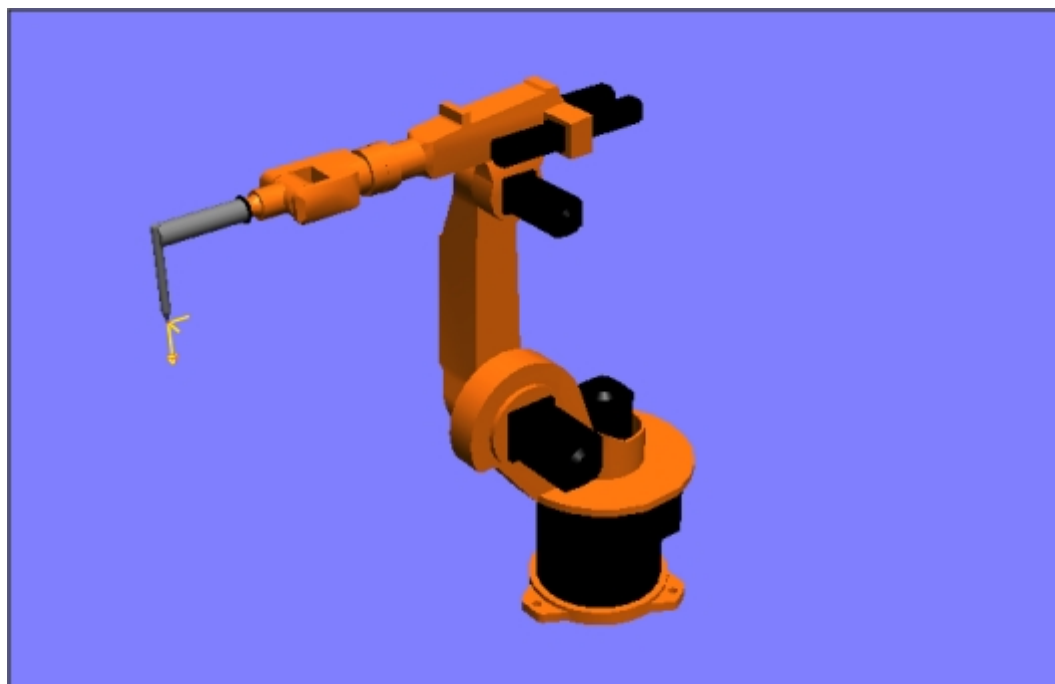


Fig. 15 Posición mecánica cero

Si al indicar las coordenadas de los ejes se omiten componentes individuales, el robot solamente desplaza los ejes indicados, mientras que los otros no cambian su posición. Con

PTP

PTP {A3 45}

por lo tanto, solamente se mueve, por ejemplo, el eje 3 hasta 45° . Debe tenerse en cuenta que para los datos angulares en la instrucción PTP, se trata de valores absolutos. El robot no gira el eje otros 45° , sino que se desplaza a la posición de 45° absolutos del eje.

Para el desplazamiento relativo sirve la instrucción PTP_REL. Para girar también, por ejemplo, los ejes 1 y 4, 35° cada uno, simplemente programe:

PTP_REL

PTP_REL {A1 35,A4 35}

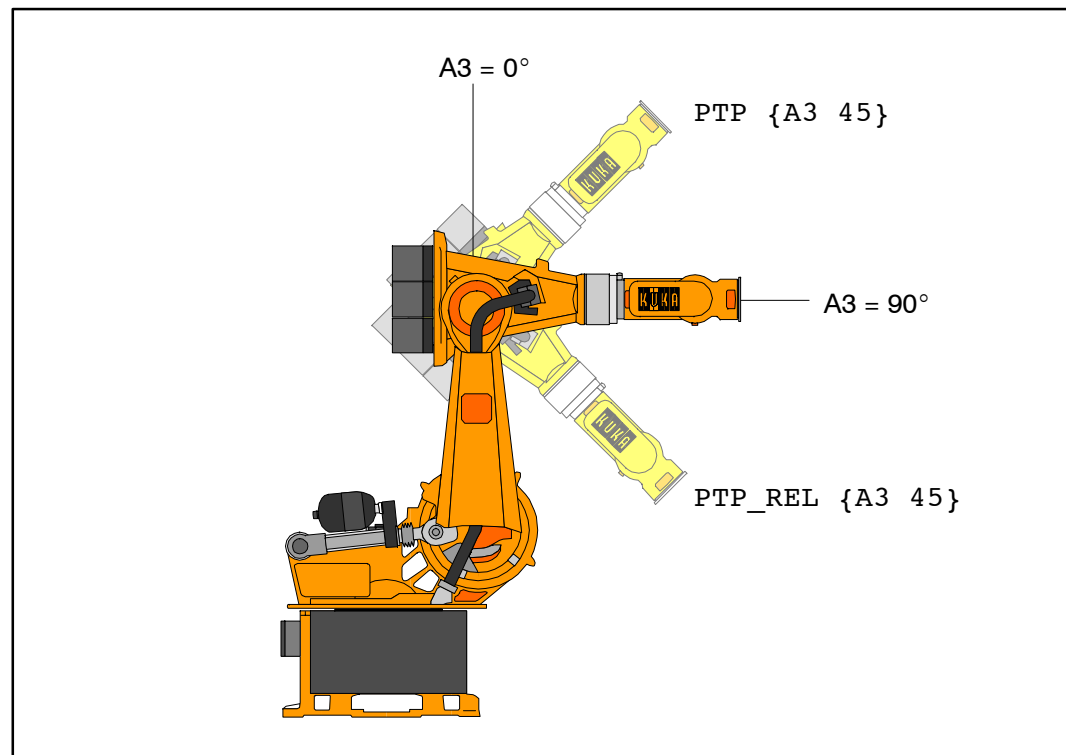


Fig. 16 Diferencia entre las coordenadas absolutas y relativas específicas del eje



Sin embargo, para el desplazamiento relativo debe tenerse en cuenta que un movimiento detenido durante la ejecución de un programa, no puede continuarse de nuevo sin problemas. La unidad de control no puede tener en cuenta, después de un nuevo arranque y de una nueva selección de paso o un cambio del modo de ejecución del programa, el recorrido ya realizado y recorre de nuevo en forma completa la distancia relativa programada, lo que finalmente conduce a un punto final incorrecto.

Sin embargo, el desplazamiento en las coordenadas específicas del eje es, en general poco práctico, ya que la persona piensa y actúa en espacio cartesiano. Para ello sirve la indicación de las coordenadas cartesianas por medio de una estructura POS, como se utiliza en el ejemplo siguiente:



```
DEF PTP_POS ( )

$BASE = $WORLD      ;Definición del sistema de coordenadas base
$TOOL = $NULLFRAME ;Definición del sistema de coordenadas de la
                    herramienta

$VEL_AXIS[1]=100    ;Especificación de las velocidades de eje
$VEL_AXIS[2]=100
$VEL_AXIS[3]=100
$VEL_AXIS[4]=100
$VEL_AXIS[5]=100
$VEL_AXIS[6]=100

$ACC_AXIS[1]=100    ;Especificación de las aceleraciones de eje
$ACC_AXIS[2]=100
$ACC_AXIS[3]=100
$ACC_AXIS[4]=100
$ACC_AXIS[5]=100
$ACC_AXIS[6]=100

PTP {POS:X 1025,Y 0,Z 1480,A 0,B 90,C 0,S 'B 010',T 'B 000010'}

END
```

Debe tenerse en cuenta ahora que al indicar el punto final en las coordenadas cartesianas, junto a los datos de velocidad y aceleración deben estar definidos obligatoriamente también el sistema de coordenadas base y el sistema de coordenadas de la herramienta.

Sistemas de
coordenadas

En nuestro caso, el sistema de coordenadas base (\$BASE) fue definido igual que el sistema de coordenadas universal (\$WORLD), que de forma estándar, se encuentra en el pie del robot (\$ROBROOT). El sistema de coordenadas de la herramienta (\$TOOL) fue ocupado con el frame cero (\$NULLFRAME = {FRAME: X 0, Y 0, Z 0,A 0,B 0,C 0}), lo que significa, que todos los datos hacen referencia al punto central de la brida. El punto central de la herramienta (Tool Center Point: TCP) se encuentra, por así decirlo, en el punto central de la brida. En el caso que una herramienta se encuentre montada sobre la brida, los valores tendrían que corregirse correspondientemente. Para ello debe remitirse a la documentación para la medición de las herramientas.

Con la instrucción PTP mostrada arriba, el robot se desplaza ahora de modo tal que en el punto final del movimiento del TCP recorre 1025 mm en dirección X, 0 mm en dirección Y, y 1480 mm en dirección Z desde el pie del robot. Los datos "A", "B" y "C" definen la orientación del TCP. El estado "S" y giro "T" definen la posición de los ejes.

Si prueba el ejemplo en un robot KR 6, obtendrá el mismo resultado que en el ejemplo anterior. El robot se desplaza a la posición mecánica cero. Las dos instrucciones son idénticas para este tipo de robot.

También en la indicación del punto final en las coordenadas cartesianas pueden omitirse aquí componentes individuales de los datos de geometría. La instrucción

```
PTP {Z 1300, B 180}
```

produce un movimiento del TCP en la dirección del eje Z a la posición absoluta 1300 mm y un "cabeceo" del TCP de 180°.

Para el desplazamiento relativo del robot, se utiliza nuevamente la instrucción PTP_REL. Con

PTP_REL {Z 180, B -90}

el robot puede retroceder de nuevo a su posición original. Debe tenerse en cuenta una vez más, que los movimientos relativos después de una interrupción, no deben volverse a seleccionar.

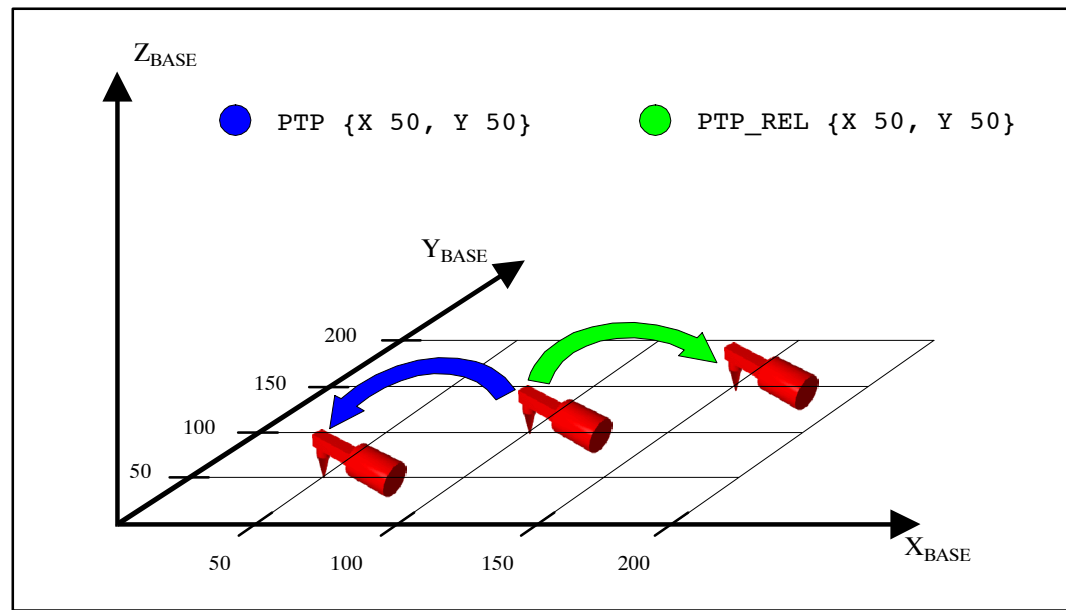


Fig. 17 Diferencia entre las coordenadas cartesianas absolutas y relativas



Para las coordenadas cartesianas resulta posible ejecutar una combinación lógica del frame por medio del operador geométrico, directamente en la instrucción de movimiento. De este modo puede iniciarse, por ejemplo, un desplazamiento hacia el sistema de coordenadas base, sin modificar la variable \$BASE.



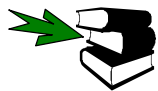
El corrimiento de la base con la ayuda del operador de dos puntos, tiene además una ventaja decisiva respecto a una nueva ocupación de \$BASE:

El corrimiento se efectúa en el paso de movimiento, mientras que una ocupación de \$BASE, debe producirse en cualquier momento antes del paso de movimiento. De este modo, está seleccionada siempre la base correcta para el movimiento también en caso de parada del programa y la selección de paso posterior.

Una nueva ocupación múltiple de \$BASE, como en la secuencia siguiente,

```
...
$BASE = $WORLD
...
PTP POS_1
$BASE = {X 100,Y -200,Z 1000,A 0, B 180,C 45}
PTP POS_2
...
```

conduciría, por el contrario, después de la cancelación del paso de movimiento POS_2 y de la nueva selección del paso POS_1, a un punto destino incorrecto, dado que ahora también se utilizaría la nueva base para el paso de movimiento POS_1. Por cierto, lo mismo ocurre también cuando se produce una parada del primer paso de movimiento, si hay activado un procesamiento en avance correspondiente.



Informaciones adicionales se encuentran en el capítulo **[Programación de movimiento]**, apartado **[Procesamiento en avance]**.

Por este motivo, si es posible, \$BASE y \$TOOL deberían ocuparse solamente una vez, por ejemplo, en la parte de inicialización del programa. Pueden llevarse a cabo otros corrimientos con el operador geométrico.



En caso de un TouchUp con el paquete base de suministro estándar, se guardan automáticamente para cada punto \$BASE y \$TOOL en la lista de datos.

En el ejemplo siguiente se lleva a cabo en la segunda instrucción PTP un corrimiento de las coordenadas del punto destino de 300 mm en la dirección X, -100 mm en la dirección Y, así como un giro de 90° en torno al eje Z.



```
DEF FR_VERS ( )

;----- Sección de declaraciones -----
EXT BAS (BAS_COMMAND :IN,REAL :IN )
DECL AXIS HOME ;Variable HOME del tipo AXIS
DECL FRAME BASE1 ;Variable BASE1 del tipo FRAME

;----- Inicialización -----
BAS (#INITMOV,0 ) ;inicialización de las velocidades,
                  ;aceleraciones, $BASE, $TOOL, etc.
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}
BASE1={FRAME: X 300,Y -100,Z 0,A 90,B 0,C 0}

;----- Sección principal -----
PTP HOME ; desplazamiento COI
; Movimiento respecto al sistema de coordenadas $BASE
PTP {POS: X 540,Y 630,Z 1500,A 0,B 90,C 0,S 2,T 35}
; Movimiento respecto al $BASE-SC corrido en BASIS1
PTP BASE1:{POS: X 540,Y 630,Z 1500,A 0,B 90,C 0,S 2,T 35}
PTP HOME
END
```



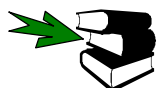
En este ejemplo, además, las declaraciones necesarias de las velocidades, aceleraciones, así como de los sistemas de coordenadas \$BASE y \$TOOL ya no se llevan a cabo “manualmente”. En lugar de ello, se utiliza el paquete base existente de forma estándar “BAS .SRC”. Tiene que darse a conocer al programa, en primer lugar, con la instrucción EXT.

Con la instrucción de inicialización

INI

```
BAS (#INITMOV, 0)
```

se ocupan finalmente todas las variables de sistema importantes con valores estándar.



Informaciones adicionales se encuentran en el capítulo **[Subprogramas y funciones]**, apartado **[Convenciones]**.

COI

Antes de que pueda procesarse un programa, en primer lugar, tiene que establecerse la coincidencia de paso (COI), es decir, la coincidencia de la posición actual del robot con la posición programada. Dado que el movimiento de coincidencia de paso no representa ningún movimiento programado probado, debe llevarse a cabo manteniendo pulsada la tecla de arranque (función de hombre muerto) y con velocidad automática reducida. Al alcanzar la trayectoria programada, el movimiento se detiene y el programa puede continuar al pulsar de nuevo la tecla de arranque.



En el modo de servicio “Automático externo” no se lleva a cabo ningún desplazamiento de coincidencia COI.

Por ello, como primera instrucción de movimiento, se recomienda un recorrido hasta la posición “Home”, en el que el robot se desplaza a una posición inicial no crítica y unívocamente definida en la que se crea también la coincidencia de paso. Al final del programa, el robot debería llevarse nuevamente a esta posición.

S y T

En una indicación POS, los datos “S” y “T” sirven para seleccionar una posición definida en forma unívoca de varias posiciones posibles del robot, para una y la misma posición en el espacio (debido a las singularidades de la cinemática).

Por ello, para la primera instrucción de movimiento, en el caso de utilizar coordenadas cartesianas, es muy importante también programar “Status” y “Turn” para que se defina una posición inicial unívoca. Dado que para los movimientos de trayectoria (véase 3.3) “S” y “T” no se tienen en cuenta, en todos los casos, la primera instrucción de movimiento de un programa (desplazamiento a Home) debe ser una instrucción completa PTP con la indicación de “Status” (estado) y de “Turn” (giro) (o de una instrucción PTP completa con coordenadas de eje).

En las siguientes instrucciones PTP pueden omitirse ahora los datos “S” y “T”, siempre que no sea necesaria una posición determinada del eje, por ejemplo, debido a obstáculos. El robot mantiene el valor de S antiguo y selecciona el nuevo valor T, con el cual resulta el recorrido axial más corto permitido, y que también siempre es el mismo en las distintas ejecuciones de la programación, debido a la única programación de “S” y “T” en el primer paso PTP.



El Status (estado) y el Turn (giro) requieren los dos datos enteros (integer), que deberían indicarse en forma binaria.

Turn

La ampliación de una indicación de posición cartesiana para la indicación de Turn (giro), permite también poder alcanzar ángulos de eje superiores a $+180^\circ$ o inferiores a -180° , sin una estrategia de desplazamiento especial (por ejemplo, puntos auxiliares). Los bits individuales determinan en ejes rotatorios el signo del valor del eje, de la forma siguiente:

Bit x = 0: Angulo del eje $x \geq 0^\circ$

Bit x = 1: Angulo del eje $x < 0^\circ$

Valor	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	$A6 \geq 0^\circ$	$A5 \geq 0^\circ$	$A4 \geq 0^\circ$	$A3 \geq 0^\circ$	$A2 \geq 0^\circ$	$A1 \geq 0^\circ$
1	$A6 < 0^\circ$	$A5 < 0^\circ$	$A4 < 0^\circ$	$A3 < 0^\circ$	$A2 < 0^\circ$	$A1 < 0^\circ$

Tab. 14 Significado de los bits de Turn (giro)

Una indicación T 'B 10011', significa entonces que los ángulos de los ejes 1, 2 y 5 son negativos, pero los ángulos de los ejes 3, 4 y 6, por el contrario, son positivos (pueden omitirse todos los bits de 0 con un valor más elevado).

Status

Con el estado S se tratan las ambigüedades en la posición del eje (véase Fig. 18). Por ello, S depende de la cinemática respectiva del robot.

Ambigüedades

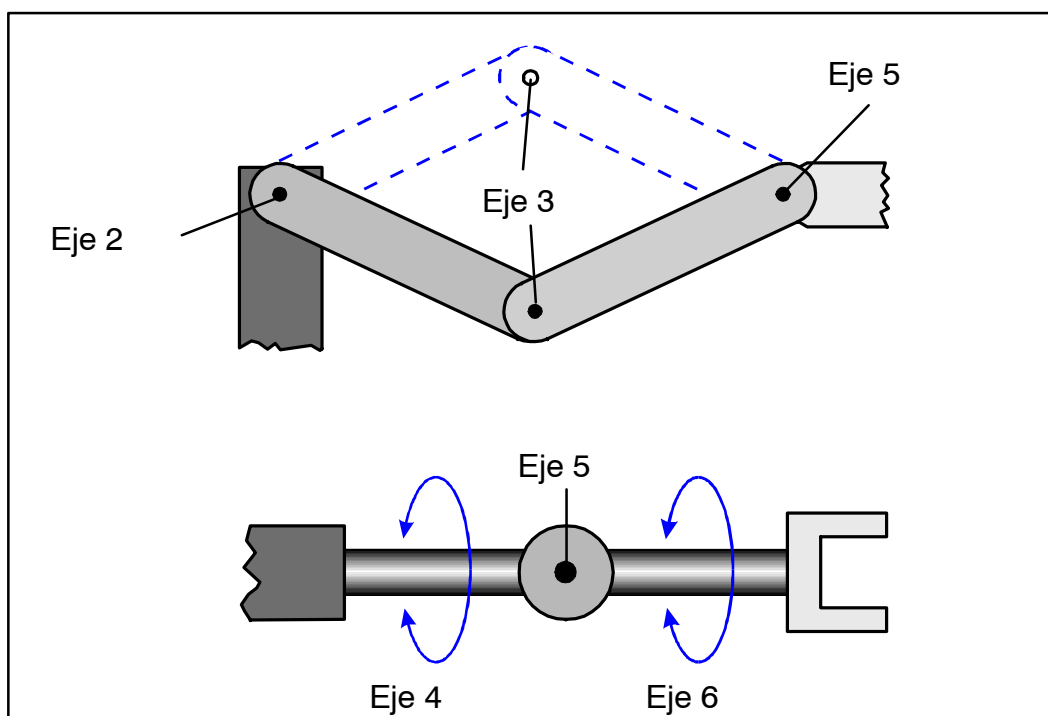


Fig. 18 Ejemplos de cinemáticas de robot ambiguas

El significado de los bits individuales es:

- Bit 0: Posición del punto de la raíz de la mano (área básica/área por sobre la cabeza)
- Bit 1: Configuración del brazo
- Bit 2: Configuración de la muñeca

Los bits para los robots de brazo articulado de 6 ejes se definen según la tabla siguiente:

Valor	Bit 2	Bit 1	Bit 0
0	$0^\circ \leq A5 < 180^\circ$ $A5 < -180^\circ$	$A3 < \phi$ (ϕ depende del tipo de robot)	Area básica
1	$-180^\circ \leq A5 < 0^\circ$ $A5 \geq 180^\circ$	$A3 \geq \phi$ (ϕ depende del tipo de robot)	Area por sobre la cabeza

Tab. 15 Bits de Status (estado) para los robots de brazo articulado de 6 ejes

Gráficamente, puede imaginarse el área básica / por sobre la cabeza, de forma cartesiana. Para ello, se definen los conceptos siguientes:

Punto de la raíz de la muñeca: Punto de intersección de los ejes de la muñeca

Sistema de coordenadas A1: Si el eje 1 está a 0° , es idéntico al sistema de coordenadas \$ROBROOT. En los valores que no sean 0° , se mueve conjuntamente con el eje 1.

Con ello, el área básica / por sobre la cabeza, puede definirse del modo siguiente:

- Si el valor x del punto de la raíz de la muñeca, expresado en el sistema de coordenadas A1, es positivo, el robot se encuentra en el área básica.
- Si el valor x del punto de la raíz de la muñeca, expresado en el sistema de coordenadas A1, es negativo, el robot se encuentra en el área por sobre la cabeza.

El bit 1 indica la posición del brazo. La activación del bit depende del tipo de robot respectivo. En los robots cuyos ejes 3 y 4 se interseccionan, se aplica lo siguiente: el bit 1 tiene el valor 0, cuando el eje 3 es $< 0^\circ$, de lo contrario, el bit 1 = 1. En los robots con un offset entre el eje 3 y el eje 4 (por ejemplo, KR 30, véase Fig. 19), la posición angular en la que se modifica el valor del bit 1, depende de la magnitud de este offset.

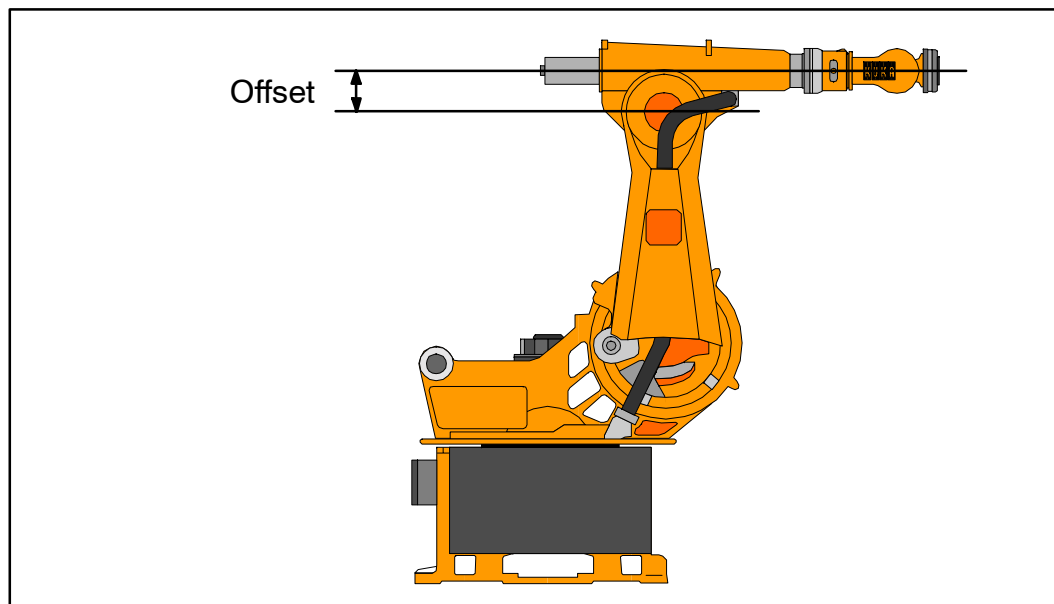


Fig. 19 Offset entre el eje 3 y 4 con un KR 30

En la Fig. 20 se representan los efectos de los bits de Status (estado) en la configuración del robot. El mismo punto en el espacio se alcanzó con cuatro posiciones del robot diferentes. En la primera posición, el robot se encuentra en la posición básica, el eje 5 tiene un valor de 45° aprox., el eje 3 de 80° aprox.

Para la segunda configuración del robot, apenas puede apreciarse alguna diferencia. Únicamente se giró 180° el eje 4 y los otros ejes corrigieron correspondientemente. Mientras que la configuración del brazo se mantiene igual, la configuración de la muñeca se ha modificado: el eje 5 tiene ahora -45° aprox., consecuentemente, el bit de estado 2 es 1.

Desde la posición 2 a 3 se modifica ahora la configuración del brazo. El eje 3 gira a una posición angular de -50° aprox., el bit de estado 1 asume el valor 0.

Finalmente, en la cuarta posición, el robot se encuentra en la posición por sobre la cabeza. Para ello, sobre todo, el eje 1 se giró 180° . El bit de estado 0 se pone a 1.

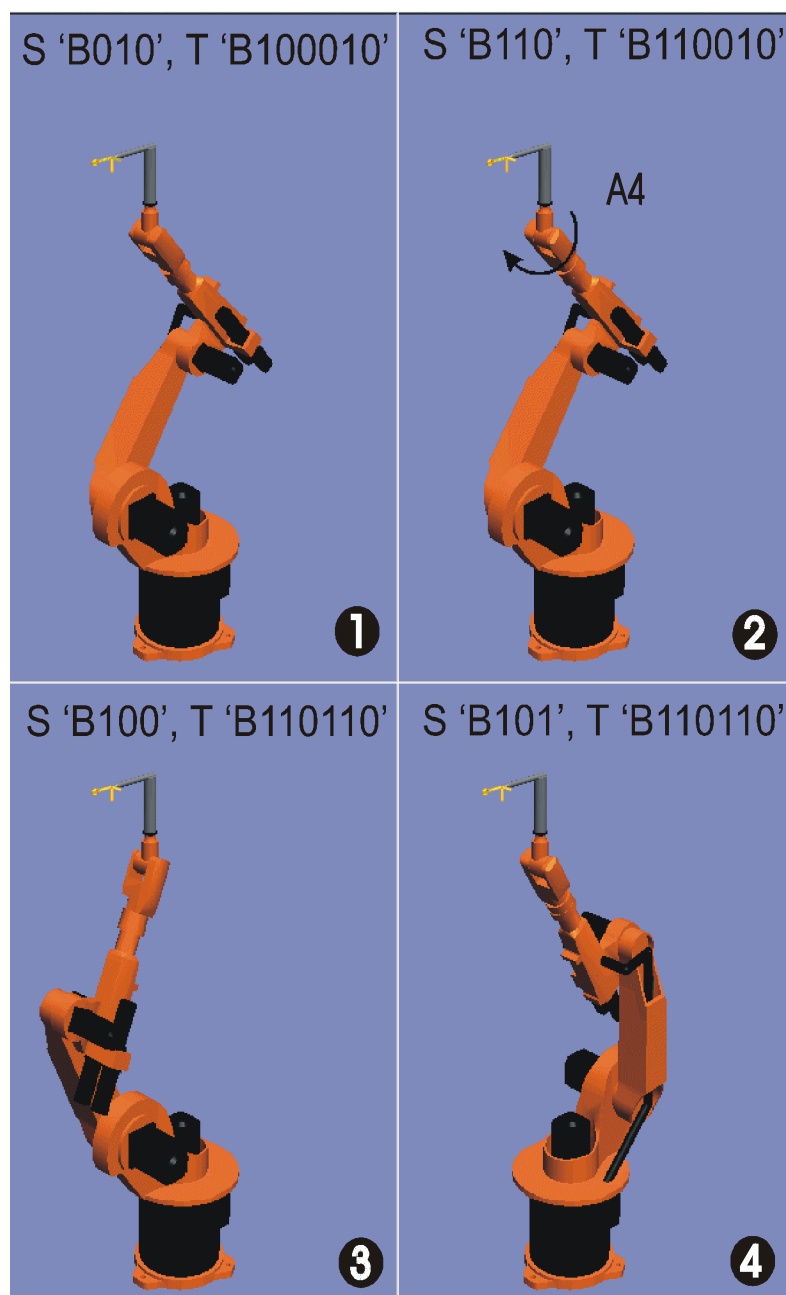


Fig. 20 Efectos de los bits de estado (Status) sobre la posición del robot

3.3 Movimientos sobre trayectorias (CP–Movimientos = Continuous Path)

3.3.1 Velocidad y aceleración

Contrariamente a lo que ocurre en los movimientos PTP, los movimientos sobre trayectoria no sólo están prefijados en la posición de partida y de destino. Adicionalmente, se requiere que la punta de la herramienta del robot se mueva en una trayectoria lineal o circular entre estos puntos.

Por ello, las velocidades y aceleraciones indicadas ya no se refieren a los ejes individuales, sino al movimiento del TCP. La punta de la herramienta se mueve, por ello, con una velocidad exactamente definida. Las velocidades y aceleraciones deben programarse para la translación, el ángulo de basculamiento y el ángulo de giro. Tab. 16 proporciona una visión general sobre las variables del sistema a programar y sus unidades.

	Nombre de variable	Tipo de datos	Unidad	Función
Velocidades	\$VEL.CP	REAL	m/s	Velocidad de trayectoria
	\$VEL.ORI1	REAL	°/s	Velocidad de basculamiento
	\$VEL.ORI2	REAL	°/s	Velocidad de giro
Aceleraciones	\$ACC.CP	REAL	m/s ²	Aceleración de trayectoria
	\$ACC.ORI1	REAL	°/s ²	Aceleración de basculamiento
	\$ACC.ORI2	REAL	°/s ²	Aceleración de giro

Tab. 16 Variables del sistema para velocidades y aceleraciones



Como mínimo, uno de los componentes del movimiento (translación, basculamiento y giro), se desplaza en cada instante de la ejecución del movimiento con aceleración o velocidad programada. Los componentes no dominantes se adaptan con sincronización de fase.



También las velocidades y aceleraciones para los movimientos de trayectoria se ocupan por defecto con los valores máximos definidos en los datos de máquina o en \$CONFIG.DAT, en el llamado de la secuencia de inicialización del paquete base.

Además, en movimientos de trayectoria se controlan las velocidades y aceleraciones de los ejes, y en caso de sobrepasar los valores límites de control definidos en las variables del sistema \$ACC_ACT_MA y \$VEL_ACT_MA, se dispara una reacción de parada y se emite un mensaje de fallo. De forma estándar, estos valores límites se encuentran en 250% del valor nominal de aceleración del eje (Acceleration) y 110% del valor nominal de la velocidad del eje (Velocity). Estos rangos de control valen para todos los modos de servicio y para el modo de movimiento manual.

Existe la posibilidad, a través de la variable del sistema \$CP_VEL_TYPE, de reducir el avance de los ejes, y con ello, la aceleración y la velocidad, para evitar el disparo de los valores límites de control (reacción de parada). Como declaración por defecto, ha sido ocupada la variable con #CONSTANT, es decir, la reducción no está activa en el servicio con el programa. Si se desea esta función en el modo de servicio T1, se debe activar la variable #VAR_T1 (en T1 se utilizan valores menores de velocidad y aceleración de los ejes) y en todos los modos de servicio #VAR_ALL. En el modo de servicio manual la reducción está siempre activa.

Con la variable del sistema \$CPVELREDMELD se consigue en ambos modos de servicio de test un mensaje, en el caso que la velocidad de trayectoria es reducida. Para ello, la variable para el mensaje debe estar puesta en "1".

3.3.2 Control de la orientación

Si durante el movimiento sobre la trayectoria la orientación de la herramienta debe ser modificada en el espacio, puede ajustarse el tipo de control orientación con la ayuda de las variables de sistema \$ORI_TYPE (véase Fig. 21):

- \$ORI_TYPE = #CONSTANT)

Durante el movimiento de la trayectoria, la orientación se mantiene constante; para el punto final se ignora la orientación programada y se utiliza la del punto inicial.

- \$ORI_TYPE = #VAR

Durante el movimiento de la trayectoria, la orientación cambia continuamente desde la orientación inicial a la final. El valor se define también durante la inicialización por medio de `BAS (#INITMOV, 0)`.

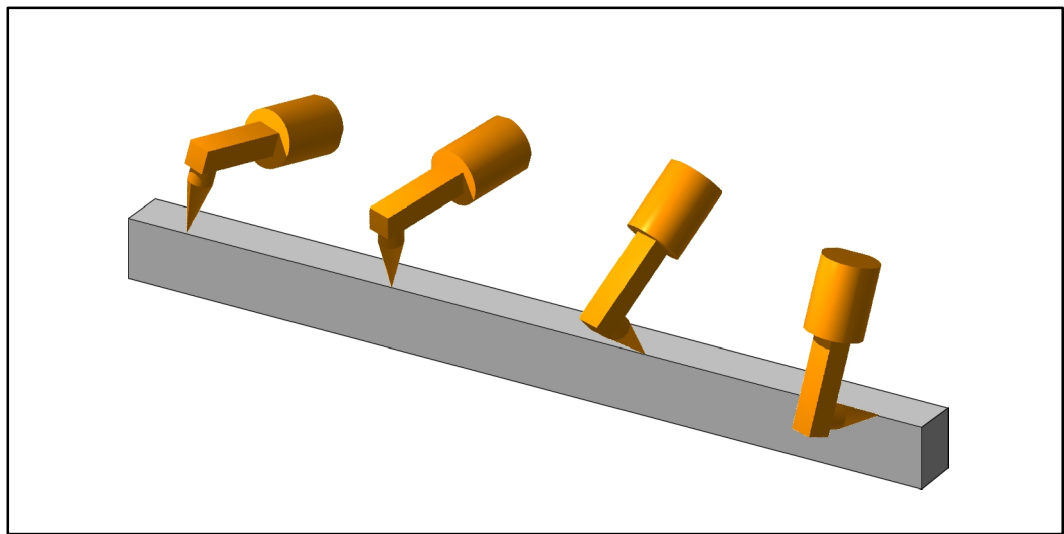


Fig. 21 Modificación de la orientación durante un movimiento lineal

Durante los movimientos circulares puede seleccionarse, adicionalmente a la orientación constante y variable, entre la orientación referida al espacio y a la trayectoria:

- $\$CIRC_TYPE = \#BASE$ Control de orientación referido al espacio durante el movimiento circular. Este valor se define también durante la inicialización por medio de $BAS(\#INITMOV, 0)$.
- $\$CIRC_TYPE = \#PATH$ Control de orientación referido a la trayectoria durante el movimiento circular.

Constante +
referencia a la
trayectoria

En un control de orientación referido a la trayectoria, el eje longitudinal de la herramienta se mantiene relativo al plano del círculo y a la tangente del mismo. Esta relación puede explicarse con la ayuda del denominado trípole que acompaña a la trayectoria (véase Fig. 22).

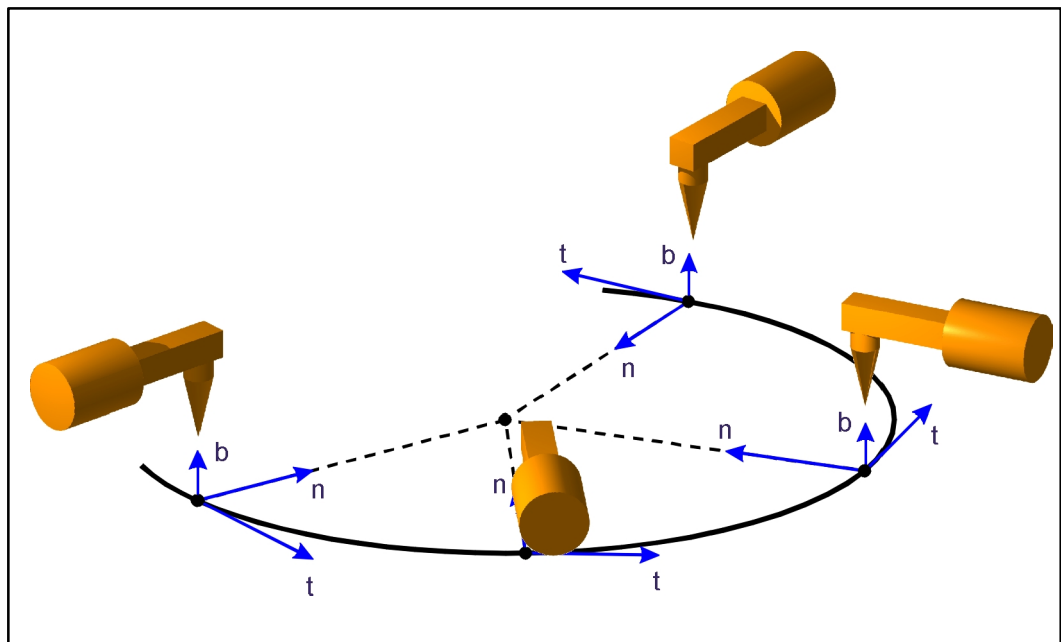


Fig. 22 Control de orientación constante referido a la trayectoria en los movimientos circulares

El trípole que acompaña a la trayectoria se compone del vector tangente a la circunferencia **t**, del vector de la normal **n** y del vector de la binormal **b**. La orientación de la herramienta en la Fig. 22 se guía posteriormente con el trípole que acompaña a la trayectoria, en el segmento de la circunferencia. Para las posiciones de la herramienta no se produce ninguna modificación de la orientación en referencia al trípole que acompaña la trayectoria. Este es, entre otros, un requisito importante para la soldadura al arco.

En el ejemplo mostrado, la orientación de la herramienta, relativa al trípole que acompaña la trayectoria, no se modifica durante el movimiento desde el punto inicial hacia el de destino ($\$ORI_TYPE=\#CONST$).

Variable +
referencia a la
trayectoria

Si se desea una modificación de la orientación referida a la trayectoria entre la posición de partida y la de destino (\$ORI_TYPE=#VAR), se ejecuta la misma en relación al tríode que acompaña la trayectoria, mediante el giro y basculamiento superpuesto (véase Fig. 23). El control de la orientación en el tríode que acompaña la trayectoria es totalmente análoga en los movimientos circulares a la orientación para los movimientos lineales.

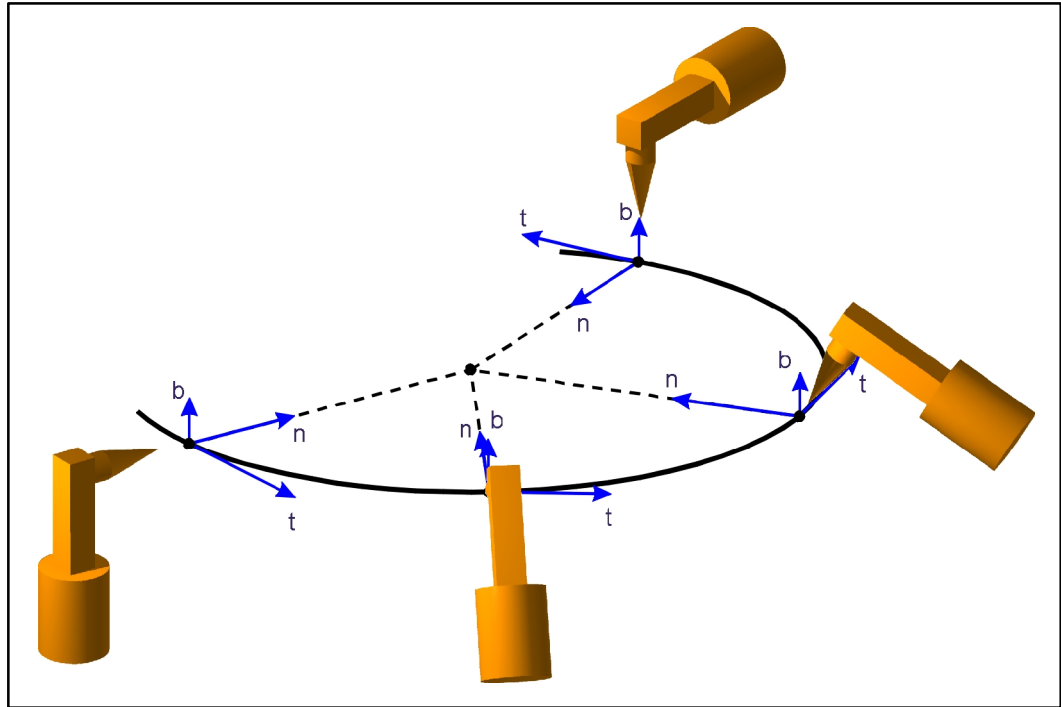


Fig. 23 Control de orientación variable referida a la trayectoria en los movimientos circulares

Constante +
referencia al
espacio

En el control de la orientación referido al espacio, la orientación se realiza en relación al sistema base actual (\$BASE).

El control de orientación referido al espacio, es especialmente indicado para las aplicaciones en donde el énfasis principal se sitúa en el movimiento sobre la trayectoria, es decir, el control de la punta de la herramienta en la trayectoria circular. Especialmente en las aplicaciones con una pequeña modificación de la orientación entre el punto inicial y de destino, o en las aplicaciones con una orientación exactamente constante en el espacio (véase Fig. 24) durante un movimiento circular (por ejemplo, aplicación de adhesivo, con una boquilla de rotación simétrica).

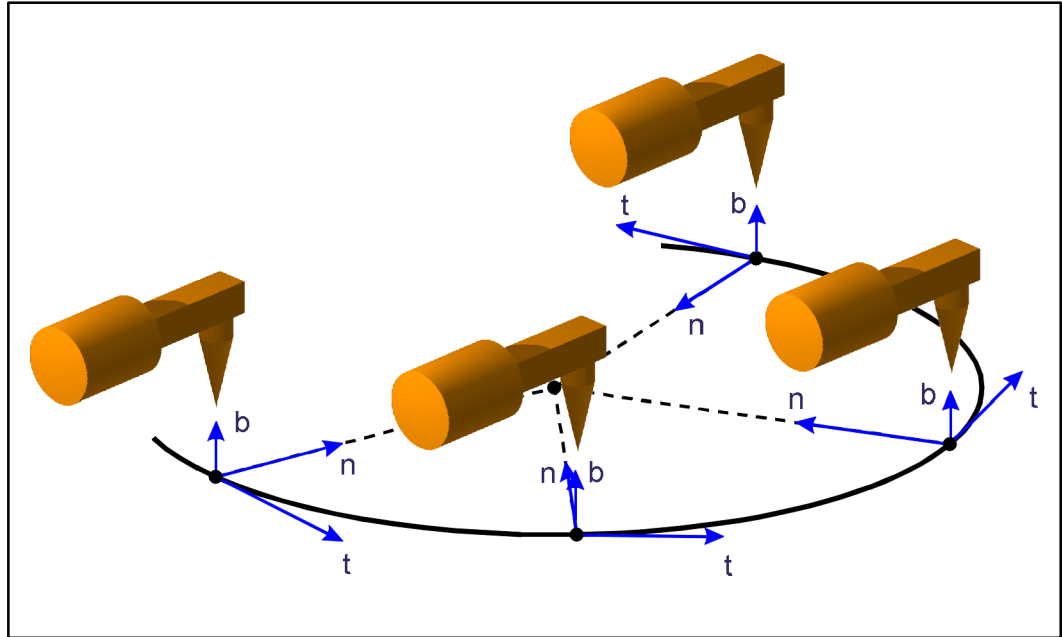


Fig. 24 Control de orientación referida al espacio constante en los movimientos circulares

Variable +
referencia al
espacio

Se ejecuta nuevamente un cambio de la orientación referida al espacio ($\$ORI_TYPE=\#VAR$) entre la posición de partida y la de destino, mediante la superposición de los movimientos de basculamiento y de giro (véase Fig. 25). Sin embargo, en este caso, en relación al sistema de coordenadas base.

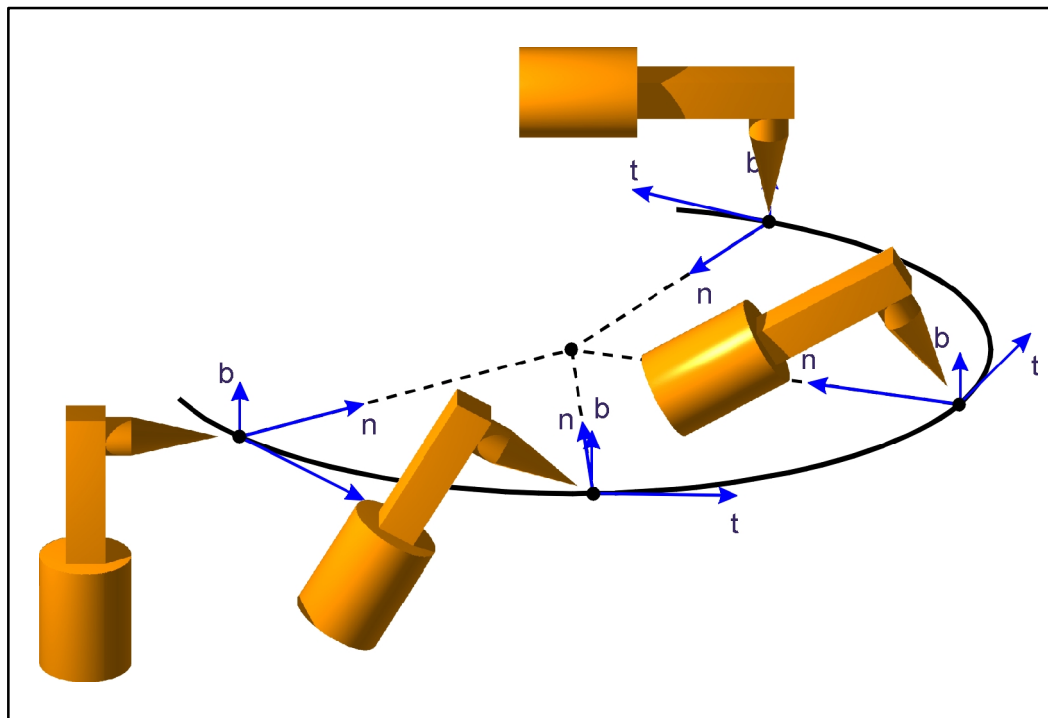


Fig. 25 Control de orientación variable referida al espacio en los movimientos circulares

En la Tab. 17 se presentan nuevamente en una lista los preajustes de las variables del sistema para el control de orientación en los movimientos sobre la trayectoria:

	en el sistema	mediante BAS (#INITMOV, 0)
$\\$ORI_TYPE$	$\#VAR$	
$\\$CIRC_TYPE$	$\#PATH$	$\#BASE$

Tab. 17 Preajustes de $\$ORI_TYPE$ y $\$CIRC_TYPE$

3.3.3 Movimientos lineales

LIN

Para un movimiento lineal, la KR C... calcula una recta desde la posición actual (en el programa es el último punto programado) hasta la posición que se indicó en la instrucción de movimiento.

La programación de un movimiento lineal se efectúa mediante las palabras claves LIN o LIN_REL en combinación con la indicación de destino, es decir, de forma análoga a la programación PTP. Para los movimientos lineales, la posición de destino se indica de forma cartesiana. Es decir, solamente se admiten los tipos de datos FRAME o POS.

En los movimientos lineales, el estado de los ángulos del punto final debe ser igual al del punto inicial. Por ello, se ignora la indicación del estado (status) y del giro (Turn) del tipo de datos POS. Por ello, antes de la primera instrucción LIN tiene que estar ya programado un movimiento PTP con la indicación completa de las coordenadas (por ejemplo, desplazamiento HOME).

La ocupación necesaria para los movimientos sobre la trayectoria de las variables velocidad y aceleración, así como la definición del sistema de coordenadas Tool y Base, se lleva a cabo en el programa del ejemplo siguiente nuevamente por medio de la rutina de inicialización BAS.SRC.



```

DEF LIN_BEW ( )

;----- Sección de declaraciones -----
EXT BAS (BAS_COMMAND: IN, REAL: IN)
DECL AXIS HOME      ;Variable HOME del tipo AXIS

;----- Inicialización -----
BAS (#INITMOV, 0)    ;inicialización de las velocidades,
                    ;aceleraciones, $BASE, $TOOL, etc.
HOME = {AXIS: A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}

;----- Sección principal -----
PTP HOME ; desplazamiento COI
PTP {A5 30}

; Movimiento lineal hacia la posición indicada, la orientación
; se modifica continuamente a la orientación destino
LIN {X 1030,Y 350,Z 1300,A 160,B 45,C 130}

; Movimiento lineal en el plano Y-Z, S y T se ignoran
LIN {POS: Y 0,Z 800,A 0,S 2,T 35}

; Movimiento lineal hacia la posición indicada, la orientación
; no se modifica con ello
$ORI_TYPE=#CONST
LIN {FRAME: X 700,Y -300,Z 1000,A 23,B 230,C -90}

; la orientación no se modifica
LIN {FRAME: Z 1200,A 90,B 0,C 0}

; Movimiento relativo a lo largo del eje X
LIN_REL {FRAME: X 300}

PTP HOME
END

```

3.3.4 Movimientos circulares

CIRC

Para la definición unívoca de una circunferencia o arco de circunferencia en el espacio, son necesarios 3 puntos, diferentes entre sí y que no se encuentran sobre una recta.

El punto inicial de un movimiento circular se indica nuevamente mediante la posición actual, como en los casos PTP o LIN.

Para la programación de un movimiento circular con las instrucciones CIRC o CIRC_REL, por ello, debe definirse también, junto al punto destino, un punto intermedio. Para el cálculo de la trayectoria de movimiento mediante la unidad de control, del el punto intermedio solamente se evalúan los componentes traslacionales (X, Y, Z). La orientación de la punta de la herramienta se modifica continuamente según el control de orientación desde el punto inicial hasta el punto final o se mantiene constante.

CA



Adicionalmente a la posición intermedia y de destino (final), puede programarse un ángulo circular con la opción CA (ángulo circular). La geometría del arco de circunferencia, se especifica como explicado anteriormente, mediante el punto inicial, auxiliar y de destino. Sin embargo, la posición de destino a alcanzar realmente en el arco de circunferencia se especifica mediante el ángulo de circunferencia programado. Esto es muy útil para la reprogramación de la posición de destino sin modificar la geometría circular.

El arco de circunferencia a recorrer puede prolongarse o acortarse de acuerdo con el ángulo de circunferencia. La orientación de destino programada se alcanza en el punto de destino real. Mediante el signo del ángulo de circunferencia puede especificarse el sentido de giro, es decir, la dirección en que debe recorrerse el arco de circunferencia (véase Fig. 26):

- $CA > 0^\circ$ en el sentido programado (punto inicial \rightarrow punto intermedio \rightarrow punto de destino)
- $CA < 0^\circ$ en contra del sentido programado (punto inicial \rightarrow punto de destino \rightarrow punto intermedio)



El valor del ángulo de circunferencia no está limitado. Especialmente, pueden programarse también circunferencias completas ($> 360^\circ$).

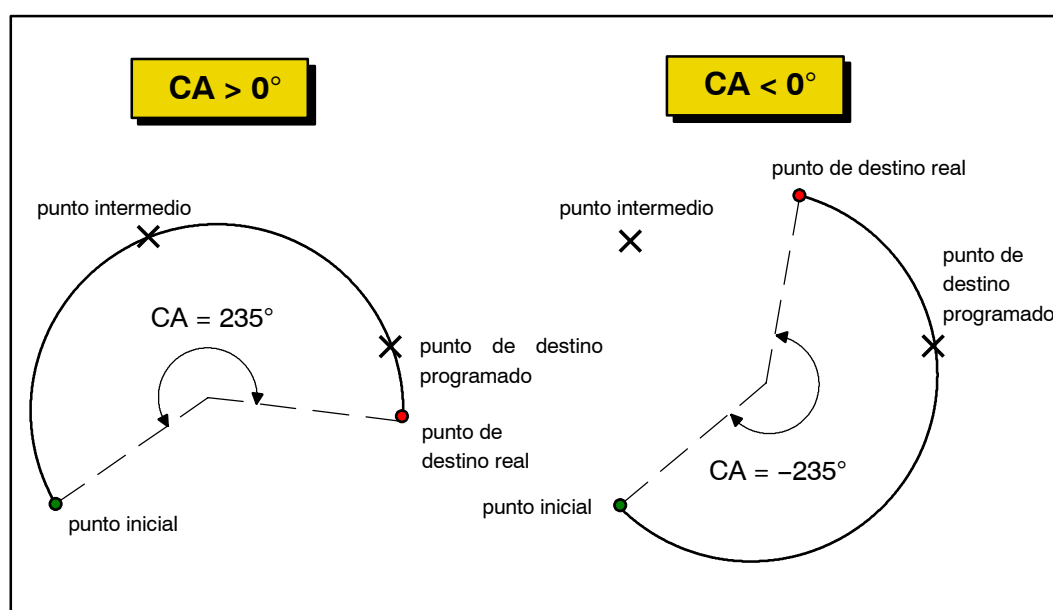


Fig. 26 Influencia de la opción CA en la instrucción CIRC o CIRC_REL

Las indicaciones relativas para la posición intermedia y de destino (CIRC_REL) hacen referencia respectivamente, a la posición inicial. Las indicaciones de posición específicas del eje no son admisibles como en los movimientos LIN. Del mismo modo, \$BASE y \$TOOL deben estar totalmente asignados antes de la ejecución de un movimiento circular.



```

DEF CIRC_BEW ( )

;----- Sección de declaraciones -----
EXT BAS (BAS_COMMAND :IN,REAL :IN )
DECL AXIS HOME

;----- Inicialización -----
BAS (#INITMOV,0 ) ;inicialización de las velocidades,
                  ;aceleraciones, $BASE, $TOOL, etc.
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}

;----- Sección principal -----
PTP HOME ;desplazamiento COI
PTP {POS: X 980,Y -238,Z 718,A 133,B 66,C 146,S 6,T 50}

; Control de orientación variable referida al espacio (preajuste)
CIRC {X 925,Y -285,Z 718},{X 867,Y -192,Z 718,A 155,B 75,C 160}

; control de orientación constante referida al espacio
; punto destino especificado mediante la indicación del ángulo
$ORI_TYPE=#CONST
CIRC {X 982,Y -221,Z 718,A 50,B 60,C 0},{X 1061,Y -118,Z 718,
A -162,B 60,C 177}, CA 300.0

; control de orientación constante referida a la trayectoria
; punto final mediante indicación del ángulo (hacia atrás)
$CIRC_TYPE=#PATH
CIRC {X 867,Y -192,Z 718},{X 982,Y -221,Z 718,A 0}, CA -150

$ORI_TYPE=#VAR
LIN {A 100} ; Reorientación del TCP

; control de orientación variable referida a la trayectoria
CIRC {X 963.08,Y -85.39,Z 718},{X 892.05,Y 67.25,Z 718.01,
A 97.34,B 57.07,C 151.11}

; movimiento circular relativo
CIRC_REL {X -50,Y 50},{X 0,Y 100}

PTP HOME
END

```

3.4 Procesamiento en avance

Una característica de rendimiento absolutamente fundamental para un robot industrial, es la rapidez con la que puede realizar sus trabajos. Junto a la dinámica del robot, es de importancia decisiva también la eficacia del procesamiento del programa de usuario, que junto a los movimientos, consta también de instrucciones aritméticas y de control de los periféricos.

Un procesamiento más rápido del programa puede conseguirse

- mediante la reducción de la duración del movimiento y
- mediante la reducción del tiempo de parada entre los movimientos.

Con las condiciones marginales prefijadas, como las velocidades y aceleraciones máximas del eje, esto se alcanza mediante el posicionamiento aproximado de los movimientos en un tiempo óptimo.



Informaciones adicionales se encuentran en el capítulo **[Programación de movimiento]**, apartado **[Movimientos con posicionamiento aproximado]**.

El tiempo de parada entre los movimientos puede reducirse cuando las instrucciones aritméticas y lógicas intensivas se elaboran entre los pasos de movimiento, durante el movimiento del robot, es decir, se procesan en avance (las instrucciones “avanzan” delante del movimiento).

\$ADVANCE

A través de la variable del sistema \$ADVANCE, pueden especificarse cuantos pasos de movimiento en avance deben adelantarse, como máximo, respecto a la ejecución principal (es decir, al paso de movimiento procesado actualmente). El puntero principal visible en la superficie de operación durante el procesamiento del programa, muestra siempre el paso de movimiento en ejecución.

Por el contrario, el puntero de avance no es visible y puede mostrar tanto instrucciones completamente ejecutadas por la unidad de control, como también pasos de movimiento solamente preparados por la misma, y que se ejecutan posteriormente en la ejecución principal (véase Fig. 27).

```

13
14  $ADVANCE=1
15
16  → LIN {X 1620,Y 0,Z 1910,A 0,B 90,C 0} ← Puntero principal
17
18  STROM={STROM*1.2}/0.5
19  FOR I=1 TO 6
20    $VEL_AXIS[I]=60
21    $ACC_AXIS[I]=35
22  ENDFOR
23
24  PTP PUNKT6 ← El puntero de avance se encuentra en
25              esta posición; $ADVANCE = 1
26  SPANNUNG=110
27
28  PTP PUNKT7
29
30
31

```

Fig. 27 Punteros principal y de avance

En la sección de programa anterior, el avance está definido en 1 y el puntero principal se encuentra en la línea 16 (es decir, se ejecuta el movimiento LIN). Un procesamiento en avance de 1 significa que las instrucciones de las líneas 16 a 22 se elaboraron completamente en forma paralela a la ejecución del movimiento, y los datos de movimiento para los movimientos PTP en la línea 24 ya se están preparando.



Para permitir un posicionamiento aproximado, debe estar declarado, como mínimo, un avance de 1. (La variable \$ADVANCE posee siempre, de forma estándar, el valor "3". Se tienen, como máximo, 5 pasos en el procesamiento en avance.)
En un subprograma de interrupción no es posible el procesamiento en avance. La unidad de control confecciona siempre los programas de interrupción por líneas, por lo que no es posible un posicionamiento aproximado en ellos.

Preajustes de \$ADVANCE

	en el sistema	mediante BAS (#INITMOV, 0)
\$ADVANCE	0	3

**Parada
autom.
procesa-
miento
avance** del
en

Instrucciones y datos que pudieran influenciar la periferia (por ej. instrucciones de entrada / salida), o que se refieren al estado actual del robot, generan una **Parada del procesamiento en avance** (ver Tab. 18). Esto es necesario para asegurar la secuencia correcta en ejecución y tiempo, de las instrucciones y los movimientos del robot.

Instrucciones	ANOUT ON	ANOUT OFF		
	ANIN ON	ANIN OFF		
	DIGIN ON	DIGIN OFF		
	PULSE			
	HALT	WAIT		
	CREAD	CWRITE	COPEN	CCLOSE
	SREAD	SWRITE		
	Combinaciones CP-PTP sin posicionamiento aproximado			
Instrucciones en combinación con una interrupción	END (en caso que en el módulo no ha sido definida una interrupción global)			
	INTERRUPT DECL (en caso que la interrupción haya sido declarada)			
	RESUME sin BRAKE			
Variables del sistema más utilizadas	\$ANOUT[Nr]	\$ANIN[Nr]		
	\$DIGIN1	\$DIGIN2	...	\$DIGIN6
	\$OUT[Nr]	\$IN[Nr]		
	\$AXIS_ACT	\$AXIS_BACK	\$AXIS_FOR	\$AXIS_RET
	\$POS_ACT	\$POS_BACK	\$POS_FOR	\$POS_RET
	\$AXIS_INC	\$AXIS_INT	\$POS_ACT_MES	\$POS_INT
	\$TORQUE_AXIS	\$ASYNC_AXIS		
	\$TECH[X].MODE, \$TECH[X].CLASS en determinadas operaciones			
	\$LOAD, \$LOAD_A1, \$LOAD_A2, \$LOAD_A3 (en caso de tratarse de un robot de exactitud absoluta con cambio de carga)			

otras variables del sistema	\$ALARM_STOP \$AXIS_ACTMOD \$INHOME_POS \$ON_PATH \$EM_STOP \$EXTSTARTTYP \$REVO_NUM \$SAFETY_SW \$ACT_TOOL \$PAL_MODE \$ACT_BASE \$ACT_EX_AX \$OV_PRO \$WORKSPACE \$IBUS_OFF \$IBUS_ON \$ASYNC_EX_A X_DECOUPLE
Variables importadas	todas, con el acceso
Otros	Si entre pasos de posicionamiento aproximado se efectúa un cambio de filtro, se genera una parada del procesamiento en avance.

Tab. 18 Instrucciones y variables que activan una parada automática del procesamiento en avance

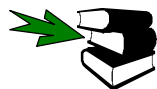
CONTINUE

En los casos de aplicación en los que tiene que impedirse esta parada del procesamiento en avance, inmediatamente antes de la instrucción correspondiente, debe programarse la instrucción **CONTINUE**. La unidad de control continúa entonces con el avance. Sin embargo, el efecto está limitado a la línea de programa siguiente (¡también línea en blanco!).



Si por el contrario, Ud. desea detener el procesamiento en avance en un lugar determinado sin tener que modificar para ello la variable de sistema **\$ADVANCE**, puede ayudar con un pequeño truco: simplemente programe en este lugar un tiempo de espera de 0 segundos. La instrucción **WAIT** activa una parada automática del procesamiento en avance, sin otro efecto en particular:

WAIT SEC 0



Informaciones adicionales se encuentran en el capítulo **[Control de ejecución de programa]**, apartado **[Tiempos de espera]**.

3.5 Movimientos con posicionamiento aproximado

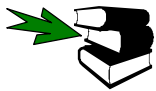
Contorno de
posicionamiento
aproximado

Para aumentar la velocidad puede utilizarse el posicionamiento aproximado en aquellos casos en donde no es necesario posicionar exactamente sobre el punto. El robot acorta el recorrido, tal como se muestra en la Fig. 28.

El contorno de posicionamiento aproximado lo genera automáticamente la unidad de control. El programador solamente influye en el comienzo y el final del posicionamiento aproximado. Para el cálculo del paso de posicionamiento aproximado, el control necesita los datos del punto inicial, del punto de posicionamiento aproximado y del punto final.

Para permitir un posicionamiento aproximado, el procesamiento en avance (\$ADVANCE) debe estar definido, como mínimo, en 1. En el caso de que el procesamiento en avance sea demasiado pequeño, aparece el mensaje "Aproximación no posible" y el desplazamiento hasta los puntos se realiza en forma exacta.

Si programa una instrucción después de una instrucción de posicionamiento aproximado que activa una parada automática del procesamiento en avance (véase Tab. 18), no es posible efectuar un posicionamiento aproximado.



Informaciones adicionales a la instrucción TRIGGER (disparo) se encuentran en el capítulo **[Trigger – Acciones de conmutación referentes a la trayectoria]**.

Posicion-
amiento
aproximado

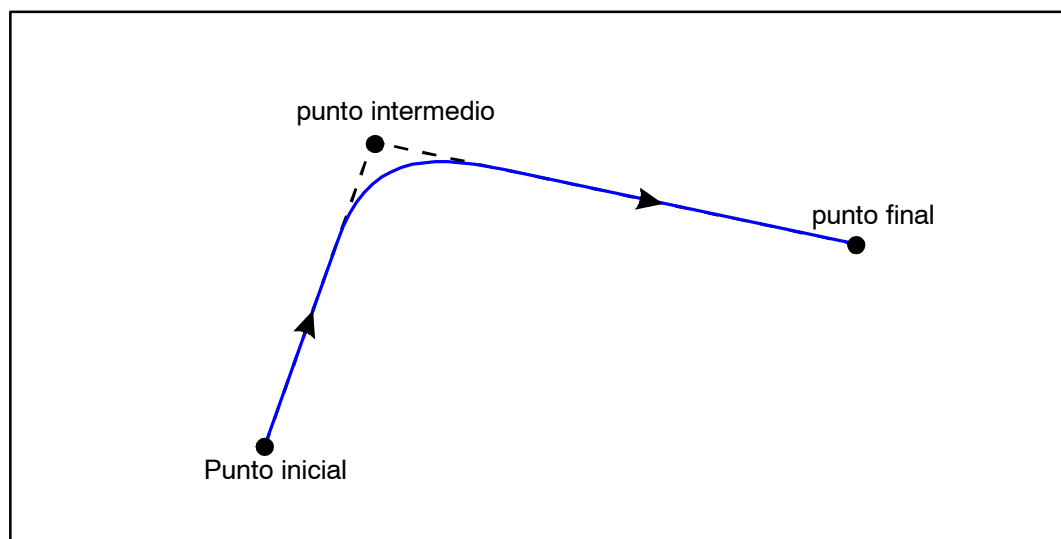


Fig. 28 Posicionamiento aproximado de los puntos auxiliares

3.5.1 Aproximación PTP-PTP

Para el posicionamiento aproximado PTP, la unidad de control calcula las distancias a recorrer axialmente en la zona de aproximación y planifica para cada eje los perfiles de velocidad para cada eje, que aseguran las transiciones tangenciales de los pasos individuales para asegurar al contorno de posicionamiento aproximado.

Comienzo de la aproximación El posicionamiento aproximado se inicia cuando el último eje (= guía) sobrepasa por defecto un ángulo determinado hacia el punto de posicionamiento aproximado. En los datos de la máquina se ha predefinido un ángulo para cada eje:

```
$APO_DIS_PTP[ 1 ] = 90
```

```
⋮
```

```
$APO_DIS_PTP[ 6 ] = 90
```

Mediante \$APO.CPTP puede indicarse en el programa el comienzo del posicionamiento aproximado en forma de porcentaje de estos valores máximos. Por ejemplo:

```
$APO.CPTP = 50
```

En este caso se inicia el posicionamiento aproximado cuando el primer eje ha recorrido un ángulo residual de 45° (50% de 90°) hacia el punto de posicionamiento aproximado. El final del posicionamiento aproximado se produce cuando el primer eje ha recorrido un ángulo de 45° desde el punto de posicionamiento aproximado.

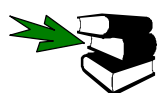
Cuanto mayor es \$APO.CPTP, más se redondea la trayectoria.

Básicamente, el posicionamiento aproximado no puede producirse por detrás del centro del paso. En este caso, el sistema se limita automáticamente al centro del paso.

C_PTP El posicionamiento aproximado de un punto se indica en la instrucción PTP adjuntando la palabra clave C_PTP:

```
PTP PUNTO4 C_PTP
```

También el paso de posicionamiento aproximado PTP se ejecuta en el tiempo óptimo, es decir, mientras se efectúa el posicionamiento aproximado, se desplaza siempre como mínimo, un eje con el valor límite programado para la aceleración o velocidad. Simultáneamente, se asegura para cada eje que no se superan los momentos admitidos para los engranajes reductores y motores. El perfil de marcha más elevado ajustado de forma estándar, garantiza un control de los movimientos invariante en velocidad y en aceleración.



Informaciones adicionales se encuentran en el capítulo **[Programación de movimiento]**, apartado **[Perfil de marcha más elevado]**.

En el ejemplo siguiente puede verse el efecto de la instrucción de posicionamiento aproximado y las variables \$APO.CPTP. La trayectoria recorrida se representa en Fig. 29 en el plano x-y. Especialmente, puede apreciarse también que los movimientos PTP del TCP no se recorren sobre una recta entre los puntos de destino.



```

DEF  UEBERPTP ( )

;----- Sección de declaraciones -----
EXT  BAS (BAS_COMMAND :IN,REAL :IN )
DECL AXIS HOME

;----- Inicialización -----
BAS (#INITMOV,0 ) ;inicialización de las velocidades,
                  ;aceleraciones, $BASE, $TOOL, etc.
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}

;----- Sección principal -----
PTP  HOME ; desplazamiento COI

PTP {POS:X 1159.08,Y -232.06,Z 716.38,A 171.85,B 67.32,C
162.65,S 2,T 10}

;Posicionamiento aproximado del punto
PTP {POS:X 1246.93,Y -98.86,Z 715,A 125.1,B 56.75,C 111.66,S 2,T
10} C_PTP

PTP {POS:X 1109.41,Y -0.51,Z 715,A 95.44,B 73.45,C 70.95,S 2,T
10}

;Posicionamiento aproximado de dos puntos
$APO.CPTP=20
PTP {POS:X 1296.61,Y 133.41,Z 715,A 150.32,B 55.07,C 130.23,S
2,T 11} C_PTP
PTP {POS:X 988.45,Y 238.53,Z 715,A 114.65,B 50.46,C 84.62,S 2,T
11} C_PTP

PTP {POS:X 1209.5,Y 381.09,Z 715,A -141.91,B 82.41,C -159.41,S
2,T 11}

PTP  HOME
END

```

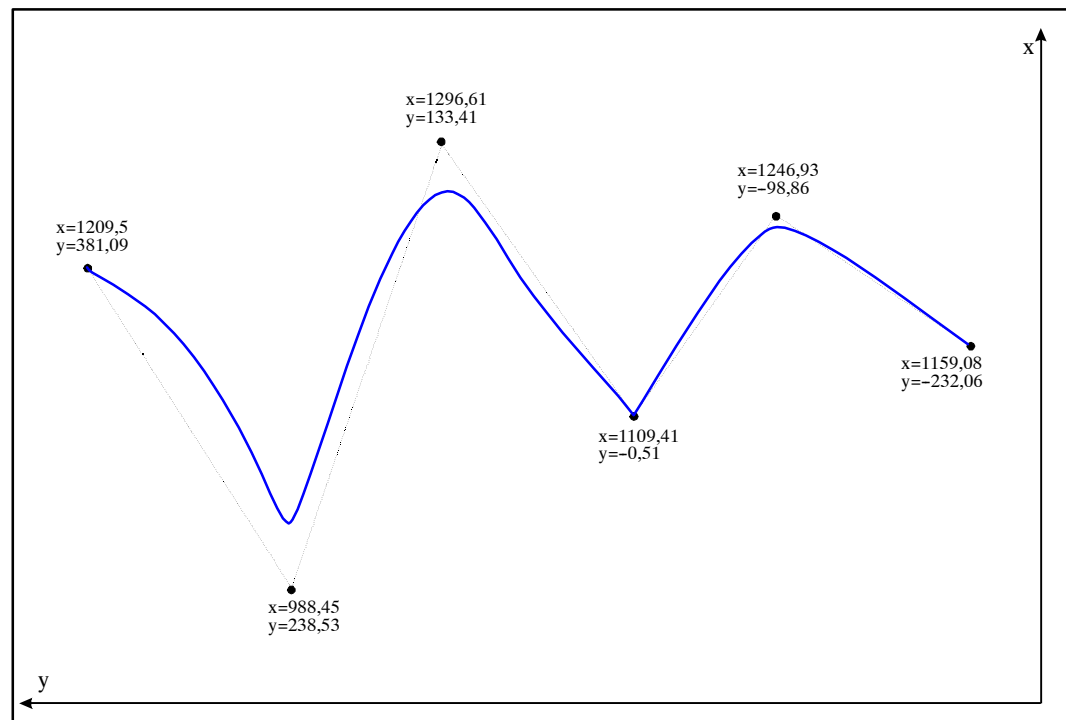


Fig. 29 Ejemplo para el posicionamiento aproximado PTP-PTP



Dado que la trayectoria de un movimiento PTP en general, no es una recta ni está en un plano en el espacio, no puede representarse estrictamente como se muestra en la Fig. 29. A pesar del hecho de que el valor z es igual en todos los puntos del ejemplo, no todos los puntos de la trayectoria de movimiento están en el plano $z=715$ mm. Por lo tanto, la trayectoria reflejada es únicamente una proyección de la trayectoria real en el plano $x-y$.

3.5.2 Aproximación LIN-LIN

Para el movimiento continuo a lo largo de trayectorias complejas, existe el requisito de efectuar el posicionamiento aproximado también entre pasos individuales lineales. Los diferentes movimientos de orientación en los pasos LIN, deben pasar gradualmente unos dentro de otros.



Como contorno de posicionamiento aproximado, la unidad de control calcula una trayectoria parabólica, dado que esta forma de contorno se aproxima muy bien al recorrido de la trayectoria de los pasos individuales con el mejor aprovechamiento de las reservas de aceleración en la zona de aproximación.

Comienzo de la aproximación

Para la especificación del comienzo del posicionamiento aproximado, se dispone de tres variables predefinidas (véase Tab. 19):

Variable	Tipo datos	Unidad	Significado	Palabra clave en la instrucción
\$APO.CDIS	REAL	mm	Criterio de distancia de translación	C_DIS
\$APO.CORI	REAL	°	Criterio de orientación	C_ORI
\$APO.CVEL	INT	%	Criterio de velocidad	C_VEL

Tab. 19 Variables del sistema para la especificación del comienzo del posicionamiento aproximado

Criterio de distancia	de	Puede asignarse un valor porcentual a la variable \$APO.CDIS . Si el posicionamiento aproximado se activa por medio de esta variable, la unidad de control abandona el contorno individual a partir del momento cuando sobrepasa por defecto la distancia hasta el punto final con valor \$APO.CDIS .
Criterio de orientación	de	Puede asignarse un valor porcentual a la variable \$APO.CORI . El contorno individual se abandona, en este caso, a partir del momento cuando el ángulo de orientación dominante (basculamiento o giro del eje longitudinal de la herramienta) pasa por debajo de la distancia al punto de posicionamiento aproximado programado, especificada en \$APO.CORI .
Criterio de velocidad	de	Puede asignarse un valor porcentual a la variable \$APO.CVEL . Este valor indica el porcentaje de la velocidad programada (\$VEL) a partir de qué se comienza con la operación de posicionamiento aproximado, en la fase de frenado del paso individual. Para ello, se evalúa cada componente de translación, basculamiento y giro que se alcanza mediante el movimiento del valor de velocidad programado, o el que más se le aproxime.

Cuanto mayores son los valores en **\$APO.CDIS**, **\$APO.CORI** o **\$APO.CVEL**, antes se comienza con el posicionamiento aproximado.
Bajo determinadas condiciones, internamente en el sistema, puede reducirse el posicionamiento aproximado (centro del paso, criterio de simetría), pero nunca agrandarlo.

C_DIS
C_ORI
C_VEL

El posicionamiento aproximado se activa mediante la adición de una de las palabras clave **C_DIS**, **C_ORI** o **C_VEL** en la instrucción **LIN** o **LIN_REL**.

Como aclaración puede servir el ejemplo siguiente en relación con Fig. 30:



```

DEF  UEBERLIN ( )

;----- Sección de declaraciones -----
EXT  BAS (BAS_COMMAND :IN,REAL :IN )
DECL AXIS HOME

;----- Inicialización -----
BAS (#INITMOV,0 ) ;inicialización de las velocidades,
                  ;aceleraciones, $BASE, $TOOL, etc.
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}

;----- Sección principal -----
PTP  HOME ; desplazamiento COI

PTP  {POS: X 1159.08,Y -232.06,Z 716.38,A 171.85,B 67.32,C
162.65,S 2,T 10}

;posicionamiento aproximado del punto según el criterio de distancia
$APO.CDIS=20
LIN  {X 1246.93,Y -98.86,Z 715,A 125.1,B 56.75,C 111.66} C_DIS
LIN  {X 1109.41,Y -0.51,Z 715,A 95.44,B 73.45,C 70.95}

;posicionamiento aproximado de dos puntos
LIN  {X 1296.61,Y 133.41,Z 714.99,A 150.32,B 55.07,C 130.23}
C_ORI
LIN  {X 988.45,Y 238.53,Z 714.99,A 114.65,B 50.46,C 84.62} C_VEL

LIN  {X 1209.5,Y 381.09,Z 715,A -141.91,B 82.41,C -159.41}

PTP  HOME
END
    
```

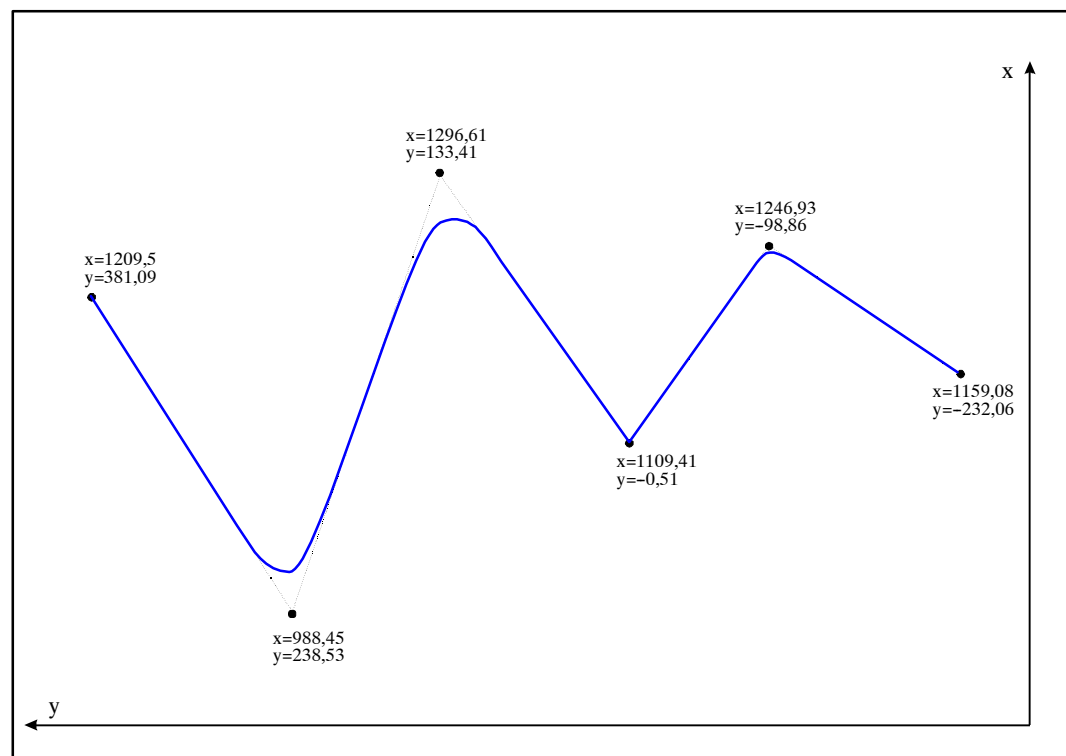


Fig. 30 Ejemplo para el posicionamiento aproximado LIN-LIN



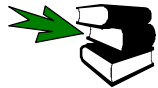
La posición en la que desemboca el contorno de posicionamiento aproximado en el paso de movimiento LIN posterior, es calculada automáticamente por la unidad de control. En el caso de que \$ACC y \$VEL sean idénticos para ambos pasos individuales y las longitudes de los pasos sean suficientes grandes, la parábola de aproximación es simétrica a la bisectriz entre los dos pasos individuales. En pasos individuales cortos se limita el comienzo de la zona de aproximación a la mitad de la longitud del paso. La velocidad se reduce para ello de forma tal, que pueda mantenerse siempre un posicionamiento exacto en el punto siguiente.

Las transiciones entre los pasos individuales y el contorno de posicionamiento aproximado son permanentes y son tangenciales. Esto garantiza una transición “suave” que protege la mecánica, ya que los componentes de velocidad son siempre constantes.

El contorno generado por la unidad de control en la zona de aproximación depende de las modificaciones del override, que se admiten para cada instante del movimiento.

3.5.3 Aproximación CIRC-CIRC y aproximación CIRC-LIN

El posicionamiento aproximado entre los pasos CIRC y otros pasos CP (LIN o CIRC), es casi idéntico al posicionamiento aproximado entre dos pasos LIN. Tanto el movimiento de orientación como el translatorio deben pasar sin saltos de velocidad de uno a otro contorno de paso individual. El comienzo del posicionamiento aproximado se define nuevamente a través de las variables \$APO.CDIS, \$APO.CORI o \$APO.CVEL, cuya evaluación se ejecuta de forma totalmente idéntica a los pasos LIN. El ajuste del criterio de posicionamiento aproximado deseado se efectúa siempre con la ayuda de las palabras clave C_DIS, C_ORI o C_VEL (véase 3.5.2).



Informaciones adicionales se encuentran en el capítulo **[Programación de movimiento]**, apartado **[Aproximación LIN-LIN]**.

El posicionamiento aproximado CIRC-CIRC se explica aquí nuevamente a través de un ejemplo y de la trayectoria de movimiento representada (véase Fig. 31):



```

DEF  UEBERCIR ( );----- Sección de declaraciones -----
EXT  BAS (BAS_COMMAND :IN,REAL :IN )
DECL AXIS HOME;----- Inicialización -----
BAS (#INITMOV,0 ) ;inicialización de las velocidades,
                  ;aceleraciones, $BASE, $TOOL, etc.
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0};----- Sec-
ción principal -----
PTP  HOME ; desplazamiento COI
PTP  {POS: X 980,Y -238,Z 718,A 133,B 66,C 146,S 6,T 50}
; Control de orientación variable referida al espacio
; posicionamiento aproximado según el criterio de distancia
$APO.CDIS=20
CIRC {X 925,Y -285,Z 718},{X 867,Y -192,Z 718,A 155,B 75,C 160}
C_DIS

; Control de orientación constante referida al espacio
; punto final especificado mediante indicación del ángulo
; no es posible el posicionamiento aproximado debido a la
parada del procesamiento en avance mediante $OUT
$ORI_TYPE=#CONST
CIRC {X 982,Y -221,Z 718,A 50,B 60,C 0},{X 1061,Y -118,Z 718,A
-162,B 60,C 177}, CA 150 C_ORI
$OUT[3]=TRUE

; Control de orientación variable referida a la trayectoria
; posicionamiento aproximado según el criterio de orientación
$ORI_TYPE=#VAR
$CIRC_TYPE=#PATH
CIRC {X 963.08,Y -85.39,Z 718},{X 892.05,Y 67.25,Z 718.01,A
97.34,B 57.07,C 151.11} C_ORI

```



```

; movimientos circulares relativos

; posicionamiento aproximado según el criterio de velocidad
$APO.CVEL=50
CIRC_REL {X -50,Y 50},{X 0,Y 100} C_VEL

; posicionamiento aproximado según el criterio de distancia
$APO.CDIS=40
CIRC_REL {X -50,Y 50},{X 0,Y 100} C_DIS

CIRC_REL {X -50,Y 50},{X 0,Y 100}

PTP HOME
END

```

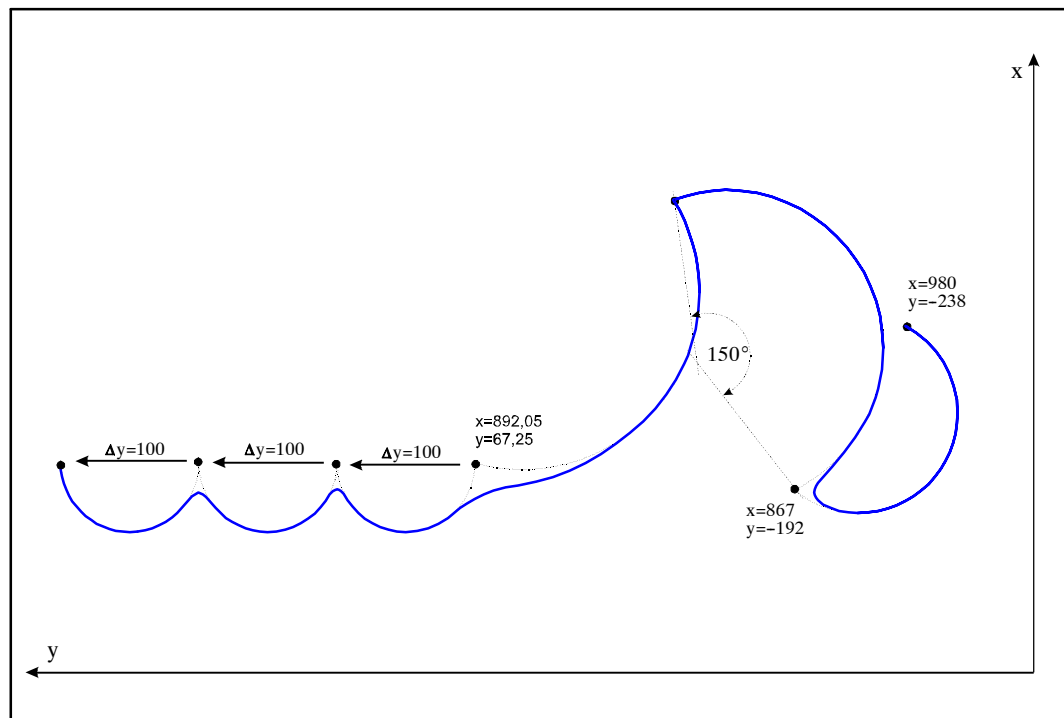
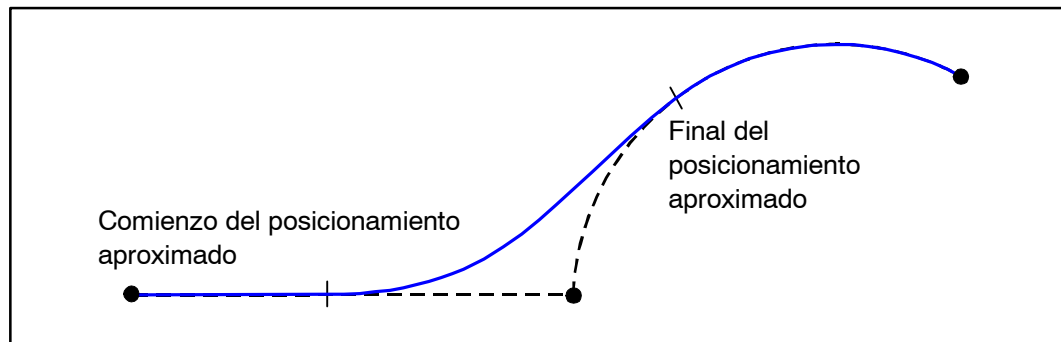


Fig. 31 Ejemplo para el posicionamiento aproximado CIRC-CIRC



Debido al requisito de transiciones tangenciales, durante el posicionamiento aproximado con pasos CIRC, no puede calcularse, en general, ningún contorno de posicionamiento aproximado simétrico. Por ello, la trayectoria de posicionamiento aproximado consta de dos segmentos parabólicos que pasan tangencialmente uno dentro de otro y que desembocan simultáneamente, de forma tangencial, en los pasos individuales (véase Fig. 32).

Aproximación
LIN-CIRC**Fig. 32** Contorno de posicionamiento aproximado en los pasos LIN-CIRC

Durante el posicionamiento aproximado CIRC, se interpola, básicamente, con referencia al espacio. La orientación de partida es siempre la orientación que se alcanzó en el punto de posicionamiento aproximado. Si se ejecutan dos pasos de posicionamiento aproximado con orientación referida a la trayectoria, a pesar de ello, la reorientación en la zona de aproximación se comporta con referencia al espacio.

3.5.4 Aproximación PTP – trayectoria

Existe la posibilidad, de efectuar posicionamientos aproximados entre pasos de movimiento PTP específicos del eje y de trayectoria cartesiana. El movimiento PTP ofrece las siguientes posibilidades:

- Básicamente es más rápido que en el mismo caso cartesiano, especialmente en la proximidad de posiciones singulares.
- En contraposición a la interpolación cartesiana, permite un cambio de configuración, por ejemplo, una transición del área base al área por sobre la cabeza o un basculamiento con la posición extendida de la muñeca.



La trayectoria exacta de un movimiento PTP no puede ser determinada con antelación, dado que el robot efectúa para él, el recorrido más rápido. Aquí la trayectoria es influenciada un poco con algunos factores (por ej. la velocidad del movimiento).

Sin embargo, las ventajas de la interpolación específica del eje sólo pueden aprovecharse totalmente cuando se tiene la posibilidad de una transición continua entre pasos específicos del eje y cartesianos, ya que durante una parada exacta, se pierde en gran parte el tiempo ganado.

La programación de la aproximación PTP – trayectoria, se efectúa de forma totalmente análoga al procedimiento presentado hasta ahora. La zona de aproximación se especifica del modo siguiente:

Aproximación PTP → trayectoria

Aproximación
PTP→
trayectoria

El comienzo de la aproximación se determina mediante el criterio PTP \$APO.CPTP de la forma habitual (véase 3.5.1). Para el paso de movimiento de trayectoria siguiente, puede indicarse explícitamente un criterio de aproximación (C_DIS, C_ORI, C_VEL), que especifica la entrada en el paso CP.

Esto se produce mediante la secuencia de instrucción:

```
PTP PUNTO1 C_PTP C_DIS
LIN PUNTO2
```

Si en el paso PTP falta una indicación para el criterio de aproximación deseado en el paso CP, C_DIS se utiliza como valor por defecto para la determinación de la entrada en el paso CP.



El contorno de aproximación de una aproximación PTP – trayectoria, o bien, trayectoria – PTP, es un movimiento PTP. Por ello no es posible una determinación exacta de la trayectoria de aproximación.

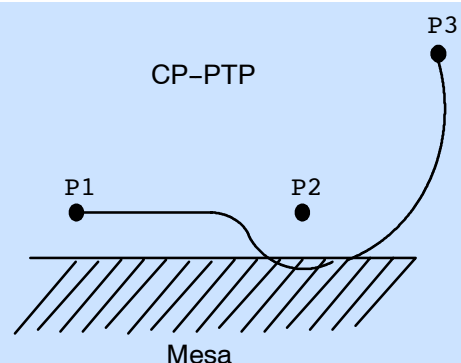
Ej.:

Si se necesita una trayectoria exacta, por ej. un movimiento de trayectoria paralelo a la mesa, entonces debe tenerse cuidado en el abandono de la trayectoria con la combinación CP-PTP en el posicionamiento aproximado. Lo mismo vale para un movimiento PTP-CP para acceder a esa trayectoria.

Programa:

```
...
LIN P1
LIN P2 C_DIS
PTP P3
...
```

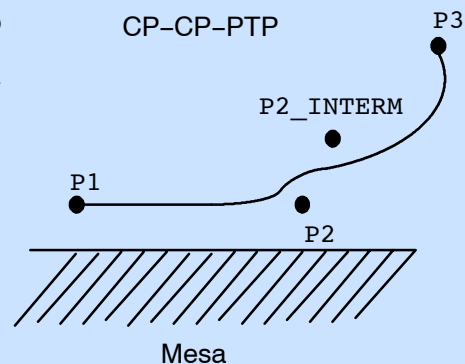
Aquí aparece el problema que el movimiento de posicionamiento aproximado no puede predecirse. Es posible que pueda existir una colisión con la mesa.



Para poder enfrentar el problema sucitado más arriba, y para evitar una parada exacta, debe insertarse otro movimiento de trayectoria (CP) (P2_INTERM).

Programa:

```
...
LIN P1
LIN P2 C_DIS
LIN P2_INTERM C_DIS
PTP P3
...
```



Aproximación trayectoria → PTP

Aproximación
trayectoria →
PTP

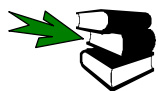
Para el paso CP se utiliza el criterio de aproximación programado, para el paso PTP se recurre a \$APO.CPTP.

Por lo tanto, una secuencia de instrucción podría tener el aspecto siguiente:

```
CIRC AUX PUNTO1 C_VEL
PTP PUNTO2
```



Sin embargo, la aproximación CP-PTP solamente puede garantizarse cuando ninguno de los ejes del robot en el paso de trayectoria gira más de 180° y el estado S no cambia, ya que estas modificaciones de posición no son previsibles durante la planificación del contorno de aproximación. Si se produce un cambio de configuración de este tipo en el paso de trayectoria antes de la aproximación (modificación de S o T), el paso de trayectoria se recorre hasta el final como un paso individual para el punto final programado, y se emite el mensaje de fallo confirmable "Progr. aproximación CP/PTP no ejecutable". El usuario debe descomponer entonces el paso CP en varios pasos individuales, de forma que el paso individual antes de la aproximación CP-PTP sea suficientemente corto para poder excluir un cambio de S o de T en todos los ejes del robot.



Informaciones adicionales se encuentran en el capítulo **[Programación de movimiento]**, apartado **[Instrucciones de movimiento]**.

En el ejemplo siguiente se programó una aproximación PTP-LIN, una aproximación LIN-CIRC y una aproximación CIRC-PTP (véase Fig. 33):



```

DEF  UEBERB_P ( )

;----- Sección de declaraciones -----
EXT  BAS (BAS_COMMAND :IN,REAL :IN )
DECL AXIS HOME

;----- Inicialización -----
BAS (#INITMOV,0 ) ;inicialización de las velocidades,
;aceleraciones, $BASE, $TOOL, etc.
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}

;----- Sección principal -----
PTP  HOME ; desplazamiento COI
PTP  {POS: X 1281.55,Y -250.02,Z 716,A 79.11,B 68.13,C 79.73,S
6,T 50}

PTP  {POS: X 1209.74,Y -153.44,Z 716,A 79.11,B 68.13,C 79.73,S
6,T 50} C_PTP C_ORI
LIN  {X 1037.81,Y -117.83,Z 716,A 79.11,B 68.13,C 79.73}

$APO.CDIS=25
LIN  {X 1183.15,Y -52.64,Z 716,A 79.11,B 68.13,C 79.73} C_DIS
CIRC {POS: X 1134,Y 53.63,Z 716},{X 1019.21,Y 124.02,Z 716,A
79.11,B 68.12,C 79.73}

CIRC {POS: X 1087.47,Y 218.67,Z 716},{X 1108.78,Y 267.16,Z 716,A
79.11,B 68.12,C 79.73} C_ORI
PTP  {POS: X 1019.31,Y 306.71,Z 716,A 80.8,B 68,C 81.74,S 6,T
59}

PTP  HOME
END

```

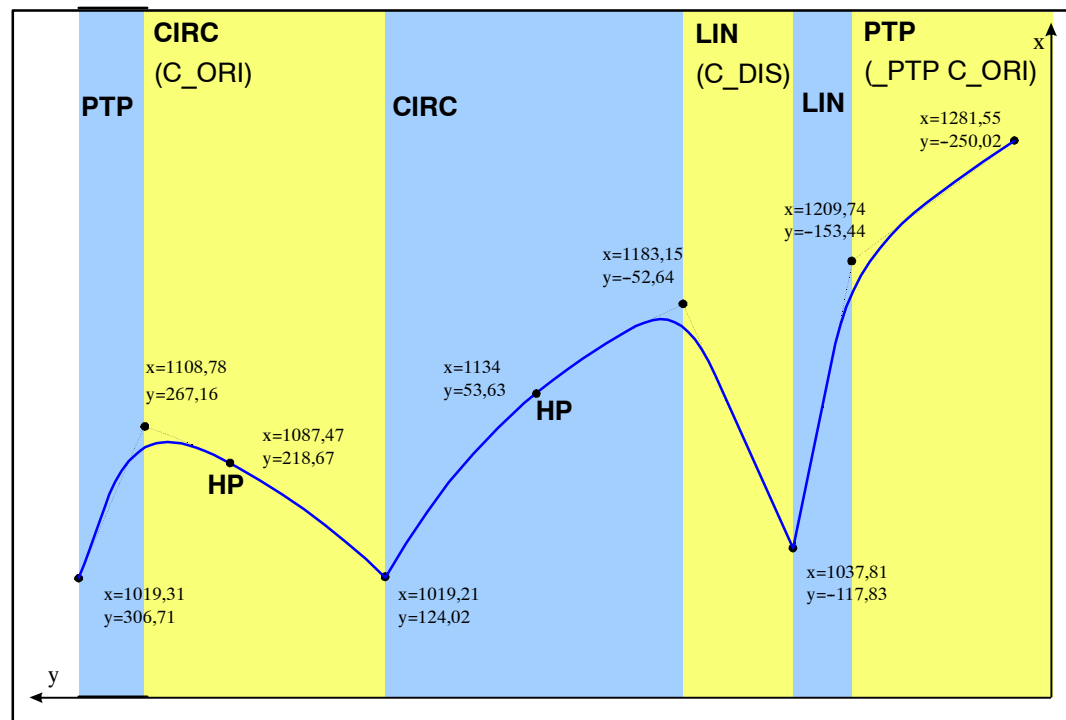


Fig. 33 Aproximación PTP-trayectoria y aproximación trayectoria-trayectoria

3.5.5 Cambio de herramienta con posicionamiento aproximado

Esta funcionalidad se encuentra disponible para todas las combinaciones de pasos individuales de PTP, LIN y CIRC.

Es posible cambiar una herramienta en forma virtual también durante el posicionamiento aproximado. Es decir, una herramienta es aprendida dos veces de forma diferente. Por ej. en una pistola de aplicación la distancia a la pieza es en "TOOL1" 5 cm y en "TOOL2" 10 cm.



Ejemplo

En este ejemplo se desea que al principio del movimiento de posicionamiento aproximado de P1 a P2, la herramienta TOOL_DATA[1] sea reemplazada por TOOL_DATA[2].

```
$TOOL=TOOL_DATA[ 1 ]
```

```
PTP P0_TOOL1;
```

Punto es programado con la herramienta TOOL_DATA[1]

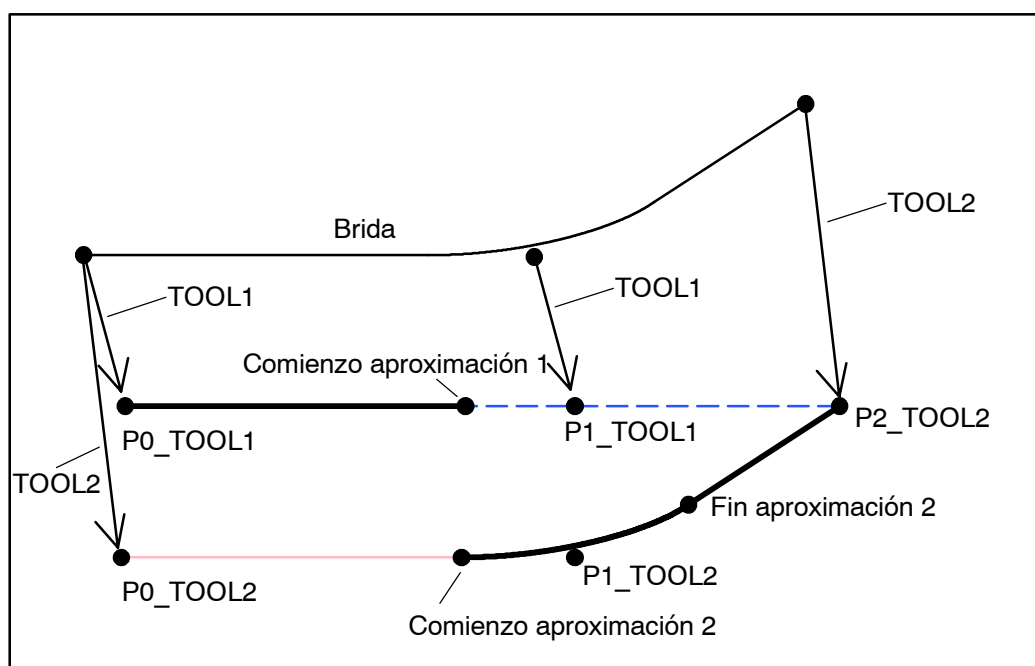
```
LIN P1_TOOL1 C_DIS;
```

Punto es programado con la herramienta TOOL_DATA[1]

```
$TOOL=TOOL_DATA[ 2 ]
```

```
LIN P2_TOOL2;
```

Punto es programado con la herramienta TOOL_DATA[2]



En este ejemplo, el cambio de herramienta se produce durante el posicionamiento aproximado. Es decir, la posición cartesiana salta al entrar al paso intermedio, de Comienzo aproximación1 a Comienzo aproximación2; los ángulos de los ejes así como también la posición cartesiana y la velocidad de la brida son guiadas de forma continua durante todo el movimiento. El salto en la posición cartesiana del TCP se consume recién durante el movimiento de Fin aproximación2 hacia P2_TOOL2, de modo tal que la trayectoria no transcurre sobre la unión lineal espacial de P1_TOOL1 hacia P2_TOOL2, sino sobre una parte de la trayectoria programada con parada exacta para TOOL2.

3.6 Programación por aprendizaje de los puntos

La integración del procedimiento de aprendizaje es una característica de calidad fundamental de un lenguaje de programación de robot.

!-símbolo

Para ello, en KRL solamente tiene que programar un símbolo ! como posicionador para las coordenadas a enseñar posteriormente:

```
PTP !
LIN ! C_DIS
CIRC ! ,CA 135.0
```

En la unidad de control pueden aceptarse entonces las coordenadas correspondientes, y después de pulsar la tecla del softkey “Modificar”, y entonces “Touch Up” memorizarlas en el programa en el lugar correspondiente. Las coordenadas actuales se escriben directamente en la estructura seleccionada, en código SRC, por ejemplo:

```
PTP {POS:X 145.25,Y 42.46,Z 200.5,A -35.56,B 0.0,C 176.87,S 2,T 2}
LIN {X -23.55,Y 0.0,Z 713.56,A 0.0,B 34.55,C -90.0} C_DIS
CIRC {X -56.5,Y -34.5,Z 45.56,A 35.3,B 0.0,C 90.0},{X 3.5,Y 20.30,
      Z 45.56,A 0.0,B 0.0,C 0.0}, CA 135.0
```



Al programar por aprendizaje las coordenadas cartesianas, el sistema de coordenadas base válido actualmente en el sistema (\$BASE), es tomado como sistema de referencia. Por ello, debe asegurarse de que durante la programación por aprendizaje esté siempre declarado el sistema de coordenadas base necesario para el movimiento posterior.



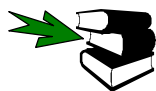
La KR C... permite otra variante de la programación por aprendizaje: programe una instrucción de movimiento con una variable que NO declare en la sección de declaraciones, por ejemplo:

```
PTP PUNTO_ARR
```

Después de activar los softkeys “Modificar” y “Var” se le requerirá que seleccione la estructura deseada. Realizado esto, en la **lista de datos** correspondiente se declara automáticamente una variable PUNTO_ARR y se ocupa con las coordenadas reales actuales respecto a \$BASE actual, por ejemplo:

```
DECL FRAME PUNTO_ARR={X 15.2,Y 2.46,Z 20.5,A -35.5,B 9.0,C 16.87}
```

Si no se ha declarado la lista de datos, aparece el correspondiente mensaje de fallo.



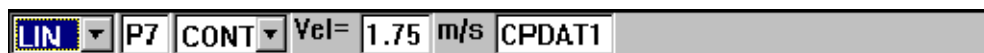
Informaciones adicionales acerca del tema “Listas de datos” se encuentran en el apartado **[Listas de datos]**.



Si ha creado una instrucción de movimiento a través de los formularios inline puede utilizar también, posteriormente, los puntos programados por aprendizaje a través del formulario, en una instrucción de movimiento KRL:

Los puntos se almacenan en la lista de datos correspondiente con los nombres indicados en el formulario y una X delante (por ello, para los nombres de puntos en los formularios inline se admiten como máximo 11 caracteres en lugar de 12).

Puede acceder al punto P7 en el formulario inline



posteriormente, como XP7 en una instrucción KRL:

```
LIN XP7
```

También aquí debe tenerse en cuenta que en ambos casos tiene que utilizarse el mismo sistema de coordenadas base para que se produzca el desplazamiento hasta el mismo punto.

3.7 Parámetros de movimiento

Esta función permite la modificación del entorno tubular de control para el control contra colisiones. Con ello puede determinarse la sensibilidad del control de colisiones.



Después de la llamada de esta instrucción se abre el siguiente formulario inline:

TORQMON. **SetDefaults** ▾

Los valores declarados en los datos de máquina ("C:\Programme\KRC\MaDa\Steu\Scu-custom.dat") se utilizan como valores por defecto. De forma alternativa, el usuario puede definir para los distintos movimientos también distintos grados de sensibilidad.

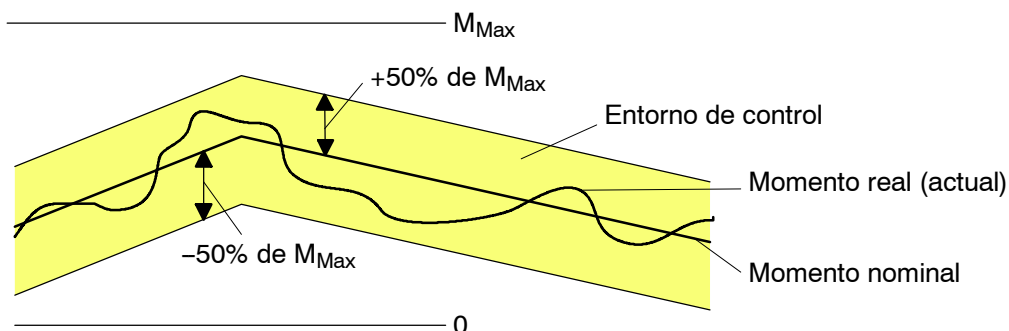
Alternativamente, el usuario puede determinar individualmente cuanto se ha de permitir la desviación entre los valores reales y nominales de los momentos. Al sobrepasar estos valores, el robot se detiene.

TORQMON. **SetLimits** ▾ Axis 1: %, Axis 2: %, Axis 3: %, Axis 4: %, Axis 5: %, Axis 6: %

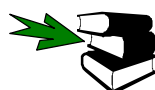


Ejemplo

En el formulario inline se han declarado para cada eje A1...A6, 50%. Si el momento actual tiene una diferencia de más de $\pm 50\%$ del máximo del momento nominal calculado, el robot se detiene emitiéndose el correspondiente mensaje en la ventana de mensajes.



Este tipo de control no representa ninguna garantía contra daños de útiles en una colisión, pero reduce los mismos.



Más detalles sobre el control de colisiones se encuentran en el **Manual de programación** en la documentación **[Configuración]**, capítulo **[Configuración del sistema, experto]**.

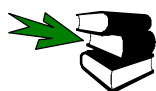
4 Asistente KRL

Los paquetes de tecnología de KUKA contienen las funciones más importantes para las aplicaciones más comunes con el robot. Funciones especiales que no estén contenidas en los paquetes tecnológicos KUKA, pueden ser realizados directamente programando el sistema del robot en “KRL” (“KUKA Robot Language”).

Para posibilitar la programación de funciones especiales en forma efectiva a aquel usuario que no utilice este idioma de programación en forma muy frecuente, se ha integrado el “Asistente de KRL”.

El “Asistente de KRL” ofrece al usuario una programación soportada por sintaxis. Una vez seleccionada la instrucción de KRL deseada, se ofrecen máscaras con indicaciones relativas a la instrucción. El contenido de estas máscaras debe ser aceptado o modificado. Todos los contenidos pueden ser modificados posteriormente siempre de nuevo.

Manejo



Para la programación de un comando de movimiento deberá seleccionar un programa o cargarlo en el editor. Para más detalle acerca de la creación y modificación de programas, puede consultarse el capítulo **[Generalidades sobre los programas KRL]**, apartado **[Crear programas y editarlos]**.



```

1  →INI
2  PTP HOME Vel= 100 % DEFAULT
3  |
4  PTP HOME Vel= 100 % DEFAULT
5  END

```

Por favor, tenga en cuenta la posición del cursor de edición. La línea de programa creada por Ud. se insertará como nueva línea debajo del cursor.

Instrucc.

Abra el menú a través de la tecla de menú “Instrucc.” el menú. Seleccione “Asistente KRL”. Aparecerá la figura siguiente:



Aquí puede seleccionar ahora los comandos de movimiento ofertados.

4.1 Indicaciones de posición

La reserva de lugar “!”



El signo “!” indica una reserva de lugar. Con ayuda de la misma, puede generarse un programa de movimiento sin conocer exactamente la posición de los puntos, que más adelante determinarán la trayectoria del movimiento del robot.

En una ejecución posterior, el programa se ha de detener en esa posición, y Ud. puede registrar el punto del siguiente modo:

Confirmar

Si durante el procesamiento ulterior del programa se muestra en la ventana de mensajes el mensaje de “Instrucción no permitida”, borre este mensaje con la tecla del softkey “Confirmar”.

Ti...	No	Abs.	Mensaje
STOP 15:42	1425		Instrucción no permitida

Desplace el sistema del robot a la posición deseada.

Touch Up

Pulse la tecla del softkey “Touch Up”. Preste atención a las indicaciones de la ventana de mensajes.

La ventana de mensajes le indica también si el punto fue aceptado y registrado.

Ti...	No	Abs.	Mensaje
15:44	0		TPEXPERT Posición actual "(POS: X1620, Y0,Z1910, A0, B90, C0, S2, T2)" se cargó

Confirmar

Ud. puede borrar este aviso de la ventana de mensajes, pulsando la tecla del softkey “Confirmar”.

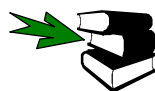
Definición de posiciones por medio de variables

KUKA01

En lugar de la reserva de lugar, puede Ud. indicar el nombre válido de una variable. Una lista de las palabras claves reservadas para KRL, las cuales no están permitidas utilizar, se encuentran en el capítulo [KRL Reference Guide].



Ud. debería desplazar el sistema del robot ya antes de la programación de esta función, a la posición deseada.



Informaciones acerca del movimiento manual del robot se encuentran en el **Manual de operación**, en la documentación **Instrucciones**, capítulo [Desplazamiento manual del robot].

VAR

Si el nombre que Ud. ha definido todavía no es conocido por el sistema, aparece, en la barra de softkeys, el softkey “VAR”. Con éste, el sistema le requiere asignarle al nombre un formato de datos.

Una vez pulsada la tecla, se modifica la asignación de la barra de softkeys:

<<	E6POS	POS	FRAME	E6AXIS	AXIS	!
----	-------	-----	-------	--------	------	---



La barra de softkeys sólo está disponible si junto al fichero “.SRC” también existe una lista de datos “.DAT”.

Después de haber activado uno de los softkeys “E6POS”, “POS”, “FRAME”, “E6AXIS” o “AXIS”, la posición actual del sistema del robot es registrada en el formato de datos seleccionado. Esto también es confirmado en la ventana de mensajes.

	Ti...	No	Abs.	Mensaje
	16:15 0	TPBASIS		Las coordenadas actuales se cargaron en el punto KUKA01

Confirmar

Ud. puede borrar este aviso, pulsando la tecla del softkey “Confirmar”.

Memorización de la posición por medio de “Touch Up”



Antes que Ud. pueda memorizar o registrar una posición por medio de “Touch Up”, debe indicarse a la unidad de control, los datos de la posición del sistema válido de coordenadas del robot, así como los datos de las herramientas y/o piezas válidas.

Para ello, debe ejecutarse la secuencia INI en el encabezamiento del programa.



Se aconseja ya antes de la programación de la función, llevar el sistema del robot a la posición deseada.

Touch Up

Si Ud. pulsa ahora la tecla del softkey “Touch Up”, se memoriza la posición actual del sistema del robot.

PTP[{POS: X 1620, Y 0, Z 1910, A 0, B 90, C 0, S 2, T 2}]

En la ventana de mensajes se confirma que el registro de la posición es válido.

	Ti...	No	Abs.	Mensaje
	00:01 0	TPEXPERT		Posición actual "[POS: X1620, Y0,Z1910, A0, B90, C0, S2, T2]" se cargó

Confirmar

Ud. puede borrar este aviso, pulsando la tecla del softkey “Confirmar”.

Indicación manual de la posición

Adicionalmente a la posibilidad de registrar posiciones a las cuales se ha desplazado el robot, Ud. puede indicar a la unidad de control puntos en el espacio en forma manual.

{ ? }

Para ello, teniendo en pantalla el formulario inline, pulse la tecla del softkey “{ ? }”. Se modifica la asignación de la barra de softkeys:

<<	E6POS	POS	FRAME	E6AXIS	AXIS	!
----	-------	-----	-------	--------	------	---

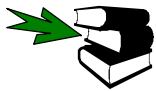
Una vez seleccionado el formato de datos pulsando la tecla del softkey correspondiente, en el formulario inline se registra la posición actual.

```
PTP[{A1 0, A2 -90, A3 90, A4 0, A5 0, A6 0}]
```

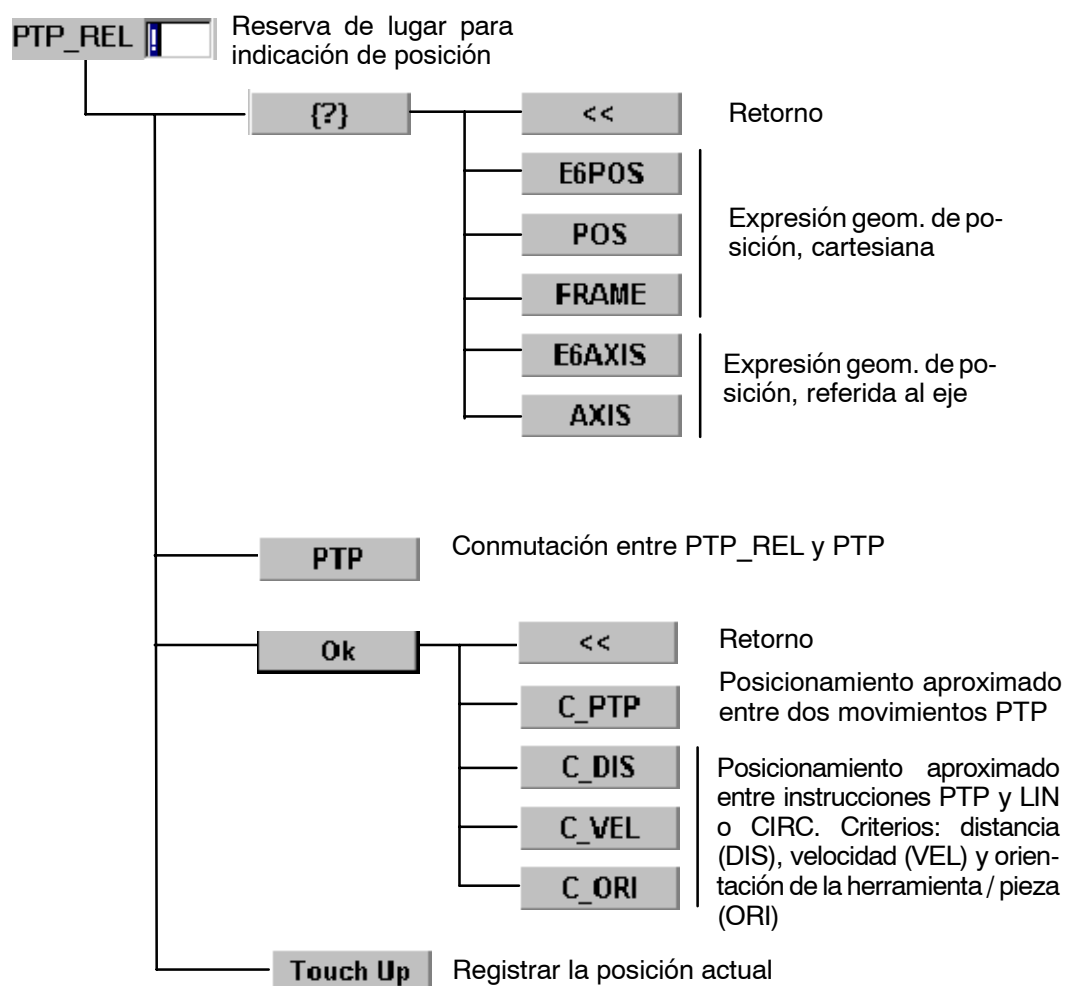
Con ayuda de las funciones de edición puede Ud. modificar las posiciones a necesidad.

El operador geométrico “:”

Con el operador geométrico “:” pueden combinarse indicaciones de posición del tipo POS y FRAME. Esto es necesario siempre cuando, por ej. debe desplazarse el origen de un sistema de coordenadas en un valor de corrección.

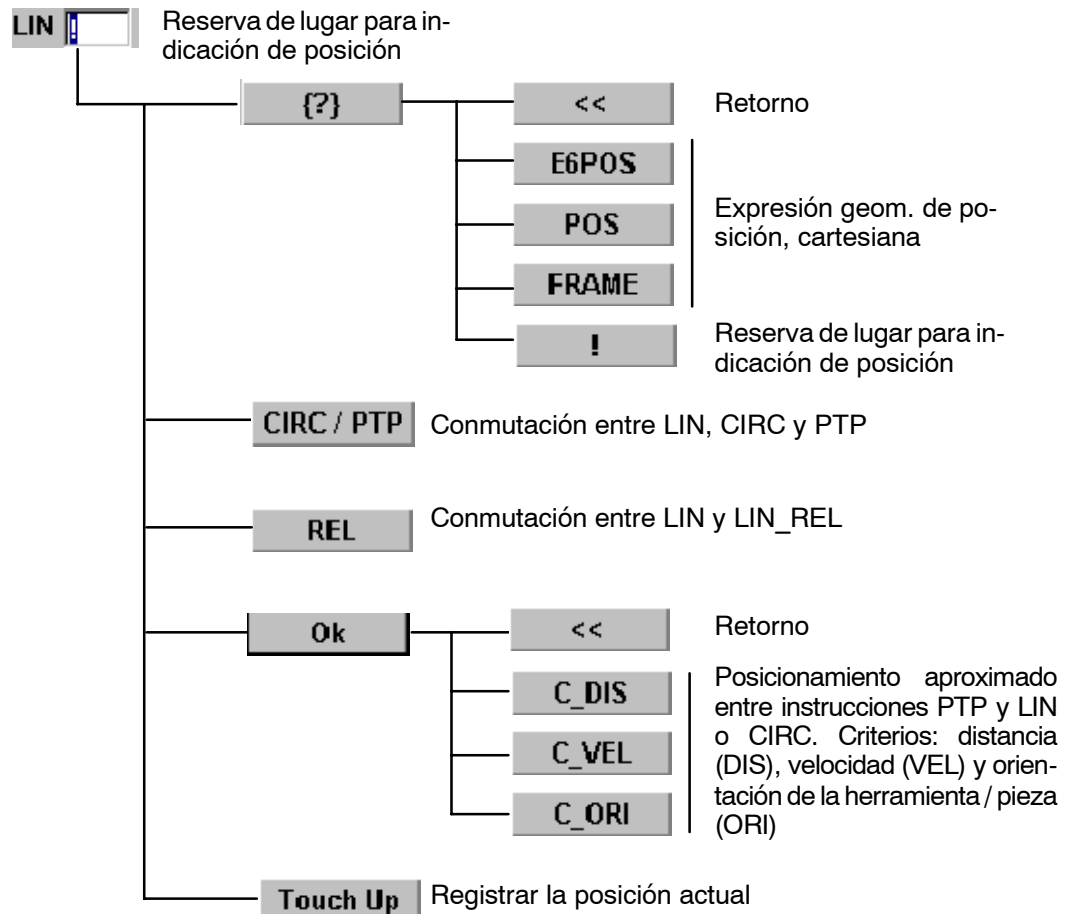


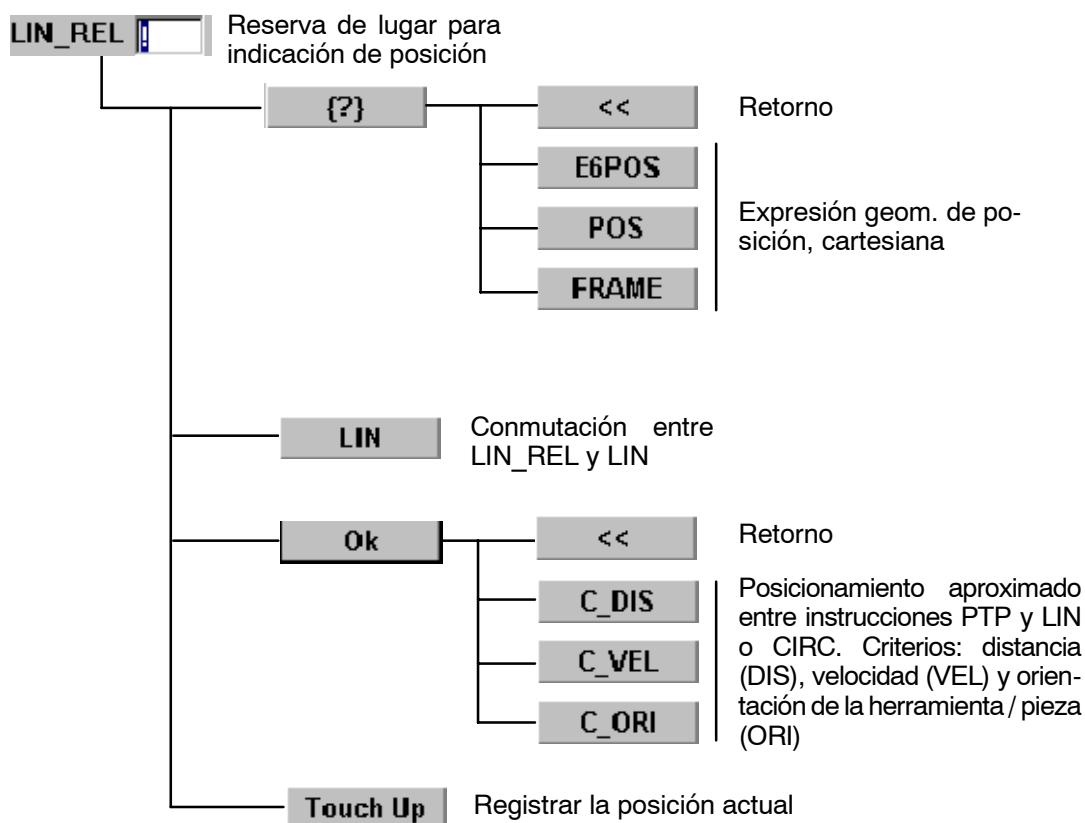
Informaciones adicionales a este tema se encuentran en el capítulo **[Variables y declaraciones]**, apartado **[Manipulación de datos]** bajo “Operador geométrico”.



4.3 Movimiento lineal [LIN]

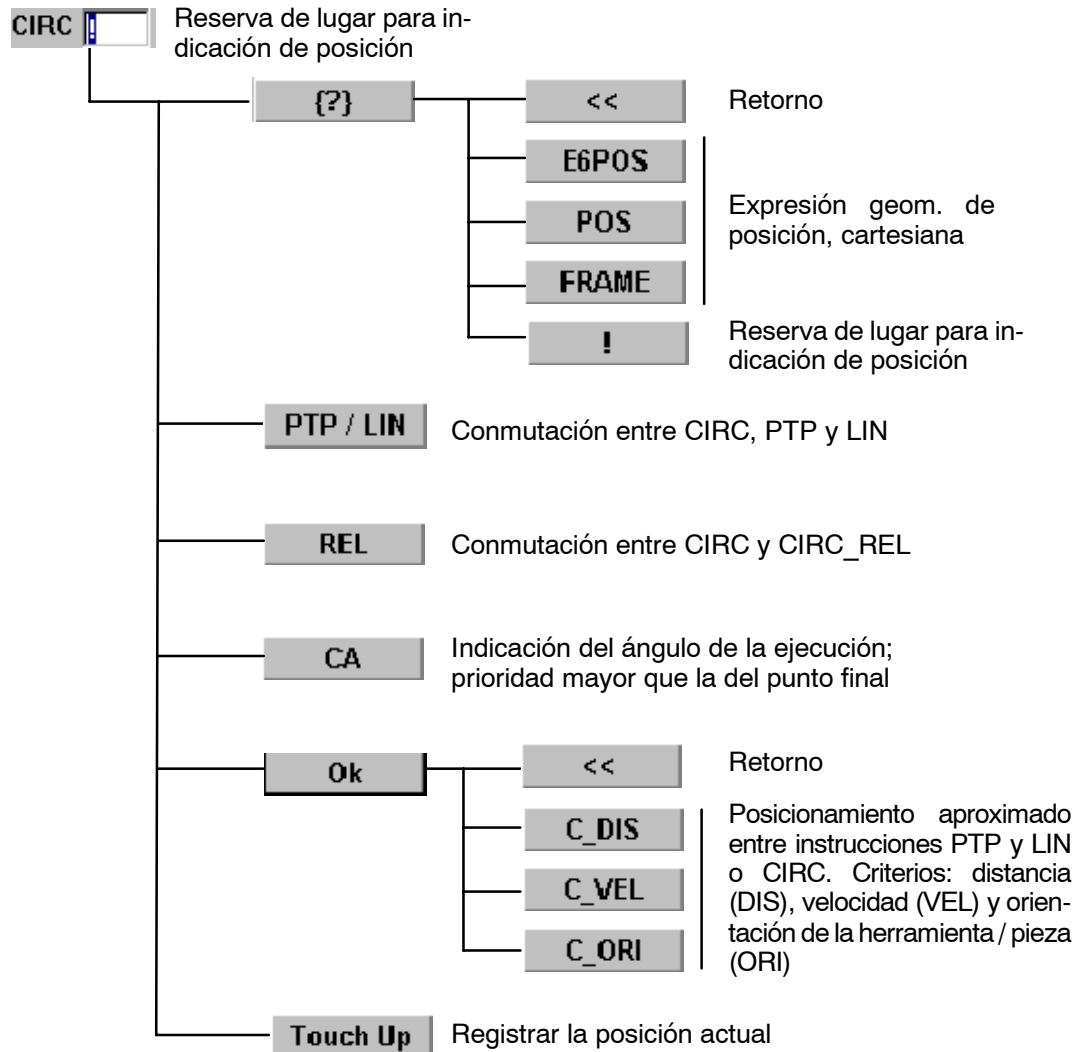
El posicionamiento del sistema del robot se ejecuta aquí en el recorrido más corto entre dos puntos del espacio de trabajo, es decir, una recta. Los ejes del sistema del robot se sincronizan de tal manera, que la velocidad de la trayectoria queda siempre constante sobre esa recta.

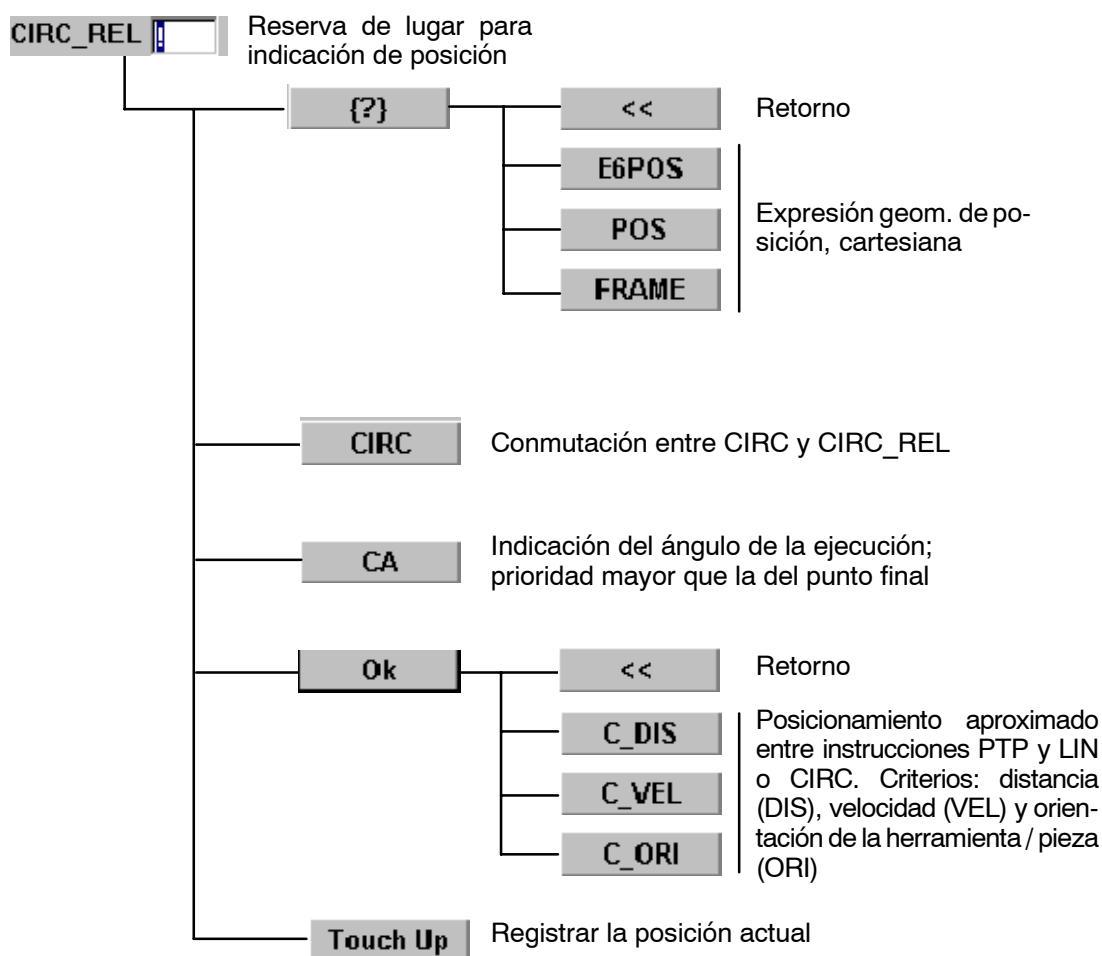




4.4 Movimientos circulares [CIRC]

El posicionamiento del sistema del robot se realiza aquí a lo largo de una trayectoria circular en el espacio de trabajo. Esta trayectoria está definida por un punto inicial, un punto intermedio y un punto final. Los ejes del sistema del robot se sincronizan aquí de modo tal, que la velocidad sobre la trayectoria circular es siempre constante.





5 Control de ejecución del programa

5.1 Ramificaciones de programa

5.1.1 Instrucciones de salto

La forma más simple de la ramificación de programa la constituye la denominada instrucción de salto incondicional. Esta puede, sin reflejar ninguna condición determinada, ser ejecutada en todo caso. A través de la instrucción

GOTO

```
GOTO MARCA
```

el puntero del programa salta a la posición MARCA. Pero la posición deberá estar definida imprescindiblemente con

MARCA:

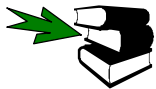
en alguna parte del programa. La instrucción de salto propiamente dicha no permite deducciones acerca de la estructura de programa generada. Por consiguiente, se deberá elegir un nombre de marca de salto de modo que la acción de salto ligada a ella, sea más comprensible. Por ej. es una diferencia notable si se escribe

```
GOTO MARCA_1
```

o

```
GOTO STOP_PEGAM
```

La instrucción GOTO conduce rápidamente a programas no estructurados sin una buena visión global, dado que además cualquier instrucción GOTO puede ser sustituida por cualquier otra instrucción de bucle, deberá, en lo posible, evitarse la utilización de GOTO.



Un ejemplo de “GOTO” se encuentra en este capítulo en el apartado **[Bucles]** bajo “Bucles no rechazantes”.

5.1.2 Ramificación condicionada

IF La instrucción IF permite la formulación de instrucciones condicionadas ofreciendo dos alternativas para la selección. En la forma más común la instrucción es la siguiente

```
IF Condición de ejecución THEN
    Instrucciones
ELSE
    Instrucciones
ENDIF
```

La condición de ejecución es una expresión booleana. Si la condición de ejecución ha sido cumplidas, se ejecutará el bloque THEN. En el otro caso, puede ejecutarse el bloque ELSE o se prescinde del mismo. Prescindir significa el abandono inmediato de la ramificación.

Se permiten la cantidad de instrucciones que se desee. Estas pueden constar a su vez de otras instrucciones IF. Esto significa que se pueden encajar los bloques IF. Sin embargo, cada instrucción IF debe cerrarse con una instrucción ENDIF propia.

En la secuencia de programa que se presenta a continuación, se realiza un desplazamiento a la posición HOME, si la entrada 10 está puesta en FALSE. Si está activada la entrada 10 y si la variable A es mayor que la variable B, se activa la salida 1 y se produce un desplazamiento al punto 1. Independientemente de A y de B, en caso de que se activa la entrada 10, se aumenta la variable A en 1 y se produce el desplazamiento a la posición HOME:

```
...
INT A,B
...
IF $IN[10]==FALSE THEN
    PTP HOME
ELSE
    IF A>B THEN
        $OUT[1]=TRUE
        LIN PUNTO1
    ENDIF
    A=A+1
    PTP HOME
ENDIF
...
```

5.1.3 Distribuidor

SWITCH

Identificación de bloque

Si existen más de 2 alternativas, se puede programar bien con una estructura IF encajada o bien – más confortable – utilizar en la programación el distribuidor SWITCH.

La instrucción SWITCH es una instrucción de selección para diferentes ramificaciones del programa. Un criterio de selección es ocupado antes de la instrucción SWITCH con un valor determinado. Si el valor concuerda con una identificación de bloque, se procesa la ramificación de programa correspondiente, saltando el programa, sin tener en consideración la identificación de bloque subsiguiente, hasta la instrucción ENDSWITCH. Si no concuerda una identificación de bloque con el criterio de selección, se procesará, en caso de que exista, un bloque DEFAULT (por defecto). En caso contrario se continuaría con la instrucción siguiente a partir de la instrucción ENDSWITCH.

A una ramificación de programa es permisible asignar varios identificadores de bloque. Sin embargo, a la inversa, no es recomendable utilizar un identificador de bloque repetidas veces ya que sólo es considerada la primera ramificación con la identificación correspondiente.

Los tipos de datos permitidos del criterio de selección son INT, CHAR y ENUM. El tipo de dato del criterio de selección deberá concordar con la identificación de bloque.

La instrucción de DEFAULT (por defecto) puede faltar, pero sólo puede aparecer una sola vez dentro de una instrucción SWITCH.

Con una instrucción SWITCH puede llamar p. ej. diferentes subprogramas en dependencia de un número de programa. El número de programa podría enviarse p. ej. del PLC a las entradas digitales del KR C... (véase apartado 6.3 acerca de la instrucción SIGNAL). Así está disponible como criterio de selección en forma de un valor Integer (entero).



```

DEF MAIN()
...
SIGNAL PROG_NR $IN[1] TO $IN[4]
    ;en la variable INT PROG_NR se transmite por el PLC
    ;seguidamente el número de programa deseado
...
SWITCH PROG_NR
    CASE 1                ;en caso de PROG_NR=1
        PARTE_1()
    CASE 2                ;en caso de PROG_NR=2
        PARTE_2()
        PARTE_2A()
    CASE 3,4,5            ;en caso de PROG_NR=3, 4 o 5
        $OUT[3]=TRUE
        PARTE_345()
    DEFAULT                ;en caso de PROG_NR<>1,2,3,4 o 5
        ERROR_UP()
ENDSWITCH
...
END

```



En forma similar está estructurado el programa CELL (CELL.SRC) estándar en la unidad de control.

5.2 Bucles

La estructura base siguiente para el control del procesamiento del programa son los bucles que contienen una o varias instrucciones de procesamiento repetitivas hasta que se haya cumplido una determinada condición. Los bucles se diferencian según la forma de la condición y posición donde se produce la interrogación respecto a su continuación.

El salto desde fuera hacia dentro de un cuerpo de bucle no está permitido siendo rechazado por la unidad de control (mensaje de fallo).

5.2.1 Bucles de conteo

Los bucles de conteo son ejecutados hasta que la variable de conteo sobrepase o quede por debajo de un determinado valor, bien mediante conteo ascendente o descendente. En la KRL se tiene a disposición una instrucción FOR. Con

FOR

```
FOR Contador = Arranque TO Final STEP Ancho de paso
    Instrucciones
ENDFOR
```

De esta forma se permite, con una buena visión global, programar una determinada cantidad de secuencias.

Como valor de **Arranque** y valor **Final** del contador deberá indicar, respectivamente, una expresión del tipo Integer (entero). Las expresiones son analizadas al principio del bucle. La variable INT **Contador** (por consiguiente deberá estar declarada previamente) es impuesta con el valor de arranque, aumentando o disminuyendo según el ancho de paso tras cada secuencia del bucle.

El **ancho de paso** no debe ser ni una variable ni tampoco cero. Si se omite la indicación del ancho de paso, dispondrá por defecto el valor estándar 1. También se admiten valores negativos para el ancho de paso.

Para cada instrucción FOR debe haber una instrucción **ENDFOR**. El programa continuará procesándose, después de la última secuencia de bucle, en la primera instrucción detrás de **ENDFOR**.

El valor del contador puede utilizarse tanto dentro como fuera del bucle. Dentro del bucle, este sirve p.ej. de índice actual para la edición de los campos. Después de salir del bucle, el contador adopta el último valor tomado (es decir **Final+Ancho de paso**).

En el ejemplo que se muestra a continuación, se colocan las velocidades de ejes \$VEL_AXIS[1]...\$VEL_AXIS[6] a 100%. Seguidamente se inicializan los componentes de un campo bidimensional con los valores calculados. El resultado se representa en Tab. 20.



```

DEF FOR_PROG()
...
INT I,J
INT CAMPO[10,6]
...
FOR I=1 TO 6
  $VEL_AXIS[I] = 100 ;Todas las velocidades de ejes al 100%
ENDFOR
...
FOR I=1 TO 9 STEP 2
  FOR J=6 TO 1 STEP -1
    CAMPO[I,J] = I*2 + J*J
    CAMPO[I+1,J] = I*2 + I*J
  ENDFOR
ENDFOR
;I adopta ahora el valor 11, J el valor 0
...
END

```

Index		I =									
		1	2	3	4	5	6	7	8	9	10
J =	6	38	8	42	24	46	40	50	56	54	72
	5	27	7	31	21	35	35	39	49	43	63
	4	18	6	22	18	26	30	30	42	34	54
	3	11	5	15	15	19	25	23	35	27	45
	2	6	4	10	12	14	20	18	28	22	36
	1	3	3	7	9	11	15	15	21	19	27

Tab. 20 Resultado de cálculo para el ejemplo 5.2

5.2.2 Bucle rechazante

WHILE

El bucle WHILE interroga al principio de la repetición sobre una condición de ejecución. Se trata de un bucle rechazante, ya que no se ejecuta ni una sola vez si la condición de ejecución no es cumplida ya desde el principio. La sintaxis del bucle WHILE es:

```
WHILE Condición de ejecución
    Instrucciones
ENDWHILE
```

La condición de ejecución es una expresión lógica que puede tratarse tanto de una variable booleana, una llamada de función booleana o una combinación lógica con un resultado booleano.

El bloque de instrucciones se ejecuta cuando la condición lógica ha tomado el valor de TRUE, es decir, que la condición de ejecución se ha cumplido. Cuando la condición lógica tiene el valor FALSE, el programa continua procesándose detrás de la instrucción ENDWHILE. Por consiguiente, deberá finalizar cada instrucción WHILE con una instrucción ENDWHILE.

La utilización de WHILE se puede deducir perfectamente del ejemplo 5.3.



```
DEF WHILE_PR()
...
INT X,W
...
WHILE $IN[4] == TRUE ;Ejecución mientras que entrada 4 activada
    PTP PALETA
    $OUT[2] = TRUE
    PTP POS_2
    $OUT[2] = FALSE
    PTP HOME
ENDWHILE
...
X = 1
W = 1
WHILE W < 5; ;Ejecución mientras que W sea menor que 5
    X = X * W
    W = W + 1
ENDWHILE
;W ahora es 5
;X ahora es 1•2•3•4 = 24
...
W = 100
WHILE W < 100 ;Ejecución mientras que W sea menor que
100
    $OUT[15] = TRUE W = W + 1
ENDWHILE
... ;El bucle no se ejecuta nunca, W queda 100
END
```


5.2.3 Bucles no rechazantes

REPEAT

Lo contrario a un bucle WHILE es un bucle REPEAT. Con REPEAT se interroga al final del bucle acerca de la condición de interrupción. Por ello un bucle REPEAT realizará, en todo caso, el ciclo completo, incluso cuando se cumpla ya de entrada la condición de interrupción desde el principio del bucle.

REPEAT

Instrucciones

UNTIL Condición de interrupción

La condición de interrupción es análoga a la del bucle WHILE, una expresión lógica, una variable booleana, una llamada de función booleana o una combinación lógica con un resultado booleano:



```

DEF REPEAT_P()
...
INT W
...
REPEAT
    PTP PALETA
    $OUT[2]=TRUE
    PTP POS_2
    $OUT[2]=FALSE
    PTP HOME
UNTIL $IN[4] == TRUE    ;Ejecución hasta que se active la entrada 4
...
X = 1
W = 1
REPEAT
    X = X * W
    W = W + 1
UNTIL W == 4           ;Ejecución hasta que W sea igual a 4
                        ;W ahora es 4
                        ;X ahora es 1•2•3•4 = 24
...
W = 100
REPEAT
    $OUT[15] = TRUE
    W = W + 1
UNTIL W > 100          ;Ejecución hasta que W sea mayor de 100
                        ;mín. 1 ejecución de bucle, es decir
                        ;W ahora es 101, la salida 15 se activa
...
END

```

Con WHILE y REPEAT tiene en sus manos una herramienta muy potente para la programación estructurada, con esta podrá sustituir en la mayoría de los casos, las instrucciones GOTO. La secuencia de instrucción

```
...
X = 0
G = 0
MARCA:
X = X + G
G = G + 1
    IF G > 100 THEN
        GOTO LISTO
    ENDIF
GOTO MARCA:
LISTO:
...
```

se puede realizar por ejemplo de una forma más elegante con REPEAT:

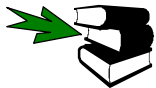
```
...
X = 0
G = 0
REPEAT
    X = X + G
    G = G + 1
UNTIL G > 100
...
```

5.2.4 Bucles sinfín

LOOP Con la instrucción **LOOP** se pueden programar bucles sinfín:

```
LOOP
    Instrucciones
ENDLOOP
```

La ejecución repetida del bloque de instrucción sólo se puede finalizar mediante la instrucción **EXIT**.



Informaciones adicionales respecto a la **instrucción Exit** se encuentran en el próximo apartado.

5.2.5 Terminación anticipada de la ejecución de bucles

EXIT Con la instrucción **EXIT** puede terminar cualquier bucle prematuramente. Llamando a **EXIT** dentro de un bloque de instrucción del bucle, se terminan los ciclos de bucles, continuando el programa detrás de la instrucción del final del bucle.

Seleccionando adecuadamente las condiciones de cancelación o de ejecución, no es necesario en la mayoría de las veces, utilizar la instrucción **EXIT** en bucles **WHILE** o **REPEAT**. Para los bucles sinfín, el **EXIT** es, sin embargo, la única posibilidad de finalizar la ejecución del bucle. Respecto a ello véase el ejemplo siguiente:



```
DEF EXIT_PRO()
PTP HOME
LOOP                                ;Inicio del bucle sinfín
    PTP POS_1
    LIN POS_2
    IF $IN[1] == TRUE THEN
        EXIT                        ;Cancelar, cuando entrada 1 activada
    ENDIF
    CIRC HELP_1,POS_3
    PTP POS_4
ENDLOOP                            ;Final del bucle sinfín
PTP HOME
END
```

5.3 Instrucciones de espera

WAIT Con la instrucción **WAIT** puede forzar la detención del programa hasta que se produzca una situación determinada. Se hace una diferenciación entre esperar que se produzca un evento y el establecimiento de temporizaciones de espera.

5.3.1 Espera a un evento

Con la instrucción

WAIT FOR Condición

puede detener el procesamiento del programa hasta que se produzca el evento especificado con **Condición**:

- Si la expresión lógica de **Condición** durante la llamada de **WAIT** ya es **TRUE**, el procesamiento del programa no será interrumpido (sin embargo se fuerza una parada del procesamiento en avance).
- Si la condición es **FALSE**, el procesamiento del programa se interrumpirá hasta que la expresión tome de nuevo el valor **TRUE**.

Ejemplos:

```
WAIT FOR $IN[14]           ;espera hasta que la entrada 14 sea TRUE
WAIT FOR BIT_1 == FALSE    ;espera hasta que la Var. BIT_1 = FALSE
```

El compilador no reconoce si la expresión no es capaz de tomar el valor **TRUE** debido a una formulación errónea. En este caso, se detiene indefinidamente el procesamiento del programa debido a que el intérprete del programa espera a una condición irrealizable.

5.3.2 Tiempos de espera

La instrucción **WAIT SEC** sirve para programar los tiempos de espera en segundos:

WAIT SEC Tiempo

Tiempo es una expresión aritmética **REAL**, con la que puede indicar la cantidad de segundos que desea que dure la interrupción del procesamiento del programa. Si el valor es negativo, no se produce la espera.

Ejemplos:

```
WAIT SEC 17.542
WAIT SEC TIEMPO*4+1
```

5.4 Detener el programa

HALT

Si desea interrumpir la ejecución del programa y detener el procesamiento, deberá programar la instrucción

HALT

Pero la última instrucción de movimiento procesada es ejecutada completamente. El procesamiento continua presionando la tecla de arranque. A continuación se ejecutará la instrucción subsiguiente después de HALT.



Caso especial: en una rutina de interrupción, se detendrá el procesamiento del programa mediante una instrucción HALT después de haber ejecutado el procesamiento en avance (véase capítulo 8, “Tratamiento de interrupciones”).

Excepción: en la programación de una instrucción BRAKE se para inmediatamente.

5.5 Confirmación de mensajes

CONFIRM

Con la instrucción

CONFIRM V_Numero

puede confirmar por programa los mensajes confirmables. Después de realizar la confirmación con éxito, el mensaje especificado con un número de gestión V_Numero ya no estará disponible.

Después de suprimir una señal de Stop se emite p. ej. siempre un mensaje confirmable. Antes de continuar el procesamiento del programa, el mensaje debe ser confirmado. El siguiente subprograma reconoce y confirma automáticamente este mensaje, en el momento que esté seleccionado el modo de servicio correcto (no servicio manual) y el estado que determinó la parada haya sido realmente solucionado (dado que un programa de robot no esta listo para funcionar si aún está presente un mensaje confirmable, deberá procesarse el subprograma en un fichero Submit):



```

DEF AUTO_QUIT()
INT M
DECL STOPMESS MLD      ;Tipo de estructura predefinida para men-
sajes de Stop
IF $STOPMESS AND $EXT THEN      ;Verificar mensaje de Stop y
comprobar modo de servicio
    M=MBX_REC($STOPMB_ID,MLD)    ;lectura del estado actual en
MLD
    IF M==0 THEN      ;Comprobación, si se puede producir la
confirmación
        IF ((MLD.GRO==2) AND (MLD.STATE==1)) THEN
            CONFIRM MLD.CONFNO      ;Confirmación de este mensaje
        ENDIF
    ENDIF
ENDIF
ENDIF
END
    
```

6 Instrucciones de entrada/salida

6.1 Generalidades

La KR C... conoce 1026 entradas y 1024 salidas. En el armario de la unidad de control estándar de KUKA, el usuario dispone, en el conector X11 (módulo MFC), de las entradas y salidas siguientes:

Entradas	1 ...16	
Salidas	1 ...16	(carga admisible máx. de 100 mA; 100% simultaneidad)
Salidas	17 ...20	(carga admisible máx. de 2 A; 100% simultaneidad)

Opcionalmente, pueden configurarse entradas/salidas adicionales como p. ej. a través de buses de campo.

Las entradas pueden ser leídas, las salidas pueden ser leídas y activadas. Estas son excitadas a través de las variables de sistema $\$IN[Nr]$ o bien. $\$OUT[Nr]$. Las salidas no utilizadas pueden usarse como marcas.

Las entradas/salidas del modulo MFC pueden conmutarse a otras zonas por medio del archivo "IOSYS.INI".

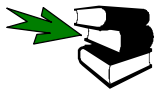


La unidad de control tipo "KR C2" se entrega, de forma estándar, sin E/S's.



Por motivos de seguridad, todas las instrucciones de entrada y de salida así como los accesos a las variables de sistema para entradas/salida, producen una parada del procesamiento en avance.

Accesos a variables del sistema para entradas/salidas no generan un stop del procesamiento en avance con una predefinición de la instrucción CONTINUE.



Informaciones adicionales se encuentran en el capítulo **[Programación de movimiento]**, apartado **[Ejecución en avance]**.

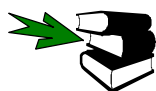
6.2 Entradas y salidas binarias

Si se excitan entradas y salidas individualmente, se puede hablar de entradas y salidas binarias. Las salidas binarias pueden tomar 2 estados: Low o High (nivel de tensión bajo o alto). Por ello, se tratan estas variables como del tipo BOOL.

Por consiguiente, las salidas pueden ser activadas con la variables del sistema

\$OUT[Nr] = TRUE y desactivadas con
\$OUT[Nr] = FALSE

El estado de una entrada \$IN[Nr] puede leerse dentro de una variable booleana o bien utilizarse como expresión booleana en las instrucciones de programa, interrupción o de disparo.



Informaciones adicionales se encuentran en el capítulo **[Control de ejecución del programa]**, en el capítulo **[Tratamiento de interrupciones]** y en el capítulo **[Trigger – Acciones de conmutación referentes a la trayectoria]**.

Las secuencias de instrucción

```

BOOL CONMUTADOR
:
CONMUTADOR = $IN[ 6 ]
IF CONMUTADOR == FALSE THEN
:
ENDIF

```

y

```

IF $IN[ 6 ] == FALSE THEN
:
ENDIF

```

tienen, por consiguiente, el mismo significado.

SIGNAL

En la KR C... tiene Ud. además, la posibilidad de asignarles nombres a cada una de las entradas/salidas. Para ello sirve la declaración de la señal. Esta deberá estar ubicada, como todas las declaraciones, en la parte de declaración del programa. Es decir que también se puede programar:

```

SIGNAL CONMUTADOR $IN[ 6 ]
:
IF CONMUTADOR == FALSE THEN
:
ENDIF

```

La variable conmutador es declarada internamente como booleana.



A las entradas y salidas del sistema pueden hacerse referencia también con \$IN y \$OUT. Sin embargo, las salidas del sistema están protegidas contra escritura. La entrada 1025 siempre es TRUE, la entrada 1026 siempre es FALSE. Estas entradas son utilizadas p. ej, en los datos de máquinas como variables “Dummy”. Se permite la multiutilización.

Como se aplican las entradas/salidas lo explica el ejemplo 6.1:



```

DEF BINSIG ( )
;----- Sección de declaraciones -----
EXT BAS (BAS_COMMAND:IN,REAL:IN )
DECL AXIS HOME
SIGNAL CANCELAR $IN[16]
SIGNAL IZQUIERDA $OUT[13]
SIGNAL CENTRO $OUT[14]
SIGNAL DERECHA $OUT[15]
;----- Inicialización -----
BAS (#INITMOV,0 ) ;Inicialización de velocidades,
                    ;aceleraciones, $BASE, $TOOL, etc.
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}

;----- Sección principal -----
PTP HOME ;Desplazamiento COI
IZQUIERDA=FALSE
CENTRO=TRUE ;en posición central
DERECHA=FALSE

WHILE CANCELAR==FALSE ;Cancelar con entrada 16 activa
  IF $IN[1] AND NOT IZQUIERDA THEN ;Entrada 1 activa
    PTP {A1 45}
    IZQUIERDA=TRUE ;en posición izquierda
    CENTRO=FALSE
    DERECHA=FALSE
  ELSE
    IF $IN[2] AND NOT CENTRO THEN ;Entrada 2 activa
      PTP {A1 0}
      IZQUIERDA=FALSE
      CENTRO=TRUE ;en posición central
      DERECHA=FALSE
    ELSE
      IF $IN[3] AND NOT DERECHA THEN ;Entrada 3 activa
        PTP {A1 -45}
        IZQUIERDA=FALSE
        CENTRO=FALSE
        DERECHA=TRUE ;en posición derecha
      ENDIF
    ENDIF
  ENDIF
ENDWHILE
PTP HOME
END

```

Mediante la activación de las entradas 1,2 o 3 el robot puede desplazarse a tres posiciones diferentes. Si el robot ha alcanzado la posición deseada, se muestra este hecho mediante la activación de las salidas correspondientes 13, 14 o bien 15.

Ya que estas salidas muestran siempre la posición actual del robot se puede evitar, mediante la consulta

```

IF $IN[1] AND NOT $OUT[13] THEN
:
ENDIF

```

de que el robot intente desplazarse de nuevo a la posición que ya tiene en cada ciclo de procesamiento del bucle While. El robot se desplaza sólo cuando esté activada la entrada (es decir, instrucción para el desplazamiento a la posición deseada) **y** la salida correspondiente **no** este activa (es decir que el robot aún no está en esa posición) (véase Tab. 21).

Con la activación de la entrada 16 se concluye el bucle While y por consiguiente, el programa.

\$IN [Nr] AND NOT \$OUT[Nr]		\$OUT[Nr]	
		TRUE	FALSE
\$IN[Nr]	TRUE	FALSE	TRUE
	FALSE	FALSE	FALSE

Tab. 21 Tabla de la verdad para una combinación lógica “AND NOT”

6.3 Entradas y salidas digitales

6.3.1 Declaración de la señal

Con la declaración de la señal no sólo se puede asignar a determinadas entradas/salidas un nombre, sino que además, se pueden agrupar varias entradas/salidas binarias a una sola señal digital. Con la declaración

```
SIGNAL SALI $OUT[10] TO $OUT[20]
```

se puede declarar ahora, p. ej., las salidas 10 hasta 20 a través de una variable interna declarada como **Entera** SALI y activada como palabra de 11 bit.

La salida digital declarada de este modo, se puede definir para cada asignación entera posible a la variable SALI, p. ej.:

```
SALI = 35
```

```
SALI = 'B100011'
```

```
SALI = 'H23'
```

- Las entradas/salidas deberán indicarse en la declaración de la señal, en orden ascendente, sin interrupciones.
- Como máximo, se pueden agrupar 32 entradas o salidas a una señal digital.
- Una señal puede aparecer en diferentes declaraciones de señales.

Si se agrupan las salidas de la 13 hasta la 15 del ejemplo 6.1 a una variable denominada POSICION, resulta el siguiente programa modificado:



```

DEF  BINSIG_D ( )

;----- Sección de declaraciones -----
EXT  BAS (BAS_COMMAND:IN,REAL:IN )
DECL AXIS HOME
SIGNAL CANCELAR $IN[16]
SIGNAL POSICION $OUT[13] TO $OUT[15]

;----- Inicialización -----
BAS (#INITMOV,0 ) ;Inicialización de velocidades,
                  ;aceleraciones, $BASE, $TOOL, etc.
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}

;----- Sección principal -----
PTP  HOME                ;Desplazamiento COI

POSICION='B010'          ;en posición central

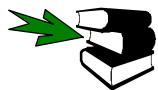
WHILE CANCELAR==FALSE    ;Cancelar con entrada 16 activa

    IF $IN[1] AND (POSICION<>'B001') THEN      ;Entrada 1 activa
        PTP {A1 45}
        POSICION='B001'                        ;en posición izquierda
    ELSE
        IF $IN[2] AND (POSICION<>'B010') THEN  ;Entrada 2 activa
            PTP {A1 0}
            POSICION='B010'                    ;en posición central
        ELSE
            IF $IN[3] AND (POSICION<>'B100') THEN;Entrada 3 activa
                PTP {A1 -45}
                POSICION='B100'                ;en posición derecha
            ENDIF
        ENDIF
    ENDIF

ENDWHILE

PTP  HOME
END

```



Informaciones adicionales se encuentran en este capítulo en el apartado **[Entradas digitales predefinidas]**.

6.3.2 Activar salidas en el punto de destino

En el momento que el robot haya alcanzado el punto de destino de un paso de movimiento, pueden activarse hasta 8 salidas referidas a la ejecución principal y sin detención de la ejecución en avance. Para ello sirve la siguiente instrucción:

```
$OUT_C[salida] = expresión booleana
```

Argumento	Tipo de dato	Significado
Salida	INT	Expresión aritmética, que define el número de la salida a activar. Se dispone de los números de salida 1 ... 1024
Expresión booleana	BOOL	Expresión lógica, que indica, que la salida correspondiente será puesta en "TRUE" o "FALSE"



Normalmente pueden utilizarse las salidas 1 ... 1024. Si correspondientemente se ha activado la variable "\$SET_IO_SIZE", se dispone también de 2048 o 4096 salidas.

Si el puntero de ejecución en avance alcanza la instrucción, se interpretará primeramente la expresión booleana. La expresión afectada se transformará en una constante. Al alcanzar el puntero de la ejecución principal, se activan entonces las salidas interpretadas.

Una asignación podría, por ejemplo, tener la siguiente forma:

```
$OUT_C[TOOL[ex]+WELD[y]] = ((NOT(x==100)) AND (SAFE==TRUE))
```

Después de efectuada la interpretación (puntero de ejecución en avance) de forma interna la instrucción podría tener, por ej., la siguiente forma:

```
$OUT_C[5] = TRUE
```

Con ello, al alcanzar el punto de destino (puntero de ejecución principal) la salida 5 es puesta en el valor "TRUE".



Al alcanzar el puntero de ejecución principal del programa, la salida es puesta al valor definido por la expresión booleana, válida en el instante de la interpretación, aún cuando este valor pudo haber sido modificado desde entonces.



Una instrucción "\$OUT_C[x]" contrariamente a "\$OUT[x]" no tiene influencia ninguna sobre la ejecución en avance del ordenador.

\$OUT_C[x] sólo puede ser escrita, para la lectura de una salida debe utilizarse "\$OUT[x]".



Una selección de pasos borra todas las asignaciones "\$OUT_C[x]" interpretadas hasta ese momento, pero no aquellas activadas. Esto es válido para una selección de paso y para un reset del programa.

Ejemplo PTP, LIN y CIRC

En instrucciones PTP, LIN y CIRC absolutas y relativas, la salida es activada inmediatamente después de la correspondiente instrucción de movimiento:



```
PTP P1
$OUT_C[10]=TRUE
$OUT_C[11]=FALSE
$OUT_C[12]=TRUE
LIN P2
```

Si en el punto "P1" se tiene una situación de parada exacta, se activan en ese momento las salidas 10 ... 12 de acuerdo con su definición.

Si, por el contrario, el punto "P1" es alcanzado por aproximación, las salidas 10 ... 12 recién serán activadas cuando se haya alcanzado el centro del entorno de aproximación. Si no es posible efectuar un posicionamiento aproximado, las salidas se activan en el punto "P1".



En los tipos de ejecución de programa "Single-Step" (MSTEP) e "I-Step" (ISTEP) se efectúa primeramente una detención de la ejecución en avance en el punto P1, las salidas no son activadas en este lugar. Recién al soltar y pulsar nuevamente la tecla de arranque se activarán las salidas definidas.



Un desplazamiento COI causa también una detención del procesamiento en avance. Recién después de un rearranque del programa, se activan las salidas de acuerdo con su definición.

Comparación con disparo (Trigger)

Tanto con "\$OUT_C[x]" como también con "Trigger" (disparo) pueden activarse salidas en la ejecución principal del programa y sin detención del procesamiento en avance.

\$OUT_C[x] :

```
PTP P1 C_PTP
$OUT_C[5]=TRUE
PTP P2
```

Trigger:

```
TRIGGER WHEN DISTANCE=1 DELAY=0 DO $OUT[5]=TRUE
PTP P1 C_PTP
PTP P2
```

En ambos casos es activada la salida 5, en el momento de alcanzar el centro del entorno de aproximación de "P1".

Puntero de ejecución principal

Si los punteros de procesamiento en avance y de ejecución principal son idénticos, es decir, el robot no ejecuta actualmente ningún paso de movimiento, la asignación se efectúa inmediatamente al alcanzar la instrucción "\$OUT_C[]".



```
PTP P1
WAIT FOR $IN[22]==TRUE
$OUT_C[12]=FALSE
PTP P2
```

En el ejemplo, en la línea “WAIT...” se crea un stop de la ejecución en avance, mientras la entrada 22 se encuentre en “FALSE”. En el momento que en la entrada 22 se presente el estado “TRUE”, correspondientemente se activa la salida 12.

Signal (Señal)

Una salida también puede ser activada mediante una convención de señal de 1 bit.



SIGNAL Test \$OUT_C[7]

PTP P1

Test = TRUE

PTP P2

En el ejemplo, la salida 7 es puesta en “TRUE”, en el momento que el puntero de ejecución principal haya alcanzado el paso de movimiento “P1”.

Bajo las siguientes condiciones no está permitido “\$OUT_C[x]”:

- Dentro de programas de interrupción o declaraciones de interrupción
- En un programa Submit
- En combinación con “\$CYCFLAG[x]”
- Dentro de una instrucción de Trigger (disparo)
- En relación con la corrección de variables

6.4 Salidas de impulso

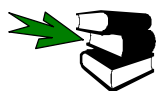
PULSE

Con la instrucción de PULSE pueden activarse o desactivarse cada una de las salidas durante un tiempo concreto. La instrucción

`PULSE ($OUT[4], TRUE, 0.7)`

pone por ejemplo, a nivel alto, la salida 4 durante 0,7 segundos. Este impulso puede procesarse paralelamente con el programa del robot (durante este proceso no se detiene el intérprete).

En vez de la indicación directa de la salida con `$OUT[Nr]` puede colocarse una variable de señal.



Informaciones adicionales se encuentran en este capítulo en el apartado **[Entradas y salidas binarias]**.

Las temporizaciones de impulsos realizables están comprendidos entre 0.012 y 2^{31} segundos. Los pasos son 0,1 segundos y el control redondea todos los valores en una décima.



- Se pueden programar como máx. 16 salidas de impulsos simultáneamente.
- Se pueden programar tanto impulsos High (alto) como impulsos Low (bajo).
- Con el “Resetear programa” o “Cancelar programa” se interrumpe el impulso.
- Un impulso activado puede ser influenciado mediante interrupciones.
- Las salidas de impulsos pueden ser programadas también desde el nivel de la unidad de control.
- La instrucción PULSE origina una parada del procesamiento en avance. Sólo en la instrucción TRIGGER se ejecuta siguiendo el movimiento.



Un impulso NO es interrumpido con

- **PARADA DE EMERGENCIA**, parada de operación o parada de error,
- **llegar al fin del programa (instrucción END)**,
- **soltar la tecla de arranque**, si el impulso se programó anterior a la primera instrucción de movimiento y el robot no ha alcanzado aún COI.

En el programa siguiente encontrará algunos ejemplos de utilización de la instrucción PULSE:



```

DEF PULSIG ( )

;----- Sección de declaraciones -----
EXT BAS (BAS_COMMAND :IN,REAL :IN )
DECL AXIS HOME
INT I
SIGNAL OTTO $OUT[13]

;----- Inicialización -----
BAS (#INITMOV,0 ) ;Inicialización de velocidades,
                  ;aceleraciones, $BASE, $TOOL, etc.
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}

FOR I=1 TO 16
$OUT[I]=FALSE      ;puesta de todas las salidas a nivel LOW
ENDFOR

;----- Sección principal -----
PULSE ($OUT[1],TRUE,2.1) ;El impulso llega directo durante 2.1s
PTP HOME                ;Desplazamiento COI

OTTO=TRUE                ;Poner la salida 13 a TRUE
PTP {A3 45,A5 30}
PULSE (OTTO,FALSE,1.7) ;Impulso LOW durante 1.7s en salida 13
                        ;El impulso llega después del
                        ;desplazamiento

WAIT SEC 2

FOR I=1 TO 4
PULSE ($OUT[I],TRUE,1) ;poner una tras otra las salidas 1-4
WAIT SEC 1                ;durante 1s a High
ENDFOR

;Disposición de un impulso respecto a la trayectoria
TRIGGER WHEN DISTANCE=0 DELAY=50 DO PULSE ($OUT[8],TRUE,1.8)
LIN {X 1391,Y -319,Z 1138,A -33,B -28,C -157}

PTP HOME
CONTINUE                ;Evitar parada de avance para la salida 15
PULSE ($OUT[15],TRUE,3) ;El impulso llega directo (en el
                        ;avance)

                        ;en salida 16
PULSE ($OUT[16],TRUE,3) ;El impulso llega después del
                        ;desplazamiento HOME
END                        ;y queda aún después de END

```

Observe detenidamente estos ejemplos, a partir de que momento están presentes los impulsos programados en las salidas: generalmente la instrucción de PULSE produce siempre una parada del procesamiento en avance. Es decir, que el impulso está presente después de finalizar el movimiento.

Existen dos formas de evitar la parada del procesamiento en avance:

- Programación de una instrucción CONTINUE justo antes de antes a una instrucción PULSE
- Utilización de una instrucción PULSE dentro de una instrucción TRIGGER (acción de conmutación referente a una trayectoria).



Informaciones adicionales se encuentran en el capítulo **[Programación de movimiento]**, apartado **[Procesamiento en avance]** (CONTINUE) y capítulo **[Trigger - Acciones de conmutación referentes a la trayectoria]** (TRIGGER).

6.5 Entradas y salidas analógicas

Además de las entradas y salidas binarias, la KR C... reconoce también entradas y salidas analógicas. A través de los sistemas de bus opcionales, la KR C... pone a disposición 8 entradas analógicas y 16 salidas analógicas. Las salidas pueden ser leídas y escritas mediante las variables de sistema \$ANOUT[1] ... \$ANOUT[16], las entradas sólo pueden ser leídas con las variables \$ANIN[1] ... \$ANIN[8].

ANIN
ANOUT

Con las entradas y las salidas pueden ser interrogadas tanto estáticamente como dinámicamente, es decir mediante una interrogación permanente en un ciclo de interpolación (actualmente es de 12 ms). Mientras que la lectura y escritura estática se produce igual que con las señales binarias mediante una simple asignación de valor, el procesamiento dinámico utiliza instrucciones especiales de ANIN y ANOUT.

6.5.1 Salidas analógicas

Los valores de emisión para las 16 salidas analógicas de la KR C... se encuentran comprendidos entre -1.0 ... +1.0 y están normalizados a una tensión de salida de ± 10.0 V. Si el valor de salida supera la tolerancia de ± 1.0 , el valor es recortado.

Para activar un canal analógico, se la asigna simplemente un valor a la variable \$ANOUT correspondiente:

```
$ANOUT[2] = 0.5 ;el canal analógico 2 se pone a +5 V
```

o bien

```
REAL V_PEGAM  
L
```

```
V_PEGAM = -0.9
```

```
$ANOUT[15] = V_PEGAM ;el canal analógico 15 se pone a -9 V
```

Estas asignaciones son estáticas, ya que el valor del canal excitado cambiará cuando se le haya asignado a la variable de sistema \$ANOUT[Nr] correspondiente un nuevo valor explícito.

A veces se desea que una salida analógica determinada vaya adquiriendo nuevos valores en un tiempo de ciclo prefijado durante el procesamiento del programa. Esta salida analógica se realiza con la instrucción ANOUT.

```
ANOUT ON HILO = 0.8 * V_HILO
```

Así p. ej. mediante una simple asignación de valor a la variable V_HILO, puede variar la salida analógica especificada con la variable de señal HILO. La tensión en la salida correspondiente sigue así a la variable V_HILO.

Previamente se le ha de declarar la variable HILO, con la declaración SIGNAL, p. ej.:

```
SIGNAL HILO $ANOUT[2]
```

Con

```
ANOUT OFF HILO
```

finaliza de nuevo la emisión analógica cíclica.

La expresión actualizada cíclica que se ha de indicar para el cálculo del valor de emisión analógica, no debe sobrepasar un nivel de complejidad determinado. Por tal motivo, la sintaxis admisible está limitada y orientada a la tecnología. La sintaxis completa es

ANOUT ON

```
ANOUT ON Nombre de señal = Factor * Elemento de regulación  
± Offset <DELAY=t <MINIMUM=U1 <MAXIMUM=U2
```

para el inicio de la emisión analógica cíclica, o bien,

ANOUT OFF

```
ANOUT OFF Nombre de señal
```

para la finalización. El significado de cada uno de los argumentos se puede ver en Tab. 22.

Argumento	Tipo de datos	Significado
Nombre de señal	REAL	Variable de señal que especifica la salida analógica (debe estar declarado con SIGNAL). No se permite la indicación directa de \$ANOUT[Nr].
Factor	REAL	Cualquier factor pudiéndose tratar de variable, nombre de señal o constante.
Elemento de regulación	REAL	A través del elemento de regulación se influye la salida analógica. Puede tratarse de una variable o nombre de señal.
Offset	REAL	Opcionalmente puede programarse un offset como elemento de regulación. El offset debe ser una constante.
t	REAL	Con la clave DELAY y una indicación de tiempo positiva o negativa, se puede retardar (+) o adelantar (-) opcionalmente una señal de salida calculada cíclicamente.
U1	REAL	La palabra clave MINIMUM define la tensión mínima que aparece en la salida. Los valores permitidos se encuentran entre -1.0 ... 1.0 (que corresponde a -10V ... +10V). El valor mínimo debe ser menor que el valor máximo, siempre que se utilicen ambos valores. Como valor también se permite una variable, una componente de estructura o un elemento de grupo.
U2	REAL	La palabra clave MAXIMUM define la tensión máxima que aparece en la salida. Los valores permitidos se encuentran entre -1.0 ... 1.0 (que corresponde a -10V ... +10V). El valor máximo debe ser mayor que el valor mínimo, siempre que se utilicen ambos valores. Como valor también se permite una variable, una componente de estructura o un elemento de grupo.

Tab. 22 Argumentos en la instrucción ANOUT



Los parámetros opcionales "Minimum" y "Maximum" no están disponibles en los formularios inline, ya que están exclusivamente a disposición para la tecnología "Aplicación de pegamentos".



En este ejemplo se han definido tanto un valor mínimo (Minimum) como un valor máximo (Maximum). Las definiciones correspondientes se escriben “MINIMUM=0.3” y “MAXIMUM=0.95”.

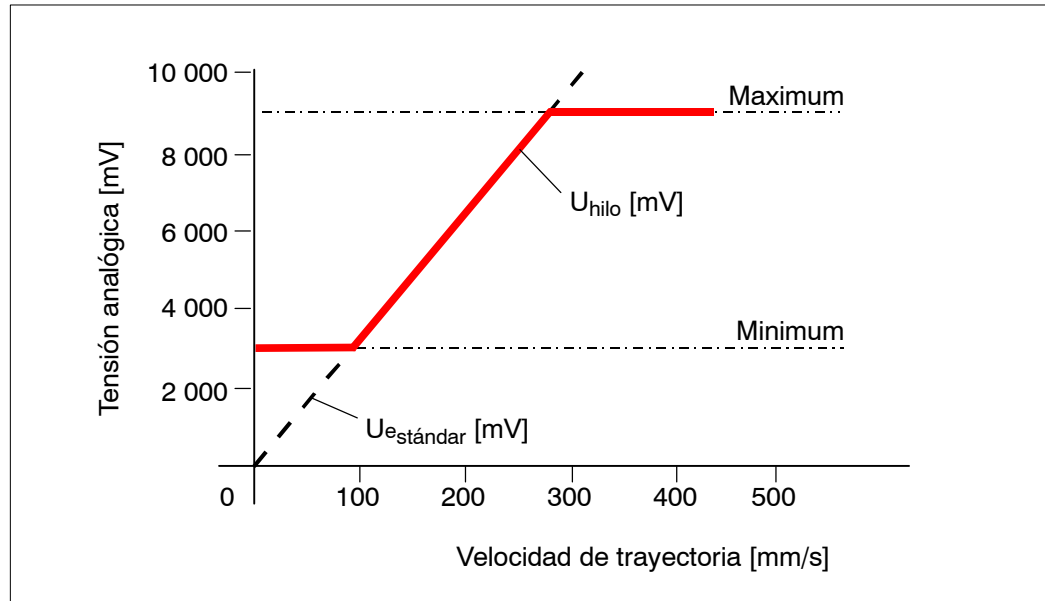


Fig. 34 Tensión analógica en dependencia de la velocidad de trayectoria

6.5.2 Entradas analógicas

Las 8 entradas analógicas de la KR C... pueden leerse en una variable REAL a través de las variables \$ANIN[1] hasta \$ANIN[8] mediante una simple asignación de valor:

```
REAL PARTE
:
PARTE = $ANIN[3]
```

o bien

```
SIGNAL SENSOR3 $ANIN[ 3 ]
REAL PARTE
:
PARTE = SENSOR3
```

Los valores en \$ANIN[Nr] se mueven entre +1.0 y -1.0 que representan una tensión de entrada de +10 V hasta -10 V.

ANIN

Para la lectura cíclica de entradas analógicas se usa la instrucción ANIN. Con ella se puede tomar lectura hasta 3 entradas analógicas simultáneamente. La lectura se realiza en el ciclo de interpolación.

Con la secuencia de instrucción

```
SIGNAL SENSOR3 $ANIN[ 3 ]
REAL PARTE
:
ANIN ON PARTE = 1 * SENSOR3
```

puede leer la entrada analógica 3 cíclicamente y con la instrucción

```
ANIN OFF SENSOR3
```

concluir la lectura.

Se ha de observar que a la vez sólo pueden estar activas, como máx., 3 instrucciones ANIN ON. Para ambas instrucciones se permite acceder a la misma interfaz analógica o asignar la misma variable.

La sintaxis completa para la lectura cíclica de una entrada analógica es la siguiente:

ANIN ON

ANIN ON Valor = Factor * Nombre de señal (± Offset

La finalización del seguimiento cíclico la efectúa con

ANIN OFF

ANIN OFF Nombre de señal

El significado de los argumentos puede tomarlos de Tab. 23.

Argumento	Tipo de datos	Significado
Valor	REAL	El valor puede ser una variable o un nombre de señal (de salida). En Valor se registra el resultado de la lectura cíclica.
Nombre de señal	REAL	Variable de señal que especifica la entrada analógica (deberá estar declarada con SIGNAL). No se permite la indicación directa de \$ANIN[Nr].
Factor	REAL	Cualquier factor pudiéndose tratar de variable, nombre de señal o constante.
Offset	REAL	Opcionalmente puede programarse un offset. El offset puede ser una constante, una variable o un nombre de señal.

Tab. 23 Argumentos en la instrucción ANIN

En el ejemplo mostrado a continuación se ilustran las instrucciones para las entradas y salidas analógicas. Con la ayuda de la variable del sistema \$TECHIN[1] y un sensor de seguimiento de trayectoria conectado a la entrada analógica, se puede realizar p. ej. una corrección de trayectoria durante el movimiento. La variable \$VEL_ACT, que contiene permanentemente la velocidad de trayectoria, puede ir valorada con los factores correspondientes utilizándola para la salida analógica proporcional a la velocidad, p. ej. para controlar la cantidad de pegamento en una aplicación de adhesivo.



```

DEF  ANSIG ( )

;----- Sección de declaraciones -----
EXT  BAS (BAS_COMMAND :IN,REAL :IN )
DECL AXIS HOME
INT I
SIGNAL ADHESIVO $ANOUT[1]      ;Abertura de boquilla para adhesivo
SIGNAL CORRECCIÓN $ANIN[5]    ;Sensor de seguimiento de trayecto-
ria

;----- Inicialización -----
BAS (#INITMOV,0 ) ;Inicialización de velocidades,
                  ;aceleraciones, $BASE, $TOOL, etc.
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}

FOR I=1 TO 16
$ANOUT[I]=0          ;Poner todas las salidas a 0 V
ENDFOR

;----- Sección principal -----
PTP HOME              ;Desplazamiento COI

$ANOUT[3] = 0.7      ;Salida analógica 3 a 7V

IF $ANIN[1] >= 0 THEN ;Proceso de pegado sólo si la entrada
                      ;analógica 1 tiene tensión positiva

    PTP POS1

    ;Corrección de trayectoria de acuerdo con la señal del sensor
    ;con la ayuda de las variables del sistema $TECHIN
    ANIN ON $TECHIN[1] = 1 * CORRECCIÓN + 0.1

    ;salida analógica proporcional a la velocidad
    ;variables de sistema $VEL_ACT contiene la velocidad de
    ;trayectoria actual
    ANOUT ON ADHESIVO = 0.5 * $VEL_ACT + 0.2 DELAY = -0.12

    LIN POS2
    CIRC POSINTERMED,POS3
    ANOUT OFF ADHESIVO
    ANIN OFF CORRECCIÓN

    PTP POS4

ENDIF

PTP HOME
END

```

6.6 Entradas digitales predefinidas

La unidad de control pone a disposición 6 entradas digitales que pueden ser leídas a través de las variables de señal \$DIGIN1...\$DIGIN6. Las entradas forman parte de las entradas de usuario normales. Pueden tener una longitud de 32 bit con su salida Strobe correspondiente.

La configuración de las entradas digitales se realiza en los datos de máquina: "/mada/steu/\$maschine.dat". Con una declaración de la señal se establece primeramente el rango y el tamaño de la entrada digital:

```
SIGNAL $DIGIN3 $IN[1000] TO $IN[1011]
```

A través de otras variables de sistema \$DIGIN1CODE...\$DIGIN6CODE, \$STROBE1...\$STROBE6 y \$STROBE1LEV...\$STROBE6LEV se prefijan las interpretaciones del signo, las salidas Strobe correspondientes y la clase de la salida Strobe:

```
DECL DIGINCODE $DIGIN3CODE = #UNSIGNED ;sin signo
```

```
SIGNAL $STROBE3 $OUT[1000] ;Prefijar la salida Strobe
```

```
BOOL $STROBE3LEV = TRUE ;Strobe es un impulso high
```

Una salida Strobe es una salida de la KR C... con un impulso determinado que produce la congelación de la señal para la lectura de un aparato externo (p. ej. codificador).

Mientras que entradas digitales diferentes pueden acceder a la misma entrada, las señales Strobe, NO pueden escribir la misma salida.

El rango de valores de \$DIGIN1...\$DIGIN6 depende de la longitud del grupo de bit definida, así como de la interpretación del signo (#SIGNED o bien #UNSIGNED) a partir de:

12 bits con signo (#SIGNED) rango de valores: -2048...2047

12 bits sin signo (#UNSIGNED) rango de valores: 0...4095

Las entradas digitales pueden interrogarse estáticamente o por medio de una asignación de valor convencional:

```
INT NÚMERO
```

```
:
```

```
NÚMERO = $DIGIN2
```

DIGIN

o bien cíclicamente, con una instrucción DIGIN:

```
INT NÚMERO
```

```
:
```

```
DIGIN ON NÚMERO = FACTOR * $DIGIN2 + OFFSET
```

```
:
```

```
DIGIN OFF $DIGIN2
```

Se permiten hasta 6 instrucciones DIGIN ON al mismo tiempo. En la instrucción DIGIN ON, también se puede acceder a la señal de entrada analógica (p. ej. como FACTOR). La sintaxis es totalmente análoga a la instrucción ANIN ON:

DIGIN ON

DIGIN ON Valor = Factor * nombre de señal <± Offset

DIGIN OFF

DIGIN OFF Nombre de señal

El significado de los argumentos puede tomarlos de Tab. 24.

Argumento	Tipo de datos	Significado
Valor	REAL	El <code>Valor</code> puede ser una variable con un nombre de señal (de salida). En <code>Valor</code> se deposita el resultado de la lectura cíclica.
Nombre de señal	REAL	Variable de señal que especifica la entrada digital. Sólo se permiten <code>\$DIGIN1...\$DIGIN6</code> .
Factor	REAL	Cualquier factor que puede ser una variable, nombre de señal o una constante.
Offset	REAL	Opcionalmente se puede programar un offset. El offset puede ser una constante, una variable o un nombre de señal.

Tab. 24 Argumentos en la instrucción DIGIN

7 Subprogramas y funciones

Para reducir la tarea de escritura de apartados de programa que se repiten con frecuencia durante la programación así como la longitud de programa, se introducen subprogramas y funciones como construcción de lenguaje.

Un efecto que no se ha de menospreciar en subprogramas y funciones es la utilización repetida de algoritmos registrados dentro de otros programas así como la aplicación de subprogramas en la estructuración del programa. Este tipo de estructuración puede conllevar la ordenación jerárquica de estructura de modo que cada uno de los subprogramas puedan llamarse por programas de niveles superiores, procesar tareas parciales y traspasar los resultados.

7.1 Declaraciones

Un subprograma o una función son partes de programas separadas, con encabezamiento de programa, sección de declaraciones y de instrucciones, que pueden llamarse desde cualquier parte del programa según convenga. Una vez procesado el subprograma o la función se produce un salto atrás a la instrucción subsiguiente de la llamada del subprograma (véase Fig. 35).

Desde un subprograma o bien una función, pueden llamarse otros subprogramas y/o funciones. La profundidad de encajado permitida en este caso es de 20. Si se excede este número aparece un mensaje de fallo indicando "REBASE APILAMIENTO DE PROGRAMA". La llamada recursiva de subprograma o de funciones esta permitida. En otras palabras, es decir que un subprograma o una función puede llamarse a sí mismo.

DEF

Todos los subprogramas son declarados exactamente igual que los programas principales con la convención DEF más nombre y cerrándose con END p. ej.:

```
DEF SUBPROG ( )
...
END
```

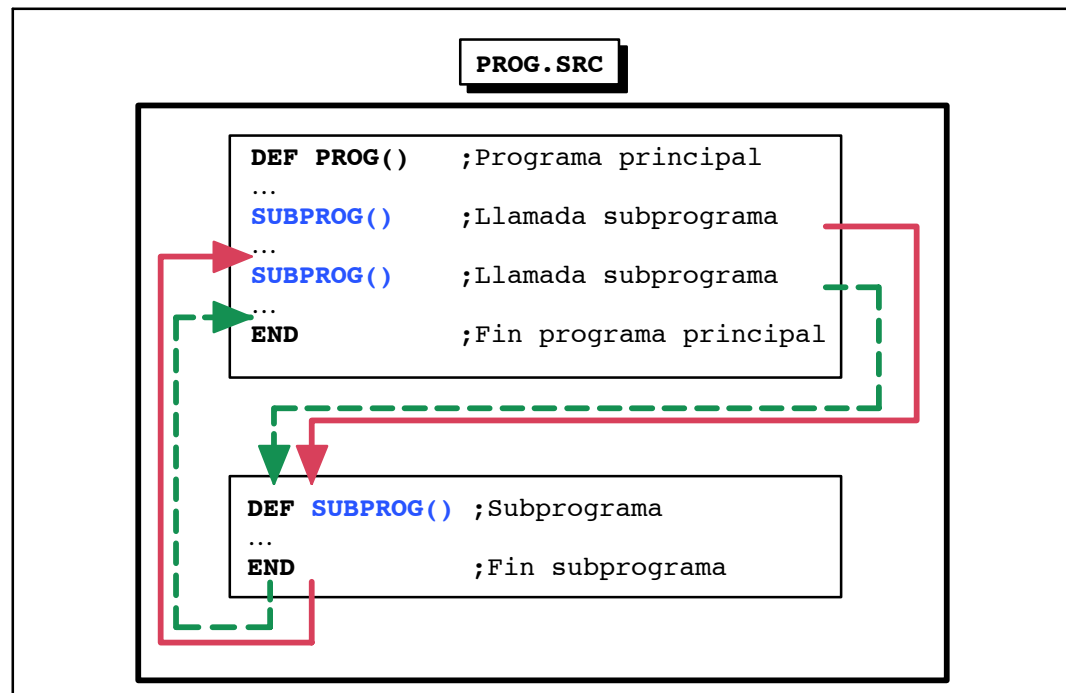


Fig. 35 Llamada subprograma y salto atrás

Una función es algo similar a un subprograma, con la única diferencia que el nombre de programa sirve a la vez de variable de un determinado tipo de datos. De esta forma se puede transferir el resultado de la función a una variable mediante una simple asignación de valor. En la convención de funciones con claves especiales `DEFFCT`, se ha de indicar, además del nombre de la función, el tipo de datos de función. Una función se cierra con `ENDFCT`. Ya que una función debe transferir un valor, deberá especificarse este valor delante de la instrucción `ENDFCT` con la instrucción `RETURN`. Ejemplo:

```
DEFFCT INT FUNCIÓN( )
...
RETURN( X )
ENDFCT
```

local Generalmente se hace una diferenciación entre subprogramas o funciones locales y globales, respectivamente. En los subprogramas o funciones locales el programa principal y las funciones/subprogramas se encuentran en el mismo archivo SRC. El archivo ostenta el mismo nombre que el programa principal. El programa principal estará en el texto fuente siempre en primer lugar, mientras que los subprogramas y funciones pueden estar ordenados en cualquier orden y cantidad después del programa principal.

global Las funciones locales/los subprogramas pueden llamarse sólo dentro de los archivos SRC en los cuales fueron programados. Si se desea que las llamadas de subprogramas /funciones se sucedan desde otros programas, deberán ser del tipo global, es decir, deberán estar memorizados en un fichero SRC propio. De este modo, cada programa es un subprograma si es llamado desde otro programa (programa principal, subprograma o función).



- En subprogramas locales, o bien, en funciones, se conocen todas las variables declaradas en la lista de datos del programa principal. Variables que han sido declaradas en el programa principal (archivo SRC) son las así llamadas “Variables Runtime” y sólo deben ser utilizadas en el programa principal. El intento de utilizar estas variables en un subprograma, lleva al correspondiente mensaje de fallo. En los subprogramas o funciones globales, las variables declaradas dentro del programa principal no son conocidas.
- En el programa principal, las variables declaradas en subprogramas o funciones globales no son conocidas.
- Un programa principal no puede acceder a subprogramas o funciones de otro programa principal.
- El nombre de subprogramas/funciones pueden tener como máximo 24 caracteres de longitud. En caso de subprogramas/funciones globales, sólo puede tener como máximo 20 caracteres de longitud (debido a la extensión de archivo).

Para que el subprograma global sea reconocido por el programa principal, sólo es necesario efectuar su llamada dentro del mismo (por ej: `PROG_2()`). Indicando la lista de parámetros (véase 7.2) también viene claramente fijada la ocupación de memoria necesaria.

Ejemplos:

```
PROG_3( )
FUNCIÓN( REAL: IN )
```

En la fig. 36 se representa la diferencia entre subprogramas/funciones locales y globales: `PROG.SRC`, `PROG_1.SRC` y `PROG_3.SRC` son programas principales independientes, `PROG_2FUN.SRC` es una función. Por la llamada de un programa (por ej. `PROG_1.SRC`) desde `PROG.SRC`, éste es declarado automáticamente como subprograma global. `LOCAL()` es un subprograma local, `LOCALFUN()` una función local de `PROG.SRC`.

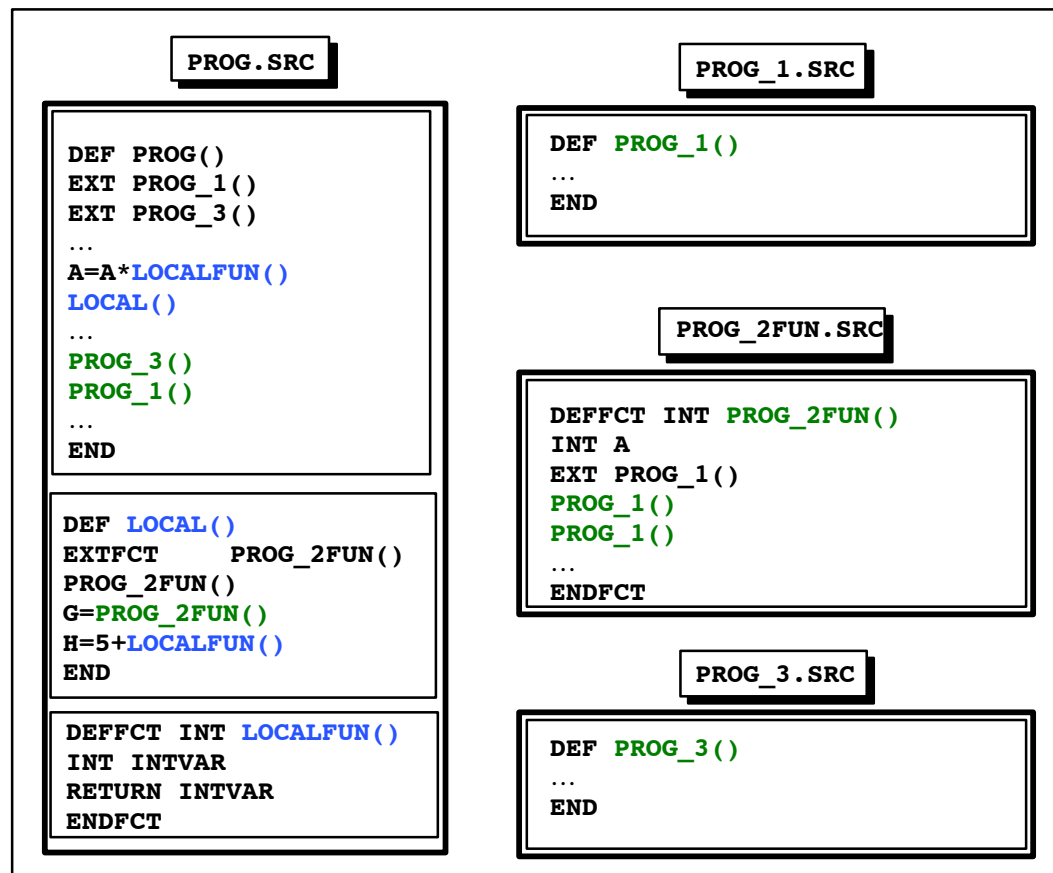


Fig. 36 Diferencia entre subprogramas locales y globales

7.2 Llamada y transferencia de parámetros

La llamada de un subprograma se realiza simplemente mediante la indicación del nombre del subprograma y paréntesis normales. De este modo tiene el aspecto de una instrucción (v. cap. 1.1), p. ej.:

```
SUBPROG1 ( )
```

Una llamada de función es una forma especial de asignación de valor. Por consiguiente, una función no puede estar sola, sino que el valor de la función deberá venir contenida dentro del marco de la expresión de una variable asignada al mismo tipo de datos, p. ej.:

```
INTVAR = 5 * INTFUNCIÓN( ) + 1
REALVAR = REALFUNCIÓN( )
```

Lista de
parámetros

En subprogramas y funciones locales, se conocen todas las variables declaradas en la lista de datos del programa principal. Contrariamente a ello, en subprogramas globales, estas variables son desconocidas. A través de una lista de parámetros puede transferirse también valores a subprogramas y funciones globales.

Es razonable, en la mayoría de los casos, añadirle también a la transferencia en subprogramas y funciones locales, la lista de parámetros debido a que se puede efectuar una separación clara entre el programa principal y el subprograma/función: las variables declaradas en el programa principal (archivo SRC) sólo son utilizadas en este, todas las transferencias en subprogramas y funciones (local y global) se efectúan por medio de listas de parámetros. Con esta forma de programación estructurada se reducen notablemente los errores de programación.

Para la transferencia de parámetros existen dos mecanismos diferentes:

- **Call by value (IN)**

En este tipo de transferencia, se transfiere un **valor** del programa principal a una variable del subprograma o función. El valor transferido puede tratarse de una constante, una variable, una llamada de función o una expresión. En caso de distintos tipos de datos, se realiza siempre que sea posible una adaptación de tipos.

- **Call by reference (OUT)**

A través de "Call by reference" se transfiere sólo la **dirección** de una variable del programa principal al subprograma o bien, a la función. El subprograma o bien función llamado puede sobrescribir seguidamente la zona de memoria mediante un nombre de variable propio y con ello, modificar el valor de la variable en el programa principal. Por consiguiente, deberán ser idénticos los tipos de datos utilizados, en este caso ya no se puede realizar una adaptación de tipos.

En Fig. 37 se muestra la diferencia entre ambos métodos. Mientras que la variable x con "Call by value" permanece invariable en el programa principal debido a la zona de memoria separada, es sobrescrita con "Call by reference" en la función por la variable NÚMERO.

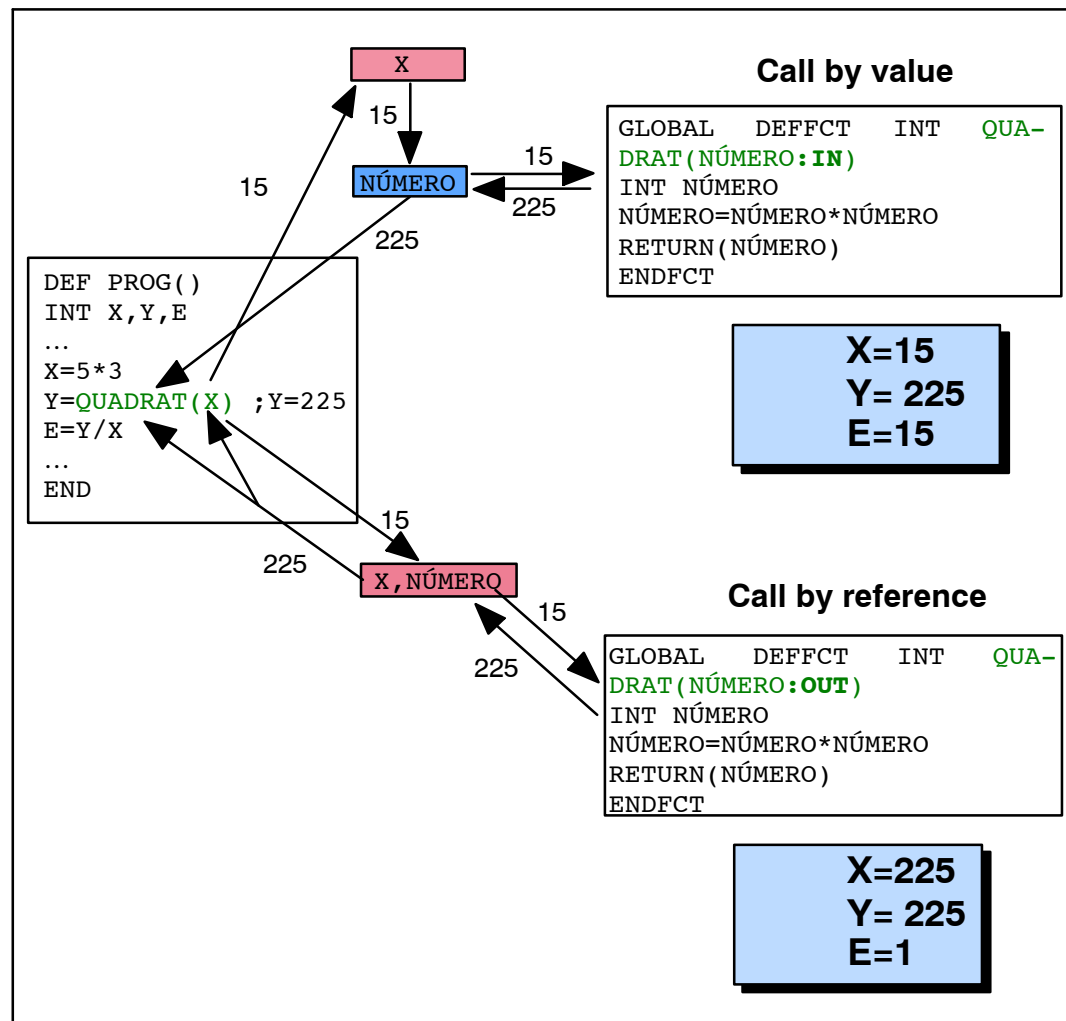


Fig. 37 Diferencia entre "Call by value" y "Call by reference"

"Call by value" es indicado en el subprograma o encabezamiento de función mediante la clave **IN** detrás de cada variable en la lista de parámetros. "Call by reference" se obtiene mediante la indicación de **OUT**. OUT también es el ajuste por defecto. Ejemplo:

```
DEF CALCULA(X:OUT,Y:IN,Z:IN,B)
```

Si el subprograma global o la función global a ser llamada no ha sido declarada como GLOBAL, se ha de indicar en la especificación externa dentro del programa principal, que tipo de datos dispone la variable respectiva, y que mecanismo de transferencia se ha de utilizar. OUT de nuevo es el ajuste por defecto. Ejemplo:

```
EXTFCT REAL FUNC1 (REAL:IN,BOOL:OUT,REAL,CHAR:IN)
```

Como utilizar IN y OUT deberá clarificarse con el ejemplo siguiente. El subprograma y la función se suponen globales.



```

DEF PROG ( )
CALCULA ( INT:OUT, INT:IN, INT:IN)
FUNC1 (REAL:IN, REAL:OUT, REAL:OUT, REAL:IN, REAL:OUT)
INT A,B,C
REAL D,E,F,G,H,X

A = 1
B = 2
C = 3
D = 1
E = 2
F = 3
G = 4
H = 5

CALCULA (A,B,C)
      ;A ahora es 11
      ;B ahora es 2
      ;C ahora es 3

X = FUNC1(H,D,E,F,G)
      ;D ahora es 3
      ;E ahora es 8
      ;F ahora es 3
      ;G ahora es 24
      ;H ahora es 5
      ;X ahora es 15

END
DEF CALCULA(X1:OUT,X2:IN,X3:IN)                                ;subprog. global
INT X1,X2,X3
X1=X1+10
X2=X2+10
X3=X3+10
END
DEFFCT REAL FUNC1(X1:IN,X2:OUT,X3:OUT,X4:IN,X5:OUT);función global
REAL X1,X2,X3,X4,X5
X1 = X1*2
X2 = X2*3
X3 = X3*4
X4 = X4*5
X5 = X5*6
RETURN(X4)
ENDFCT

```

En la transferencia de un campo, el campo deberá estar declarado nuevamente en el subprograma o en la función, pero sin estar indexado. Para ello véase el ejemplo siguiente en el cual se doblan los valores de un campo X[] (la función es global):



```

DEF  ARRAY ( )
EXT  BAS (BAS_COMMAND:IN,REAL:IN)
INT X[5]      ;Declaración de campo
INT I
EXT  DOBLE (INT[:OUT)

BAS (#INITMOV,0)

FOR I=1 TO 5
    X[I]=I      ;Array X[] inicializar
ENDFOR        ;X[1]=1,X[2]=2,X[3]=3,X[4]=4,x[5]=5

DOBLE (X[]) ;Llamada del subprograma con parámetros de campo
              ;X[1]=2,X[2]=4,X[3]=6,X[4]=8,X[5]=10
END

DEF  DOBLE (A[:OUT)
INT A[]      ;Nueva declaración del campo
INT I
FOR I=1 TO 5
    A[I]=2*A[I] ;Doblar los valores de campo
ENDFOR
END

```

En la transferencia de campos multidimensionales, tampoco se indican índices, sin embargo, la dimensión del campo deberá ser especificada mediante la indicación de comas, Ejemplos:

A[,] para campos bidimensional
A[, ,] para campos tridimensional

8 Tratamiento de interrupciones

Al utilizar robots en instalaciones de producción complejas, existe la necesidad que el robot reaccione de inmediato y en forma directa a determinados eventos externos o internos, pudiéndolos ejecutar acciones paralelas al proceso del robot. Esto significa que se debe interrumpir un programa en curso del robot e iniciar un programa o función de interrupción. Una vez procesado el programa de interrupción, deberá continuar el programa de robot interrumpido si no se ha especificado algo distinto.

La interrupción o bien arranque directo de un programa lo posibilitan las instrucciones de interrupción. Con ellas, el usuario tiene la posibilidad de reaccionar mediante programa, a un evento determinado que se ha producido, no sincronizado en el tiempo, con la secuencia del programa.

Las interrupciones pueden ser activadas por

- aparatos tales como sensores, unidades periféricas, etc.,
- mensajes de fallo,
- el usuario, o
- los circuitos de seguridad.

Por ejemplo después de accionar un pulsador de parada de emergencia, se llama una rutina de interrupción que resetea determinadas señales de salida (programa preparado `IR_STOPM.SRC`).

8.1 Declaración

Antes de poder activar una interrupción, deberán definirse previamente las posibles causas de la misma y las reacciones respectivas del sistema.

INTERRUPT

Esto se realiza con las declaraciones de interrupción, teniendo en cuenta que a cada interrupción se le ha de asignar una prioridad, un evento y la rutina de interrupción que ha de activar. La sintaxis completa es:

INTERRUPT DECL Prioridad WHEN Evento DO Subprograma

Para el significado que tienen los argumentos véase Tab. 25.

Argumento	Tipo de datos	Significado
Prioridad	INT	Expresión aritmética que indica la prioridad de la interrupción. Están disponibles los niveles de prioridad del 1...39 y 81...128. Los valores 40...80 están reservados para una asignación automática de prioridades por parte del sistema. La interrupción del nivel 1 tiene la máxima prioridad.
Evento	BOOL	Expresión lógica que define el evento de interrupción. Se permite: <ul style="list-style-type: none"> • una constante booleana • una variable booleana • un nombre de señal • una comparación
Subprograma		Nombre del programa de interrupción que se ha de ejecutar cuando se produzca un evento.

Tab. 25 Argumentos en la declaración de interrupciones

La instrucción

```
INTERRUPT DECL 4 WHEN $IN[3]==TRUE DO UP1()
```

declara p. ej. una interrupción de prioridad 4, que llama el subprograma UP1 () en el momento que la entrada 3 pasa a nivel high.

¡La declaración de interrupción es una instrucción, por consiguiente, no debe estar colocada en la sección de declaraciones!

Una interrupción (Interrupt) recién es reconocida a partir del nivel del programa en la cual fue declarada. En los niveles más altos de programa, a pesar de estar activada, la interrupción no es reconocida. Es decir, una interrupción declarada en un subprograma, no es conocida en el programa principal (ver Fig. 38).

GLOBAL

Pero si una interrupción es declarada como GLOBAL, entonces puede estar declarada en cualquier subprograma, y no pierde su validez al abandonar este nivel (ver . Fig. 38).

```
GLOBAL INTERRUPT DECL 4 WHEN $IN[3]==TRUE DO UP1()
```



- Una declaración puede ser sobrescrita en todo momento por una nueva.
- Una interrupción GLOBAL se diferencia de una interrupción normal, en que al abandonar el subprograma en el cual ha sido declarada, sigue teniendo validez.
- Simultáneamente pueden estar declaradas, como máximo, 32 interrupciones.
- En la condición de interrupción no se puede acceder a variables o componentes de estructura.
- A la subrutina de interrupción no deben ser entregados como parámetros, variables de duración temporal, excepto que se trate de GLOBAL o variables declaradas en la lista de datos.

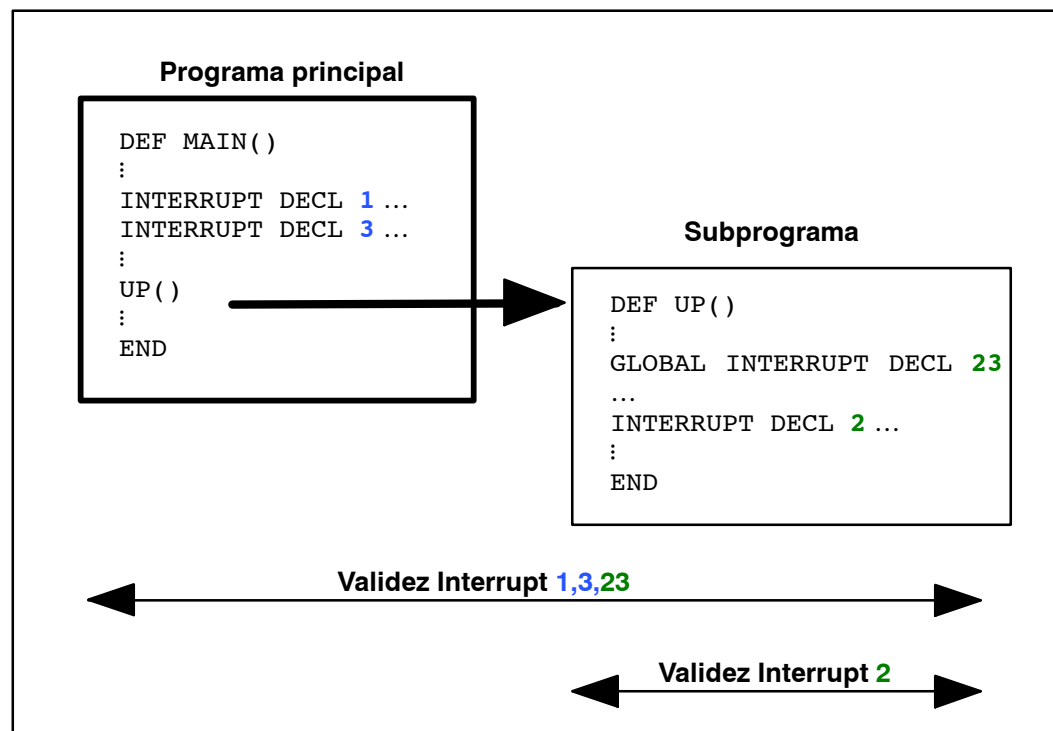


Fig. 38 Rango de validez de una interrupción dependiente del lugar y del tipo de declaración

8.2 Activación de interrupciones

Activar interrupciones Después de la declaración, en primer momento, la interrupción está desactivada. Con la instrucción

```
INTERRUPT ON 4
```

se activa la interrupción con la prioridad 4, con

```
INTERRUPT ON
```

se activan todas las interrupciones. Sólo activadas, se puede esperar que se produzcan las reacciones ante la presencia de una interrupción definida. Recién ahora se controla cíclicamente la presencia del evento de interrupción.

Disparo por flancos La supervisión del evento se produce mediante disparo por flancos, es decir, que una interrupción sólo será disparada cuando la condición lógica cambie su estado de FALSE a TRUE, pero no si en el momento de activarla, ya estaba la condición en TRUE.

Debido a tiempos de cálculo no se permiten más de 16 interrupciones activadas simultáneamente. Esto se ha de tener en cuenta sobre todo en la activación global de todas las interrupciones.

Desactivar interrupciones Del mismo modo que funciona la activación de cada una o de todas las interrupciones, funciona también su desactivación:

```
INTERRUPT OFF 4
```

o

```
INTERRUPT OFF
```

Bloquear / liberar Con las claves de ENABLE y DISABLE pueden liberarse o bloquearse interrupciones activadas de forma global.

La instrucción de bloqueo posibilita proteger determinadas partes del programa contra una interrupción. Una interrupción bloqueada es detectada y almacenada pero no ejecutada. Una vez que se produzca la liberación, se procesarán las interrupciones en el orden de prioridad respectivo.

```
DISABLE 4
```

o

```
DISABLE
```

A un evento memorizado ya no se reaccionará si la interrupción ha sido desactivada antes de que se produzca el disparo. Si una interrupción bloqueada se produce repetidas veces, sólo se ejecutará una vez después de su liberación.

Las condiciones previas para el disparo de las interrupciones son:

- la interrupción deberá estar declarada (INTERRUPT DECL ...)
- la interrupción deberá estar activada (INTERRUPT ON)
- la interrupción no deberá estar bloqueada
- se deberá haber producido el evento correspondiente (disparo por flancos)

Prioridad Cuando se produzcan simultáneamente interrupciones, será procesada primeramente la que tenga la prioridad más alta y a continuación las de prioridades más bajas. El nivel de prioridad 1 es el de máxima prioridad y el nivel 128 la mínima.

Cuando se detecta un evento, se memoriza la posición real actual del robot y se llama la rutina de interrupción. La interrupción originada así como todas las prioridades de niveles más bajos, se bloquearán durante el tiempo que dure el procesamiento. Al regresar del programa de interrupción, se cancelará el bloqueo implícito indicado incluso para la interrupción actual.

Es decir que en el momento que se produzca nuevamente (incluso durante el programa de interrupción), puede procesarse otra vez la interrupción. Sin embargo, si se desea evitar esto, deberá bloquearse o desactivarse la interrupción explícitamente antes de su salto atrás.

Un interrupt puede ser interrumpido después del primer comando en el programa de interrupciones mediante un interrupt de nivel superior. En el primer comando, el programador tiene la opción de evitar esto bloqueando/desactivando uno o todos los interrupts. Si una interrupción se desactiva automáticamente dentro del subprograma de interrupción, éste se procesará hasta el final.

Después de finalizar una interrupción de prioridad más alta, se continuará el programa de interrupción interrumpido en el mismo lugar en el cual fue interrumpido.



- A un programa de interrupción pueden transferirse parámetros IN.
- Si desea que un programa de interrupción local regrese un parámetro, deberá estar declarada la variable en la lista de datos del programa principal. En los programas de interrupción global deberá operarse con la lista de datos \$CONFIG.DAT.
- Las modificaciones de \$TOOL y \$BASE en el programa de interrupción, sólo son efectivas en el mismo (servicio de comando).
- En el programa de interrupciones no existe una ejecución en avance, ya que funciona en el nivel de comandos, es decir que es procesado paso por paso (\Rightarrow no se permiten las asignaciones \$ADVANCE). Con ello, no se puede realizar un posicionamiento aproximado.

Casos especiales:

- Las interrupciones sobre las variables del sistema \$ALARM_STOP y \$STOPMESS son procesadas incluso en caso de anomalía, es decir que a pesar de una parada del robot, continúan procesándose las instrucciones de interrupciones (no los movimientos).
- Durante una parada de operación, se puede reconocer cada interrupción declarada y activada. Después de un nuevo arranque, se procesarán las interrupciones dependiendo de sus prioridades (en caso de que estén liberadas) y a continuación, seguirá la ejecución normal del programa.

Con la llamada de un programa de interrupción, no se interrumpe el movimiento del robot en marcha. Mientras se procesa el programa de interrupción, se ejecutan todos los movimientos preparados dentro del programa interrumpido. Si el programa de interrupción, durante este tiempo, ha terminado de procesarse, continúa la ejecución del programa, sin parada del movimiento, es decir, sin prolongar el tiempo del procesamiento. Si por el contrario, no ha concluido la acción de interrupción, el robot se detendrá hasta que, después del salto atrás, se procese y continúe el movimiento subsiguiente.

Si en el propio programa de interrupciones existen instrucciones de movimiento, el programa de interrupción parará en la primera instrucción de movimiento hasta que termine de procesarse la ejecución en avance del programa principal.

El ejemplo indicado a continuación muestra como utilizar las instrucciones de interrupciones y de variables especiales del sistema. En este se controlan en forma permanente dos sensores (en las entradas 1 y 2) durante el movimiento lineal. En el momento que el sensor 1 detecta una pieza (pasa a nivel alto), se produce la llamada de un subprograma de interrupción en la cual se almacena la posición de la pieza, mostrándose como indicación, la salida correspondiente. El movimiento del robot no es interrumpido durante este proceso. A continuación se realiza de nuevo un desplazamiento hacia las piezas detectadas.



```

DEF  INTERRUPT ( )

;----- Sección de declaraciones -----
EXT  BAS (BAS_COMMAND :IN,REAL :IN )
DECL AXIS HOME
POS TEIL[2]
INT I

;----- Inicialización -----
BAS (#INITMOV,0 ) ;Inicialización de velocidades,
                  ;aceleraciones, $BASE, $TOOL, etc.
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 30,A6 0}
FOR I=1 TO 16
    $OUT[I]=FALSE ;resetear todas las salidas
ENDFOR
INTERRUPT DECL 10 WHEN $IN[1]==TRUE DO SAVEPOS (1 )
INTERRUPT DECL 11 WHEN $IN[2]==TRUE DO SAVEPOS (2 )

;----- Sección principal -----
PTP HOME ;Desplazamiento COI

PTP {X 1320,Y 100,Z 1000,A -13,B 78,C -102}

INTERRUPT ON ;activar todas las interrupciones
LIN {X 1320,Y 662,Z 1000,A -13,B 78,C -102} ;Tramo de
localización
INTERRUPT OFF 10 ;Desactivar la interrupción 10
INTERRUPT OFF 11 ;Desactivar la interrupción 11

PTP HOME

FOR I=1 TO 2
    IF $OUT[I] THEN
        LIN PARTE[I] ; Desplazarse hacia la pieza detectada
        $OUT[I]=FALSE
        PTP HOME
    ENDIF
ENDFOR

END

;----- Programa de interrupción -----
DEF SAVEPOS (NR :IN ) ;Pieza detectada
INT NR
$OUT[NR]=TRUE ;Colocar marcador
PARTE[NR]=$POS_INT ;Memorizar posición
END
    
```



Junto al paquete base (BAS.SRC) se encuentra en la unidad de control, en forma estándar, un archivo IR_STOPM(). Este subprograma ejecuta, en caso de fallo, determinadas instrucciones fundamentales. Entre otras, además de acciones específicas tecnológicas, forma parte el reposicionamiento del robot sobre su trayectoria de desplazamiento. Mientras que el robot, después de accionar el pulsador de PARADA de EMERGENCIA, continúa en su trayectoria, se produce en el disparo de los dispositivos de protección referidos directamente al operador (p. ej. puerta de protección), una parada de lado hardware fuera de la trayectoria.

Por consiguiente, Ud. debería implementar siempre en la parte de inicialización de sus programas, la secuencia siguiente (en forma estándar se encuentra siempre en la fold de BAS INI):

```

INTERRUPT DECL 3 WHEN $STOPMESS==TRUE DO IR_STOPM ( )
INTERRUPT ON 3
    
```

En el archivo `IR_STOPM()` se produce la reposición mediante la instrucción `PTP $POS_RET` estableciendo de nuevo la coincidencia de paso.

Otras variables útiles del sistema para los trabajos con interrupciones, aparecen en Tab. 26. Las posiciones hacen referencia siempre al sistema de coordenadas actual en la ejecución principal.

Específica de ejes	Cartesiana	Significado
<code>\$AXIS_INT</code>	<code>\$POS_INT</code>	Posición en la que se produce la interrupción
<code>\$AXIS_ACT</code>	<code>\$POS_ACT</code>	Posición actual REAL
<code>\$AXIS_RET</code>	<code>\$POS_RET</code>	Posición en la cual se abandono la trayectoria
<code>\$AXIS_BACK</code>	<code>\$POS_BACK</code>	Posición del punto inicial de la trayectoria
<code>\$AXIS_FOR</code>	<code>\$POS_FOR</code>	Posición del punto final de la trayectoria

Tab. 26 Variables de sistema útiles en el tratamiento de interrupciones

Las posiciones `..._BACK` y `..._FOR` dependen en el movimiento de aproximación donde se encuentre la ejecución principal. Véase al respecto Fig. 39 hasta Fig. 40.

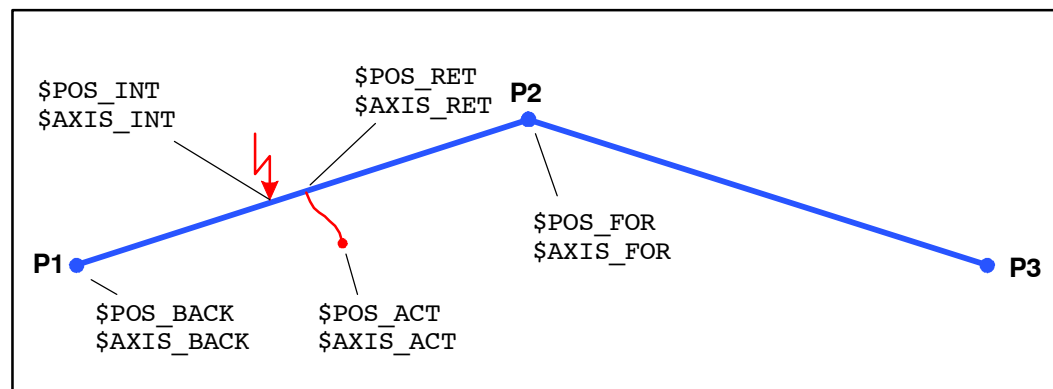


Fig. 39 Variables del sistema – interrupción con puntos de parada exactos

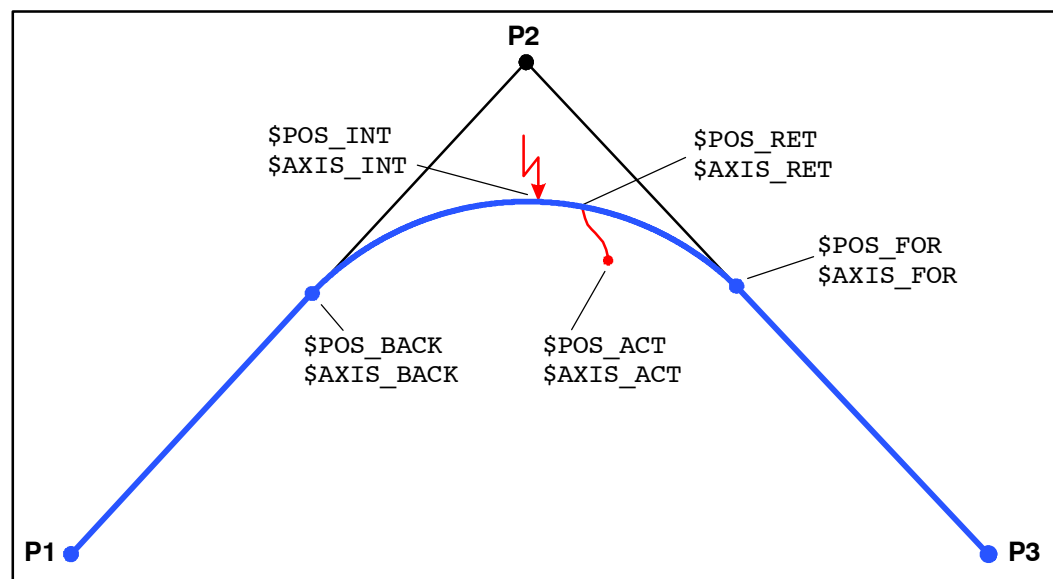


Fig. 40 Variables del sistema – interrupción en una zona de aproximación

8.3 Detención de movimientos en ejecución

BRAKE

Si se desea que se detengan movimientos en ejecución del robot cuando se produzca una interrupción, entonces se sirve uno de la instrucción **BRAKE** en el programa de interrupción. Si se programa

BRAKE

sin parámetros, esto produce un frenado del movimiento con los valores de aceleración de ejes y de trayectoria programados. El comportamiento es el mismo que el que se produce cuando se acciona la tecla **STOP**. En esta actuación, no se abandona la trayectoria de movimiento programada.

Con la instrucción

BRAKE F

(brake fast) alcanzará carreras de frenado más cortas. El robot es frenado aquí con un retardo con la mayor exactitud de trayectoria posible.

La instrucción **BRAKE** sólo puede estar en un programa de interrupción. En otros programas producirá una parada con fallo.

La instrucción **BRAKE** no tiene por que efectuarse directamente después de su llamada, sino que puede producirse en cualquier parte del programa de interrupciones. Su efectividad depende de que si en el momento de su tratamiento, aún se está ejecutando un movimiento del programa interrumpido. Si el robot está detenido, la instrucción no tiene consecuencias. Una movimiento en ejecución del programa interrumpido, se parará con el modo de frenado programado. Sin embargo, la instrucción **BRAKE** no sustituye la instrucción **HALT**, si se ha de parar la secuencia del programa. La ejecución del programa de interrupción continuará después de la parada correcta del robot, con la instrucción siguiente.



¡Después del salto atrás al programa interrumpido, se continuará en el programa con el movimiento interrumpido por **BRAKE o **BRAKE F**!**

8.4 Cancelación de rutinas de interrupción

En el ejemplo 8.1 se detectan durante el movimiento del robot, como máximo, 2 objetos mediante 2 detectores de aproximación, registrándose también sus posiciones para el arranque posterior.

El trayecto de búsqueda es ejecutado en su totalidad, incluso cuando estén ya detectados ambos objetos. Para ahorrar tiempo, es deseable interrumpir el movimiento de inmediato cuando se hayan detectado la máxima cantidad de piezas.

RESUME

La cancelación de un movimiento del robot es posible efectuarla en la KR C... con la instrucción

RESUME

La instrucción RESUME cancela todos los programas de interrupción en procesamiento y todos los subprogramas en marcha hasta el nivel en la cual se encuentra declarada la interrupción actual.

Igual a lo que sucede con la instrucción BRAKE, sólo se permite la instrucción RESUME dentro de un programa de interrupción.

En el momento de la instrucción RESUME, el puntero de avance no debe estar en el mismo nivel en el cual se declaró la interrupción, sino como mínimo, en un nivel inferior.

Al tener que interrumpirse el trayecto de búsqueda con RESUME, deberá programarse el movimiento de búsqueda en un subprograma. Esto se realiza en el ejemplo que se muestra a continuación en MOVEP (), el subprograma de interrupción se llama IR_PROG ().

Lo importante en los subprogramas que se han de interrumpir con RESUME, es que se interrumpa el procesamiento en avance antes de la última línea. Sólo así estará garantizado de que el puntero de avance de ejecución no esté con RESUME en el mismo nivel donde fue declarada la interrupción. En MOVEP () se realizó esto con la asignación \$ADVANCE=0.

En el propio programa de interrupción – en el momento que se detecten 4 piezas por el sensor en la entrada 1 – se parará el movimiento de búsqueda mediante BRAKE y a continuación se cancelará con la instrucción RESUME (ya que junto al IR_PROG () también finaliza MOVEP ()). Sin la instrucción BRAKE se trataría primero el movimiento de búsqueda en el procesamiento en avance.

Después de RESUME, continúa el programa principal con la instrucción siguiente a continuación de la llamada del subprograma, es decir \$ADVANCE=3 (resetear procesamiento en avance).



```

DEF SEARCH ( )
;----- Sección de declaraciones -----
EXT BAS (BAS_COMMAND :IN,REAL :IN )
DECL AXIS HOME
;----- Inicialización -----
INTERRUPT DECL 3 WHEN $STOPMESS==TRUE DO IR_STOPM ( )
INTERRUPT ON 3 ;tratamiento de errores estándar
BAS (#INITMOV,0 ) ;Inicialización de velocidades,
;aceleraciones, $BASE, $TOOL, etc.
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 30,A6 0}
INTERRUPT DECL 11 WHEN $IN[1] DO IR_PROG ( )
I[1]=0 ;colocar el contador predefinido a 0
;----- Sección principal -----
PTP HOME ;desplazamiento COI
INTERRUPT ON 11
MOVEP ( ) ;recorrido del trayecto de búsqueda
$ADVANCE=3 ;resetear avance
INTERRUPT OFF 11
GRIP ( )
PTP HOME
END
;----- Subprograma -----
DEF MOVEP ( ) ;Subprograma para el recorrido del trayecto de
búsqueda
PTP {X 1232,Y -263,Z 1000,A 0,B 67,C -90}
LIN {X 1232,Y 608,Z 1000,A 0,B 67,C -90}
$ADVANCE=0 ;Parar el avance
END ;
;----- Programa de interrupción -----
DEF IR_PROG ( ) ;Guardar posición de piezas
;INTERRUPT OFF 11
I[1]=I[1]+1
POSICIÓN[I]=$POS_INT ;Guardado de la posición
IF I[1]==4 THEN ;4 piezas son detectadas
BRAKE ;Detención del movimiento
RESUME ;Cancelación de IR_PROG & MOVE
ENDIF
;INTERRUPT ON 11
END
;----- Subprograma -----|

DEF GRIP ( ) ;Coger las piezas detectadas
INT POS_NR ;Variable de conteo
FOR POS_NR=I[1] TO 1 STEP -1
POSICIÓN[POS_NR].Z=POSICIÓN[POS_NR].Z+200
LIN POSICIÓN[POS_NR] ;Pasar 200mm sobre la pieza
LIN_REL {Z -200} ;Aproximarse perpendicularmente a la pieza
; Coger pieza
LIN POSICIÓN[POS_NR] ;Subir nuevamente hacia arriba
LIN {X 634,Y 1085,Z 1147,A 49,B 67,C -90}
; Depositar pieza
ENDFOR
END

```



Si existe el riesgo de que una interrupción dispare, por error, dos veces debido a la sensibilidad de un sensor (“rebotes de teclas”), puede evitarlo desactivando la interrupción en la primera línea del programa de interrupción. Sin embargo se ha de tener presente que aquí tampoco se detectará, en ningún momento, una interrupción real durante el procesamiento de interrupción. Antes de realizar el salto hacia atrás, si tiene que continuar estando activa la interrupción, se ha de conectar de nuevo la misma.



Si se ha interrumpido un movimiento como en el ejemplo anterior con `RESUME`, el movimiento siguiente no debe ser un movimiento `CIRC`, ya que el punto de inicio es distinto cada vez (\Rightarrow círculos diferentes).

En la acción de búsqueda programada en el ejemplo 8.2, las entradas son interrogadas en el ciclo de interpolación (actualmente 12 ms). Aquí se produce una inexactitud máxima de 12 ms multiplicada la velocidad de trayectoria.

“Medición
rápida”

Si se desea evitar inexactitudes, no se deberá conectar el detector de aproximación a las entradas de usuario, sino a través de entradas especiales (4 unidades) en el conector de periféricos X11. Estas entradas pueden ser activadas a través de las variables del sistema `$MEAS_PULSE[1]...MEAS_PULSE[4]` (tiempo de respuesta 125 μ s).

Al conectar la interrupción, el impulso de medición no debe estar presente, caso contrario se emite el correspondiente mensaje de fallo.

8.5 Utilización de banderas (flags) cíclicas

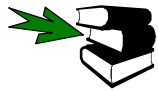
En la instrucción para declaraciones de interrupciones no se permiten combinaciones lógicas.

Para definir eventos complejos deberá trabajarse con banderas (flags) cíclicas ya que sólo estas permiten una actualización continua.

Con la secuencia

```
:  
$CYCFLAG[3] = $IN[1] AND ([ $IN[2] OR $IN[3] )  
INTERRUPT DECL 10 WHEN $CYCFLAG[3] DO IR_PROG()  
INTERRUPT ON 10  
:
```

pueden supervisarse 3 entradas simultáneamente y combinarlas entre sí en forma lógica.



Informaciones adicionales se encuentran en el capítulo **[Variables y declaraciones]**, apartado **[Variables y archivos del sistema]**.

9 Trigger – Acciones de conmutación referentes a la trayectoria

Al contrario de las funcionalidades de interrupción independientes del movimiento, algunos casos de aplicación requieren determinados acciones de conmutación que se disparan dependientes de la trayectoria del movimiento. Estos casos de aplicación son p. ej:

- cerrar o abrir una pinza de soldadura durante la soldadura por puntos
- conectar/desconectar la corriente de soldadura en la soldadura al arco sobre una trayectoria
- conectar/desconectar el caudal durante el proceso de pegado y aplicación de sellantes

En la KR C..., estas acciones de conmutación referentes a la trayectoria se pueden realizar mediante instrucciones de disparo (TRIGGER). Paralelamente al próximo movimiento de robot, se puede procesar con TRIGGER, en dependencia de la trayectoria, un subprograma o efectuar una asignación de valor a una variable o una instrucción PULSE o activarse una salida.

9.1 Acciones de conmutación en el punto de inicio o de destino de la trayectoria

TRIGGER

Si se desea una acción de conmutación referente a un punto de inicio o de destino de una trayectoria de movimiento, deberá programar delante de la instrucción de movimiento en cuestión (PTP, LIN o CIRC) una instrucción de TRIGGER con la sintaxis siguiente:

```
TRIGGER WHEN DISTANCE=Punto de conmutación DELAY=Tiempo  
DO Instrucción <PRIO=Prioridad>
```

En la siguiente tabla se describen los argumentos en forma más detallada.

Argumento	Tipo de datos	Significado
Punto de conmutación	INT	En los pasos individuales la DISTANCE=0 define el punto de inicio y DISTANCE=1 el punto final del siguiente movimiento. En los pasos de aproximación la DISTANCE=1 marca las indicaciones del centro de la curvatura de aproximación siguiente. Si el paso precedente es un paso con posicionamiento aproximado, DISTANCE=0 marca el punto final de la curvatura de aproximación precedente.
Tiempo	INT	Con la indicación de DELAY se puede retardar el punto de conmutación en un determinado tiempo. El punto de conmutación sólo podrá desplazarse de modo que quede comprendido dentro del paso de referencia. La unidad se expresa en milisegundos .
Instrucción		La instrucción puede ser <ul style="list-style-type: none"> • una llamada de un subprograma • una asignación de valor a una variable • una instrucción OUTPUT (también Pulse).
Prioridad	INT	Cada instrucción TRIGGER con llamada de subprograma deberá ir asignada a una prioridad. Son permisibles valores entre 1...39 y 81...128. En este caso se trata de las mismas prioridades que en las interrupciones (véase capítulo 8). Los valores 40...80 están reservados por el sistema para una asignación automática de prioridades. Programe para ello PRIO=-1.

Tab. 27 Argumentos en la instrucción TRIGGER

Con la secuencia de instrucciones

```

:
LIN PUNTO2
:
:
TRIGGER WHEN DISTANCE = 0 DELAY=20 DO $OUT[4]=TRUE
TRIGGER WHEN DISTANCE = 1 DELAY=-25 DO UP1() PRIO=-1
LIN PUNTO3
:
:
LIN PUNTO4
:

```

se activa durante el movimiento lineal hacia el PUNTO3 la salida 4 después de 20 milisegundos tras el inicio del movimiento y se produce la llamada del subprograma UP1(), 25 milisegundos antes de alcanzar el punto final. La asignación de prioridad se produce automáticamente por el sistema.

Para entender mejor los diferentes efectos de la indicación de DISTANCE en pasos individuales y de aproximación, véase Fig. 41 – Fig. 44.

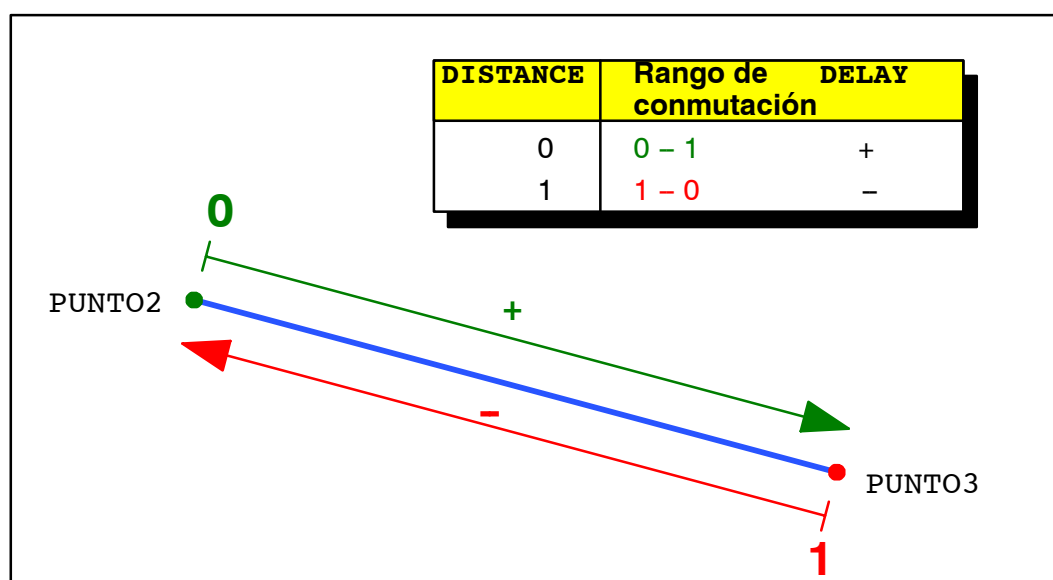


Fig. 41 Rangos de conmutación y posibles valores de retardo cuando el punto de inicio y de destino son puntos de parada exactos

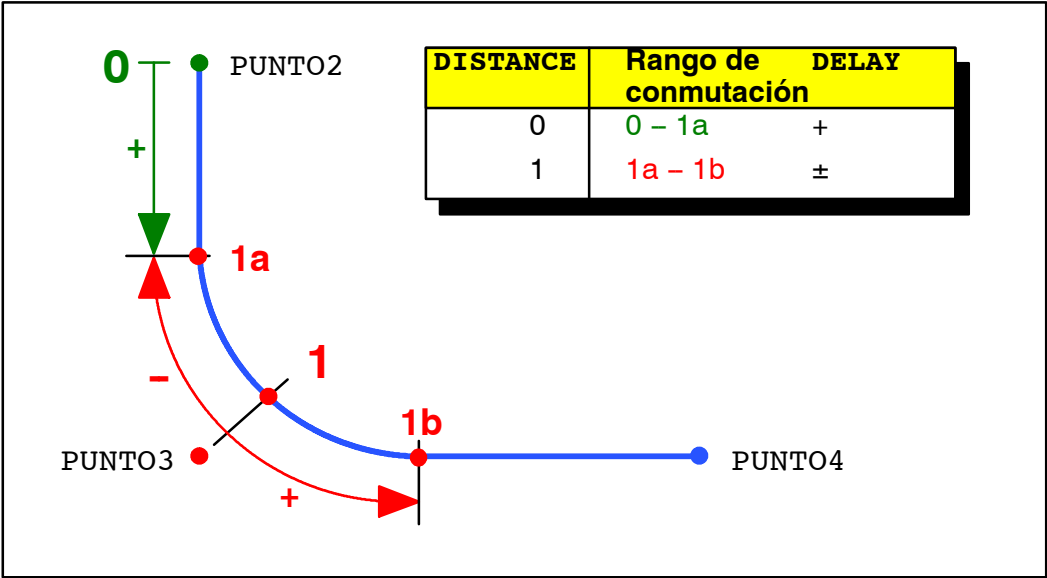


Fig. 42 Rangos de conmutación y posibles valores de retardo cuando el punto de inicio es un punto de parada exacta y el punto final uno de aproximación

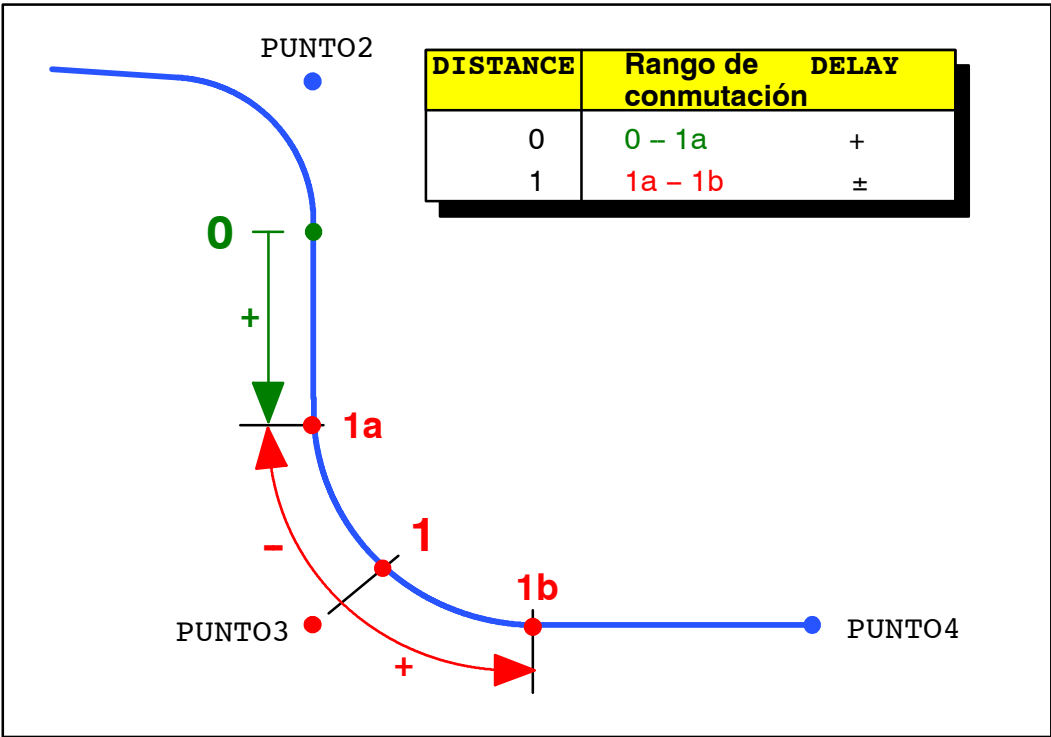


Fig. 43 Rangos de conmutación y posibles valores de retardo cuando los puntos de inicio y de destino son puntos de aproximación

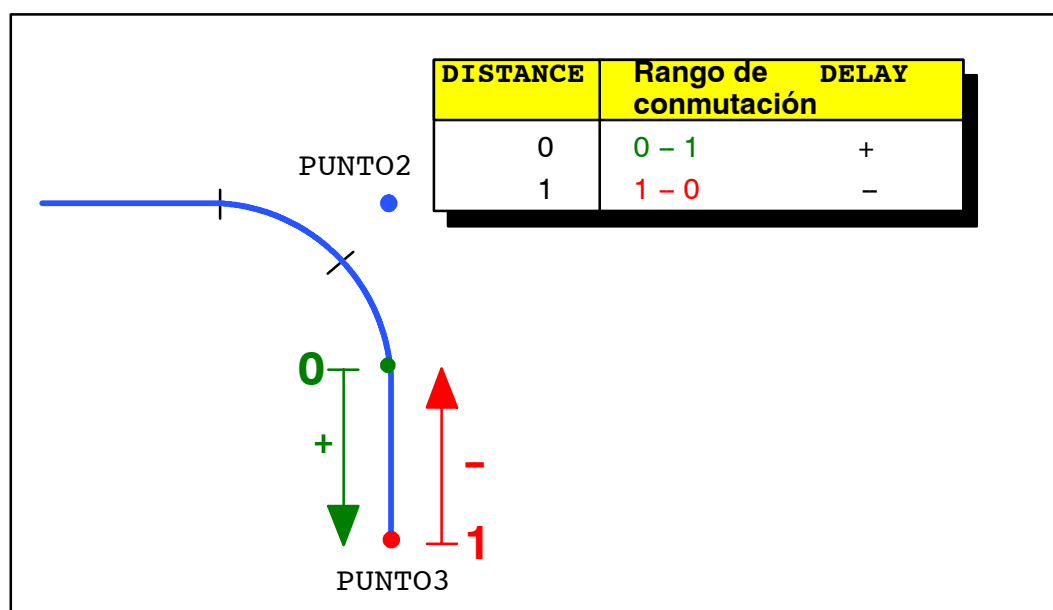


Fig. 44 Rangos de conmutación y posibles valores de retardo cuando el punto de inicio es un punto de aproximación y el punto final un punto de parada exacta

9.2 Acciones de conmutación en cualquier lugar sobre la trayectoria

Con la instrucción TRIGGER referente a la trayectoria, se puede disparar las acciones de conmutación en cualquier lugar de la trayectoria con la indicación de la distancia. Adicionalmente, el disparo, a su vez, puede desplazarse temporalmente, como en el caso de las acciones de conmutación para el inicio o el punto final.

La acción de conmutación referente a la trayectoria sólo es permisible para los movimientos de trayectoria (LIN o CIRC). La instrucción TRIGGER hace referencia para ello al paso de movimiento programado a continuación, y su sintaxis es la siguiente:

```
TRIGGER WHEN PATH = Trayecto DELAY = Tiempo DO Instrucción
<PRIO=Prioridad
```

En la siguiente tabla se describen los argumentos en forma más detallada.

Argumento	Tipo de datos	Significado
Trayecto	INT	<p>Con Trayecto indica la distancia deseada anterior al punto final donde se produce el disparo.</p> <p>Si este punto final es un punto con posicionamiento aproximado, Trayecto indica la distancia deseada entre la acción de conmutación y la posición de la zona de aproximación más próxima al punto final.</p> <p>El punto de conmutación puede adelantarse mediante un Trayecto negativo hasta el punto de inicio. Si el punto de inicio es un punto de posicionamiento aproximado, se puede desplazar el punto de conmutación hasta el principio de la zona de aproximación.</p> <p>Con una indicación positiva de Trayecto, se puede efectuar un desplazamiento hacia el punto de parada exacta después del punto de disparo.</p> <p>La unidad se expresa en milímetros.</p>
Tiempo	INT	<p>Con la indicación de DELAY es posible de retardar (+) o de acortar (–) en un tiempo determinado el punto de conmutación en forma relativa respecto a la indicación de PATH.</p> <p>Pero el punto de conmutación solamente puede ser desplazado en la zona de conmutación arriba mencionada (tanto, que puede alcanzar el próximo punto de parada exacta). En movimientos de posicionamiento aproximado, el punto de conmutación se puede adelantar como máximo, hasta el comienzo de aproximación del punto de inicio.</p> <p>La unidad se expresa en milisegundos.</p>
Instrucción		<p>La instrucción puede ser</p> <ul style="list-style-type: none"> • una llamada de un subprograma • una asignación de valor a una variable • una instrucción OUTPUT (también Pulse).
Prioridad	INT	<p>Cada instrucción TRIGGER con llamada de subprograma deberá ir asignada a una prioridad. Son permisibles valores entre 1...39 y 81...128. En este caso se trata de las mismas prioridades que en las interrupciones (véase capítulo 8).</p> <p>Los valores 40...80 están reservados por el sistema para una asignación automática de prioridades. Programe para ello PRIO=–1.</p>

Tab. 28 Argumentos en la instrucción TRIGGER

Secuencia de instrucción:

```

:
LIN PUNTO2 C_DIS
TRIGGER WHEN PATH = Y DELAY= X DO $OUT[2]=TRUE
LIN PUNTO3 C_DIS
LIN PUNTO4 C_DIS
LIN PUNTO5
:

```

Al poderse desplazar el punto de conmutación a partir del punto de movimiento del cual fue programado mas allá de todos los puntos de posicionamiento aproximado siguientes hasta el próximo punto de parada exacta, es posible un desplazamiento del punto de conmutación desde la zona de aproximación PUNTO2 hasta PUNTO5. Si en esta secuencia de instrucciones el PUNTO2 no es de posicionamiento aproximado, entonces el punto de conmutación solo podrá ser desplazado hasta el punto de parada exacta PUNTO2.

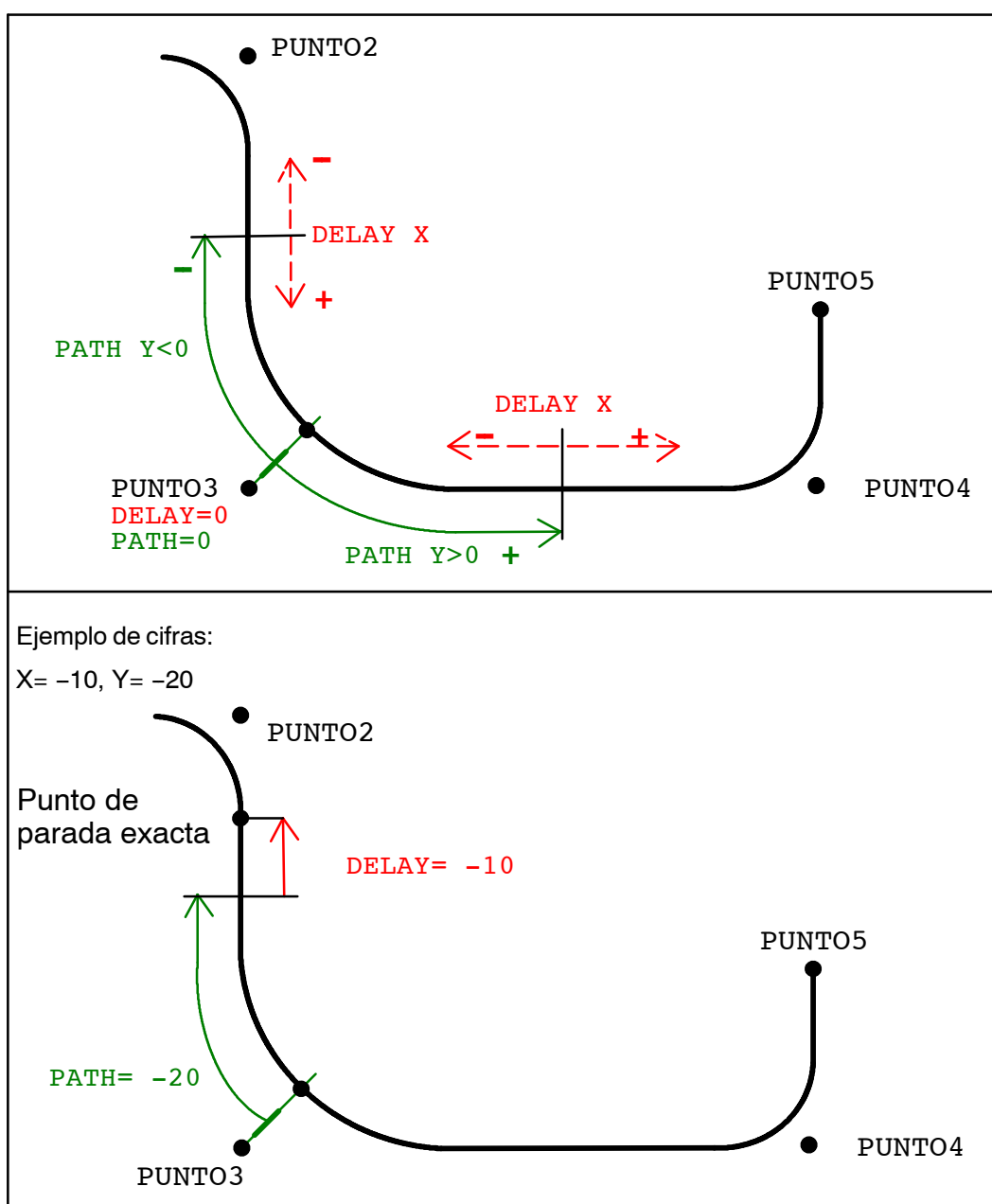


Fig. 45 Zona de conmutación cuando el punto de inicio es un punto de posicionamiento aproximado



Casos especiales:

▪ Desplazamiento COI

Si se realiza una selección de paso en un movimiento de trayectoria, se produce un desplazamiento de coincidencia de paso COI. Al poder ser el punto inicial del desplazamiento COI uno cualquiera, no puede ser un punto inicial recomendable para una indicación de distancia. Si delante de este tipo de movimiento se encuentran programadas instrucciones **Trigger** con indicación de **PATH**, y se produce una selección de paso sobre estas instrucciones, se ejecutarán todas en el punto final.

▪ No se puede realizar un posicionamiento aproximado

Si no se puede realizar una aproximación, se produce en este punto un movimiento con parada exacta. Pero éste básicamente es tratado bajo el aspecto como si se fuese un movimiento de aproximación. Las acciones de conmutación que deban producirse sobre la trayectoria, permanecen almacenadas y se activarán en los puntos correspondientes. Sin embargo, por regla general, ya no serán muy exactas ya que resulta una trayectoria distinta y por consiguiente una longitud de trayectoria también distinta. Las acciones de conmutación colocadas en la primera parte de la zona de aproximación, debido a un valor negativo del **PATH**, se pueden disparar, como muy pronto, en el punto de aproximación:

```
:
LIN P1 C_DIS
TRIGGER WHEN PATH=-120 DELAY=0 DO UP1( ) PRIO=-1
TRIGGER WHEN PATH=-70 DELAY=0 DO $OUT[2]=TRUE
LIN P2 C_DIS
:
```

En el ejemplo anterior ahora la distancia entre el punto de inicio y de destino debe ser 100 mm. Si se puede realizar un posicionamiento aproximado en P1, se ejecutará la llamada del subprograma **UP1()** 20 mm antes de alcanzar el punto de trayectoria más cercano al punto de aproximación P1. La activación de la salida 2 se efectúa unos 30 mm después de este punto de la trayectoria. Si no se pudo realizar el movimiento de posicionamiento aproximado en P1, la trayectoria pasa a través del punto P1, en el cual también es posicionado. Inmediatamente al salir de P1 se ejecuta la llamada del subprograma **UP1()**, se activa la salida 2 a una distancia de 30 mm de P1.

▪ Cancelación de un movimiento

Si un movimiento es interrumpido, p. ej. mediante una selección de paso o por un reset, y no lleva a término, las acciones de conmutación que todavía no han sido ejecutadas, ya no serán ejecutadas y se borrarán (como nel caso de la indicación **DISTANCE**).

▪ Instrucción **TRIGGER** – referente a un trayecto para un movimiento **PTP**

Si una instrucción **PATH-TRIGGER** con indicación de trayecto es programada para que sea un movimiento **PTP**, el interpretador la rechazará durante la ejecución.

▪ Posicionamiento aproximado **PTP** – trayectoria

Si una instrucción **PATH-TRIGGER** es programada para un movimiento cuyo punto de inicio es un punto de posicionamiento aproximado **PTP** – trayectoria, entonces, dado que la zona de aproximación completo es movimiento **PTP**, las acciones de conmutación se realizarán, como muy pronto, al final de la zona de aproximación.

En una zona de aproximación trayectoria – **PTP**, todas las instrucciones **TRIGGER** aún activas sin conmutar, se disparan en el punto de inicio de la zona de aproximación. Ya que a partir de este punto se produce un recorrido **PTP**, sin opción a asignarle una trayectoria.

En el ejemplo presentado a continuación se han programado tanto acciones de conmutación con indicación de **DISTANCE** como indicación de **PATH**. Cada uno de los puntos de conmutación y la trayectoria de movimiento están representados en Fig. 46.



```

DEF TRIG ( )
;----- Sección de declaración -----
EXT BAS (BAS_COMMAND :IN,REAL :IN)
DECL AXIS HOME
INT I
SIGNAL ADHESIVO $OUT[3]
;----- Inicialización -----
INTERRUPT DECL 3 WHEN $STOPMESS==TRUE DO IR_STOPM ( )
INTERRUPT ON 3
BAS (#INITMOV,0 ) ;Inicialización de velocidades,
                    ;aceleraciones, $BASE, $TOOL, etc.
$APO.CDIS=35      ;prefijar la distancia de aproximación
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 30,A6 0}
POS0={POS: X 1564,Y -114,Z 713,A 128,B 85,C 22,S 6,T 50}
POS1={X 1383,Y -14,Z 713,A 128,B 85,C 22}
POS2={X 1383,Y 200,Z 713,A 128,B 85,C 22}
POS3={X 1527,Y 200,Z 713,A 128,B 85,C 22}
POS4={X 1527,Y 352,Z 713,A 128,B 85,C 22}
FOR I=1 TO 16
    $OUT[I]=FALSE
ENDFOR
;----- Sección principal-----
PTP HOME ;desplazamiento COI
PTP POS0
TRIGGER WHEN DISTANCE=0 DELAY=40 DO $OUT[1]=TRUE
TRIGGER WHEN PATH=-30 DELAY=0 DO UP1(2) PRIO=-1
LIN POS1
TRIGGER WHEN DISTANCE=1 DELAY=-50 DO PEGAMENTO=TRUE
TRIGGER WHEN PATH=180 DELAY 55 DO PULSE($OUT[4],TRUE,0.9)
TRIGGER WHEN PATH=0 DELAY=40 DO $OUT[6]=TRUE
LIN POS2 C_DIS
TRIGGER WHEN DISTANCE=0 DELAY=40 DO PULSE ($OUT[5],TRUE,1.4 )
TRIGGER WHEN PATH=-20 DELAY=-15 DO $OUT[8]
LIN POS3 C_DIS
TRIGGER WHEN DISTANCE=1 DELAY=-15 DO UP1 (7 ) PRIO= -1
LIN POS4
PTP HOME
END
DEF UP1 (NR :IN )

INT NR
IF $IN[1]==TRUE THEN
    $OUT[NR]=TRUE
ENDIF
END
    
```

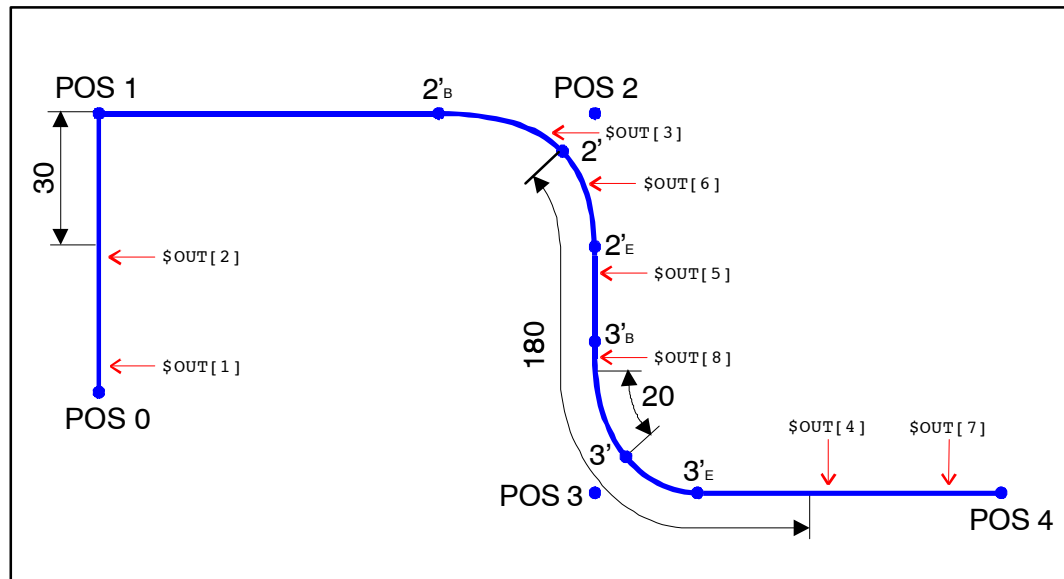


Fig. 46 Puntos de conmutación y trayectoria de movimiento pertenecientes al ejemplo anterior

9.3 Sugerencias & trucos

9.3.1 Interacción de instrucciones de disparo (Trigger)

El programa representado, si bien está en condiciones de funcionar, puede presentar problemas debido a instrucciones de disparo de igual prioridad.



```
;FOLD PTP P1
TRIGGER WHEN DISTANCE=0 DELAY=0 DO SUB1() PRIO=15
PTP P1
...
;ENDFOLD
...
;FOLD PTP P2
TRIGGER WHEN DISTANCE=0 DELAY=-75 DO SUB1() PRIO=15
...
;ENDFOLD
```

Si la primera instrucción de disparo todavía no ha finalizado, mientras la segunda instrucción de disparo ya se activa, se emite un fallo de tiempos. Esto puede ocurrir, por ejemplo, si ambos puntos se encuentran muy próximos.



Como solución al problema, se ofrecen dos posibilidades:

- Asignarle a ambas instrucciones de disparo, de forma manual, distintas prioridades;
- Asignarle a ambas instrucciones de disparo, la prioridad “-1”, para que el sistema determine automaticamente la prioridad correcta.

10 Listas de datos

10.1 Listas de datos locales

Las listas de datos sirven para poner a disposición declaraciones específicas de programa o de nivel superior. De estas forman parte también las informaciones de punto tales como p. ej. coordenadas:

- Para cada archivo SRC se puede crear una lista de datos. Este llevará el mismo nombre que el archivo SRC y termina con la extensión “.DAT”.
- La lista de archivo es local a pesar de tratarse de un archivo propio.
- En una lista de datos **sólo** pueden encontrarse declaraciones e inicializaciones.
- Se puede declarar en una línea e inicializarse.
- No se acepta ninguna variable del sistema.

DEFDAT

La declaración de listas de datos se realiza de forma análoga a la declaración de archivos SRC. Esta declaración es canalizada mediante la clave **DEFDAT** y el nombre del programa y concluye con la clave **ENDDAT**.

La inicialización de variables se realiza mediante la asignación de valor en las variables respectivas, directamente en la línea de declaración.

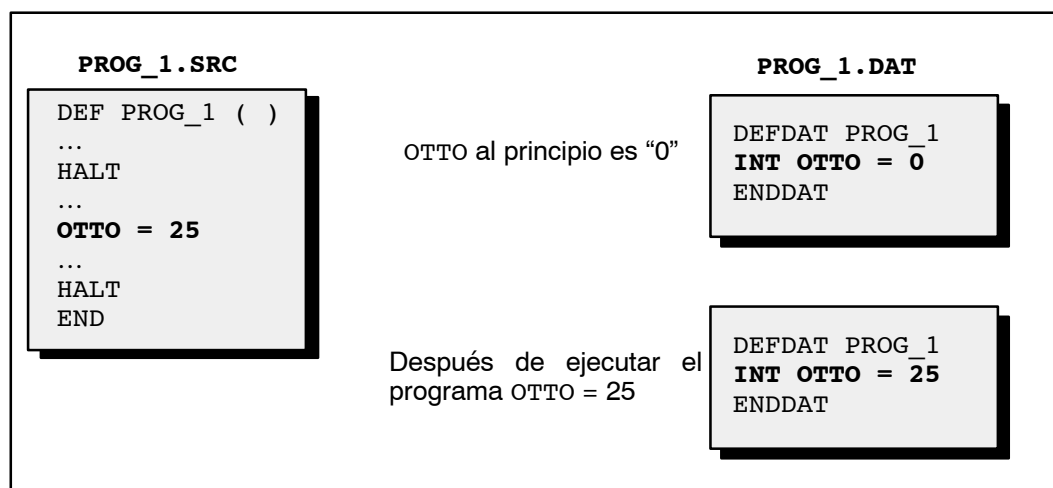


Fig. 47 Inicialización y asignación de valor de las variables declaradas en listas de datos

A través de la declaración e inicialización en la lista de datos, se prescinde de ello en el programa principal. Si se la asigna a la variable `OTTO` en el programa principal un nuevo valor, este también será registrado en la lista de datos, permaneciendo en este de forma memorizada permanente (véase Fig. 47).

Después de efectuar un ON/OFF de la unidad de control se trabajará con el “nuevo” valor. Esto es imprescindible para una corrección online u otras correcciones de programa.

Si un programa principal ha de iniciarse siempre con el mismo valor, deberá declararse la variable correspondiente en el programa principal con el valor deseado.

En las listas de datos deben encontrarse las siguientes declaraciones:

- Declaraciones externas para subprogramas y funciones que pueden ser utilizadas en el archivo SRC.
- Declaraciones de importación para variables importadas.
- Declaraciones e inicializaciones de variables utilizadas en un archivo SRC.
- Declaraciones de nombres de canal y de señal, que pueden ser utilizadas en el archivo SRC.
- Declaraciones de tipos de datos y de conteo (Struc, Enum) que pueden ser utilizados en la lista de datos o en el archivo SRC.

10.2 Listas de datos globales

Las variables definidas en una lista de datos pueden ser facilitadas para que pueda acceder un programa principal “externo”.

PUBLIC

Para ello deberá figurar y definirse en el encabezamiento de la lista de datos la clave opcional **PUBLIC**, como “accesible al público”. Ahora existen dos posibilidades de declarar la variable:

IMPORT

- Una variable se define, por ej. con `INT OTTO = 0` en la lista de datos, y en el programa principal externo debe importarse esta variable con la instrucción `Import`, para tener acceso a la misma.

A una variable importada puede asignarse en el programa principal un nombre diferente que el original que tenía en la lista de datos de donde se realizó la importación.

Es decir que si desea utilizar en un programa `PROG_2 ()` la variable `OTTO` procedente de la lista de datos anterior `PROG_1`, deberá programar junto a la palabra clave de **PUBLIC** en la lista de datos, las especificaciones de importación siguientes dentro del programa `PROG_2 ()`:

```
IMPORT INT OTTO_2 IS /R1/PROG_1 .. OTTO
```

La variable `OTTO` de la lista de datos `PROG_1.DAT` en el directorio `/R1` ahora es conocida bajo el nombre de `OTTO_2` incluso en el programa `PROG_2 ()` (véase Fig. 48).

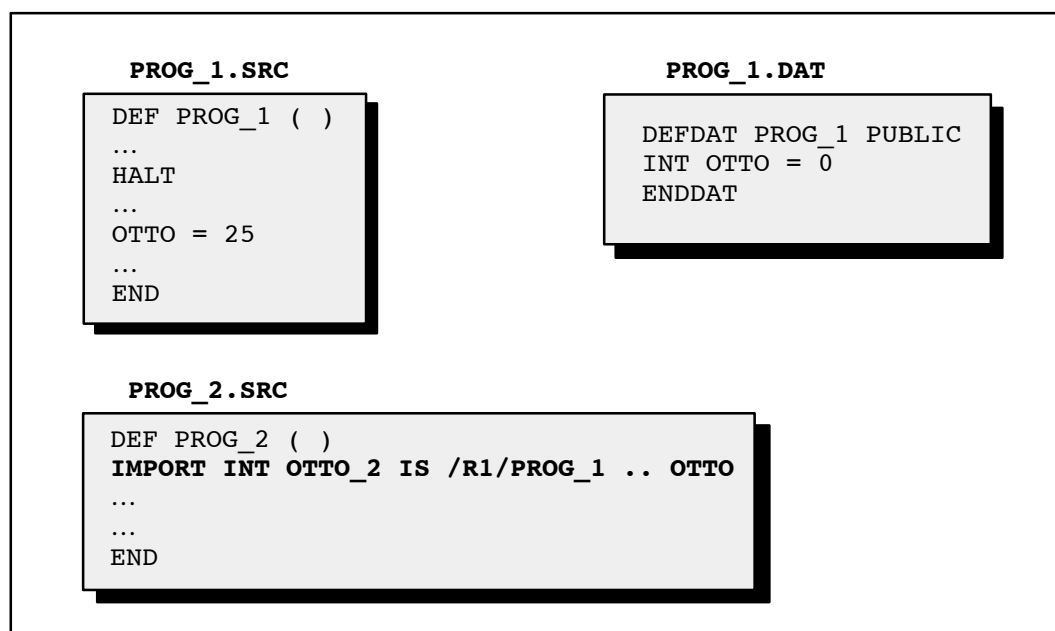


Fig. 48 Importación de variables de listas de datos “externas” con `Import`

Global

- La variable es declarada como “Variable global” por ej. `DECL GLOBAL INT OTTO = 0` y es accesible a todo otro programa principal externo sin instrucción de importación.

Cuando se ha declarado una variable global, no es posible cambiar el nombre de la variable en otro programa principal.

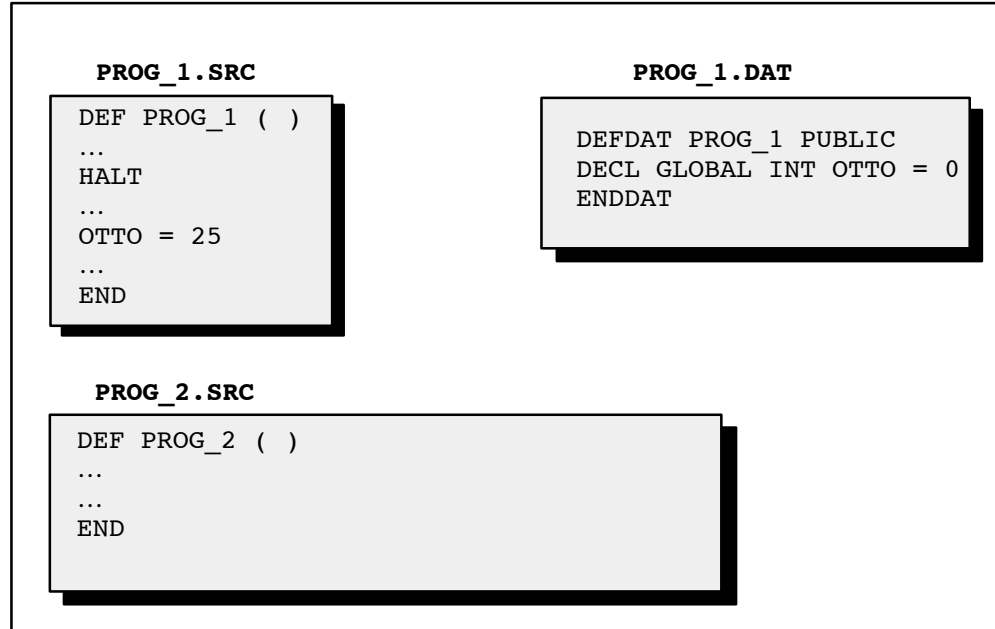
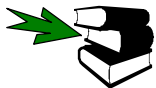


Fig. 49 Importación de variables de listas de datos “externas” sin Import



La declaración de una variable global solo está permitida en listas de datos. Si se utiliza en ficheros SRC o SUB, se genera un mensaje de fallos.

En la lista global del sistema `$CONFIG.DAT` ya predefinida, pueden definirse variables, estructuras, canales y señales, válidas por tiempo prolongado y de importancia superior para muchos programas. Las variables en el fichero `$CONFIG.DAT` no necesitan ser declaradas con `IMPORT`, dado que son conocidas automáticamente por todos los programas de aplicación.



Informaciones adicionales acerca de `$CONFIG.DAT` se encuentran en el capítulo **[Variables y declaraciones]**, apartado **[Variables y archivos del sistema]**.

11 Editor externo

Este programa adicional amplía el software del robot en funciones no disponibles en la superficie de operación.

Limpiar el programa

Puntos en una trayectoria y parámetros de movimiento no referenciados son borrados de la lista de datos.

Activar y desplazar finales de carrera

Manipulación de bloques

- Marcar y copiar, borrar o recortar un bloque.
- Invertir la trayectoria del robot en la zona marcada, es decir, que el primer punto programado del sector marcado de la trayectoria, será el último al cual se desplazará el robot, y el último punto programado pasará a ser el primero.
- Efectuar con la trayectoria del robot, una imagen especular en el plano X-Z del sector marcado, en el sistema de coordenadas universales.
- Modificar los parámetros de movimiento (velocidad, aceleración, etc.) en la zona marcada.
- Correr todos los puntos dentro de la zona de trayectoria marcada en el sistema de coordenadas de la pieza (BASE), la herramienta (TOOL) o universales (WORLD). El corrimiento o giro puede efectuarse por la entrada manual de un valor offset en el sistema de coordenadas correspondiente, o por programación por aprendizaje de puntos de referencia.
- Desplazamiento específico del eje de todos los puntos en la zona marcada.

Adaptar puntos de trayectoria

- a otra herramienta, o
- a otro sistema de coordenadas de la pieza.

Adaptar puntos de trayectoria

- mientras un programa se está ejecutando en la unidad de control, pueden efectuarse desplazamientos de coordenadas de un punto en el sistema de coordenadas Tool, Base y universales.

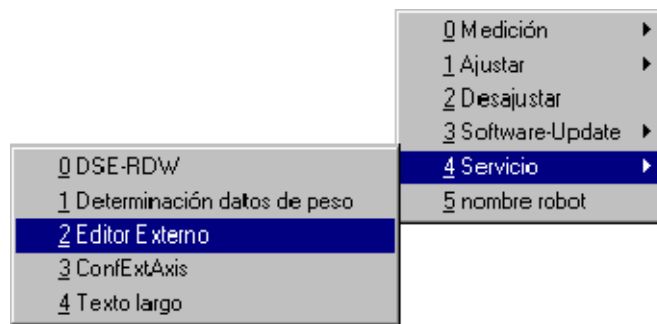
11.1 Arranque del editor externo



El editor externo recién está disponible a partir del grupo de “Expertos”.

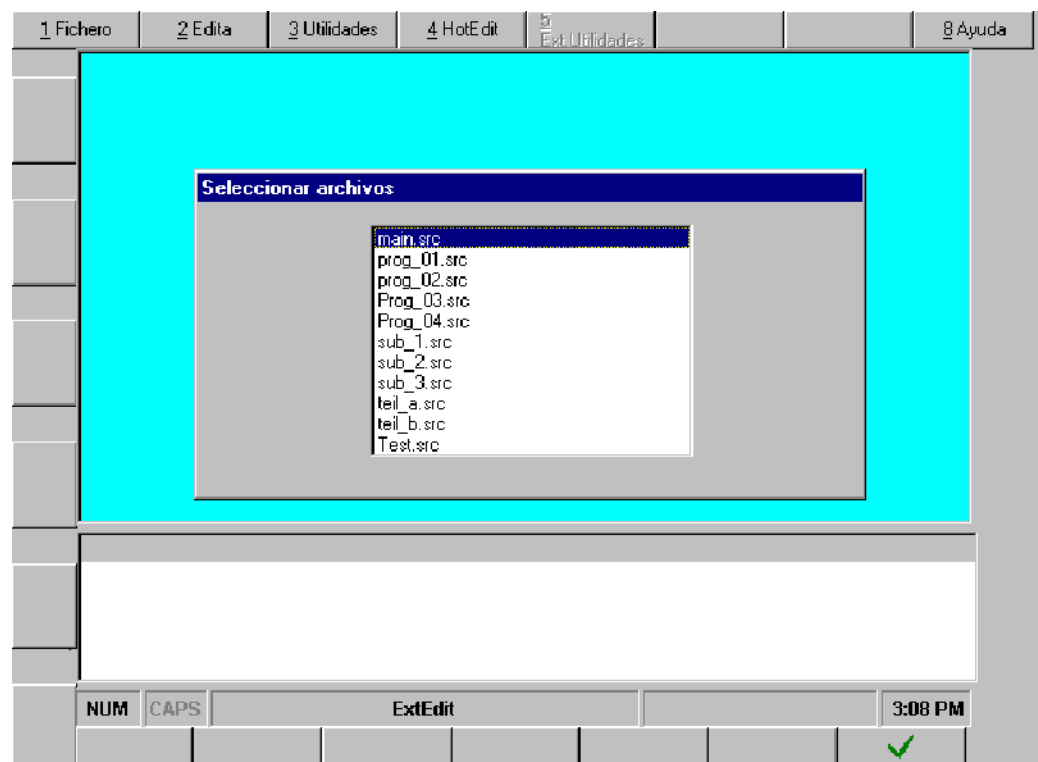
Inicialización

Llame al “Editor externo” a través del menú “Inicialización” y la opción ofrecida allí de “Servicio”.



Si el menú “Inicialización” no se encuentra disponible, entonces debe cerrar el programa seleccionado. Recuerde: seleccione para ello del menú “Procesar” la opción “Cancelar programa”.

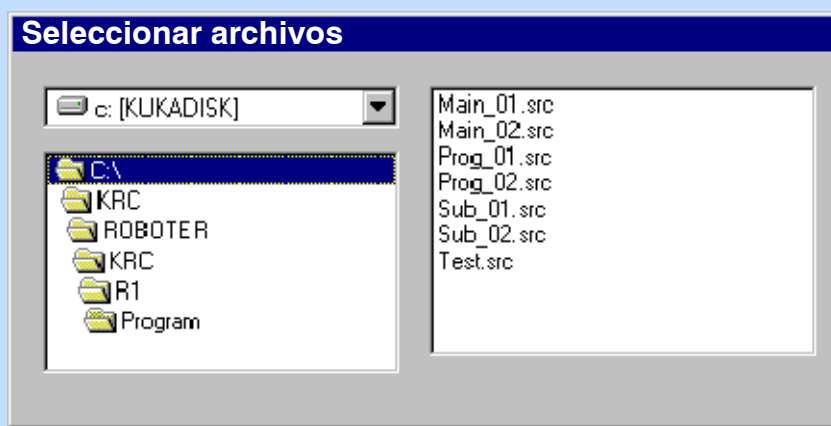
A continuación se visualiza en pantalla el editor externo.



Inmediatamente después del arranque del editor aparece el diálogo “Seleccionar archivos”. Con la tecla del cursor “↓” y “↑” seleccione el fichero a editar. Se dispone de todos los ficheros SRC del directorio R1, excepto de aquellos ficheros estándar definidos en “HotEdit.ini” (búsqueda “C:\Krc\Util”).



Si el “Editor externo” es arrancado como programa fuera de línea (offline), se dispone adicionalmente de campos de selección para la indicación de trayectos de búsqueda.

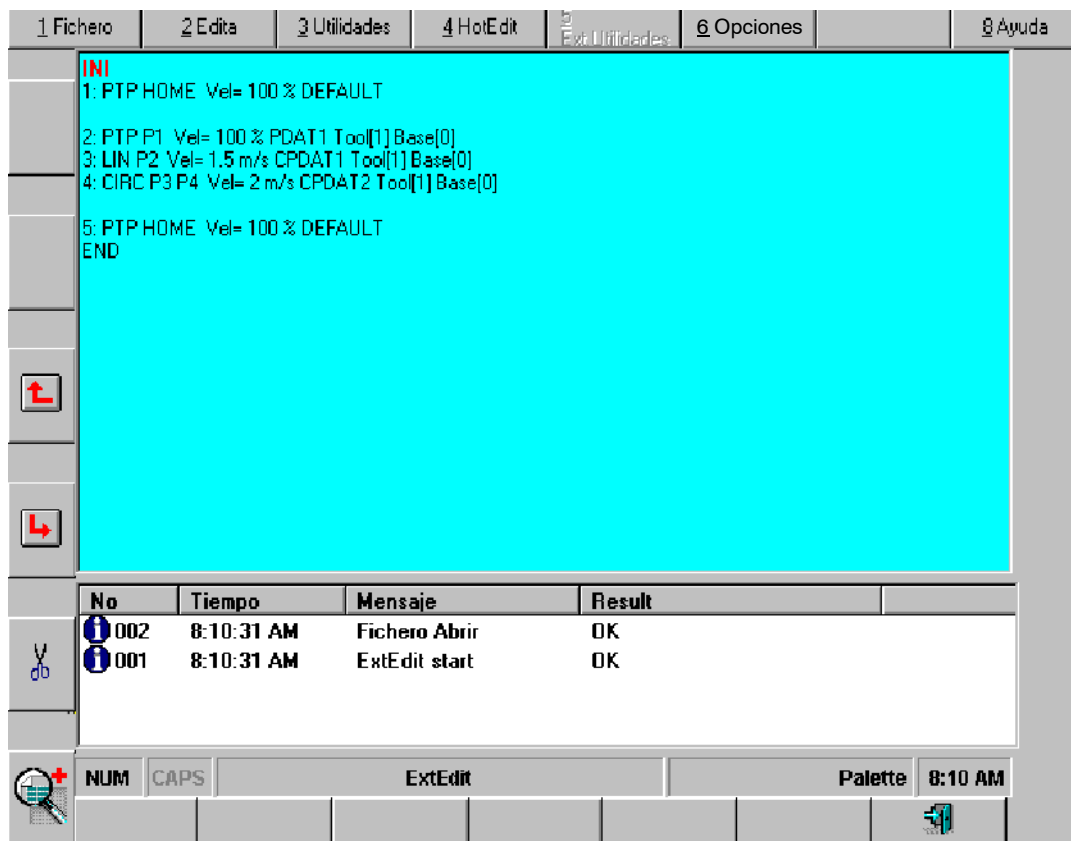


Si Ud. ha marcado el fichero deseado de esta manera, entonces pulsando la tecla del softkey correspondiente o la tecla de entrada, se lo carga en el editor. El tiempo de duración de carga depende del tamaño del fichero.



Si aparece el mensaje “*.DAT-File not found” (Archivo no encontrado), entonces Ud. ha intentado cargar un programa en el editor, el cual no posee una lista de datos. Esta función no está disponible en la presente versión del editor. Confirme este mensaje pulsando la tecla de entrada. Abra el menú “Archivo” y seleccione la opción “Abrir”, para tener acceso a la ventana “Seleccionar archivos”.

Si el fichero ha sido cargado con éxito, el programa es presentado en forma parecida a la presentación por el editor de la superficie de operación.



11.2 Operación

Los elementos de la superficie de operación normal se encuentran también en la del editor externo. Así, se tiene aquí también a disposición, funciones de estado, funciones con menús, softkeys, una ventana de programas, así como también una ventana de mensajes y una línea de estados.



La apertura de un menú puede efectuarse de forma alternativa, manteniendo pulsada la tecla “ALT” y a continuación la letra subrayada en el menú. De forma análoga, se selecciona también la instrucción que ofrece el menú.

Funciones de estado



Con las teclas de las funciones de estado “Arriba”, o bien, “Abajo”, puede Ud. mover la marca de línea (el foco) línea por línea en dirección del comienzo de la lista o al final de la misma. Las teclas del cursor “↑” y “↓” así como las teclas “PGUP” y “PGDN” cumplen la misma función.

```
INI
1: PTP HOME Vel= 100 % DEFAULT
2: PTP P1 CONT Vel= 100 % PDAT1 Tool[1] Base[0]
3: LIN P2 CONT Vel= 1.5 m/s CPDAT1 Tool[1] Base[0]
4: CIRC P3 P4 Vel= 2 m/s CPDAT2 Tool[1] Base[0]
5: PTP HOME Vel= 100 % DEFAULT
END
```

Foco


```
INI
1: PTP HOME Vel= 100 % DEFAULT
2: PTP P1 CONT Vel= 100 % PDAT1 Tool[1] Base[0]
3: LIN P2 CONT Vel= 1.5 m/s CPDAT1 Tool[1] Base[0]
4: CIRC P3 P4 Vel= 2 m/s CPDAT2 Tool[1] Base[0]
5: PTP HOME Vel= 100 % DEFAULT
END
```

Foco



Con las teclas “PGUp” y “PGDn” puede Ud. pasar páginas hacia adelante o hacia atrás. Para ello debe estar desactivada la función “NUM” o utilizar un teclado externo.

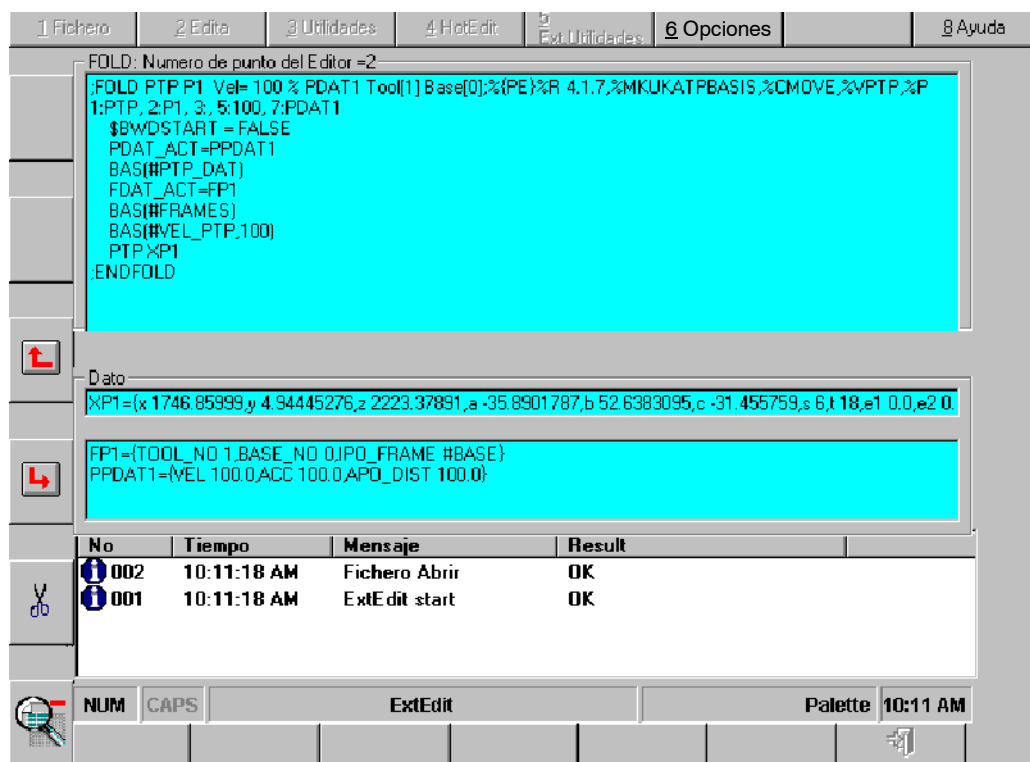


Se borra el contenido de la ventana de mensajes



Si Ud. pulsa la tecla de la función de estado “KRL”, puede visualizar en la pantalla el programa tal como se presenta en el modo del experto con los ajustes “Abrir todas Fold” y “Visibilidad Limitada no”.

En el rango “FOLD” es indicada la instrucción marcada con el foco (“fichero SRC”). En el rango “Datos” se visualizan los datos correspondientes a la instrucción marcada (“fichero DAT”).



Pulsando la tecla de la función de estado “Lupa” se retorna a la representación del programa completo.

Barra de softkeys



Este softkey responde a la instrucción del menú “Cerrar”, que cierra el programa actual, pero deja al editor seguir en procesamiento.

Ventana de mensajes

En la ventana de mensajes se emiten mensajes relevantes para el editor externo.

No	Tiempo	Mensaje	Result
002	11:13:13	Fichero Abrir	OK
001	11:13:13	ExtEdit start	OK

No.: tipo de mensaje y nro.

Tiempo: el instante, en el cual se crea el mensaje

Mensaje: el tipo de mensaje

Result: la información del resultado de la acción

Línea de estados

En la línea de estados se visualizan informaciones adicionales.

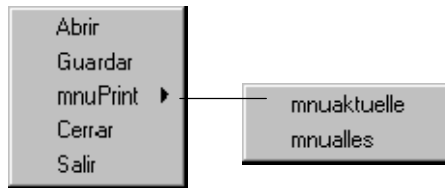
NUM	CAPS	ExtEdit	Palette	12:56
-----	------	---------	---------	-------

NUM: ha sido activada la entrada de números en el bloque numérico
CAPS: tecla de mayúsculas con retención desconectada
ExtEdit: el módulo "Editor externo" está activo
Palette: el nombre del programa actualmente abierto
12:56: el horario actual del sistema

11.3 Menú “Fichero”

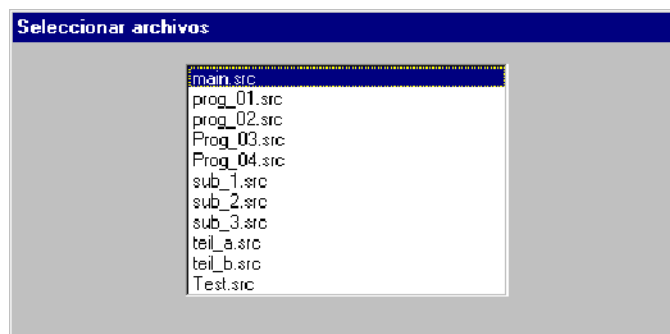


En este menú se encuentran las funciones para la manipulación de ficheros.



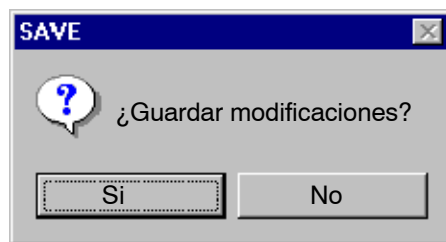
11.3.1 Abrir

Al activar esta opción, aparece la ventana “Seleccionar archivos” tal como descrito en el apartado 11.1.



11.3.2 Guardar

Si el contenido del archivo cargado en el editor fue modificado con las funciones ofrecidas, entonces, antes de memorizar la nueva versión de este archivo, o bien, si se cierra el editor, el sistema requiere una respuesta de seguridad si realmente se aceptan estas modificaciones.



Si quiere Ud. aceptar ahora las modificaciones efectuadas, basta con pulsar la tecla de entrada, confirmando de esta manera la preselección.



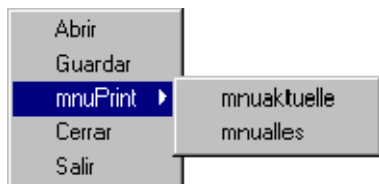
Los contenidos de la versión del fichero hasta ese momento existente y editado, se pierden indefectiblemente, por lo menos en parte.

Para cancelar la acción, deb Ud. primeramente pulsar una de las teclas del cursor, para desplazar el foco al botón “No”. A continuación puede confirmarse la selección pulsando la tecla de entrada. La versión hasta ahora válida del fichero editado queda en este caso sin modificar.

11.3.3 Imprimir (“mnuPrint”)



Esta instrucción permite la edición de programas sobre el medio declarado bajo “Opciones” -> “Output setting” (declaraciones sobre la edición).



Se dispone aquí de las siguientes funciones:

Programa actual (“mnuaktuelle”)

El fichero indicado en el editor externo es editado en la forma representada en la ventana del editor.



```
KRC:\Palette.SRC

*****

INI
1:PTP HOME Vel=100 % DEFAULT

2:PTP P1 CONT Vel=100 % PDAT1 Tool[1] Base[0]
3:LIN P2 CONT Vel=1.5 m/s CPDAT1 Tool[1] Base[0]
4:CIRC P3 P4 Vel=2 m/s % CPDAT2 Tool[1] Base[0]

5:PTP HOME Vel=100 % DEFAULT
END
```

Todos usuarios (“mnualles”)

Se editan todos los programas de aplicación. En la edición con destino al medio “Fichero”, se crea automáticamente el subdirectorio “USERPROG”.



¡La edición a través de la opción “todos usuarios” (“mnualles”) puede tardar algún tiempo de acuerdo con el archivo a editar!

11.3.4 Cerrar

Si el fichero que se encuentra en el editor ha sido modificado desde el momento de la carga, puede Ud. primero asegurar las modificaciones (como descrito en el apartado 11.3.2).

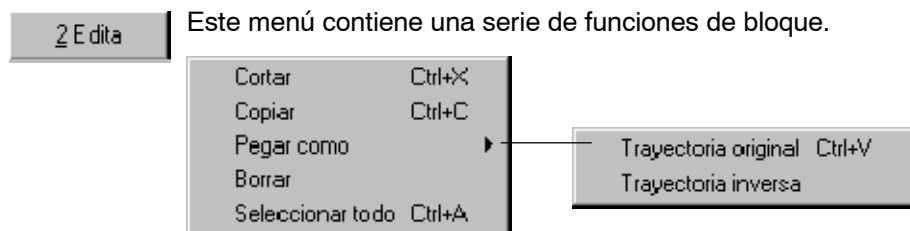
Después de haber cerrado el archivo que se encuentra en el editor, a través del menú “Fichero” y de la opción ofrecida “Abrir” (ver también apartado 11.3.1), puede cargarse otro archivo en el editor.

11.3.5 Salir

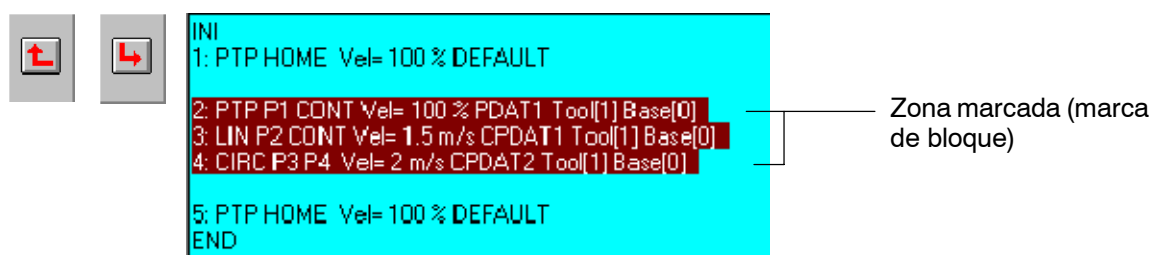
Si el fichero que se encuentra en el editor ha sido modificado desde el momento de la carga, puede Ud. primero asegurar las modificaciones (como descrito en el apartado 11.3.2).

La ventana del editor se cierra y emerge nuevamente la superficie de operación del software del robot.

11.4 Menú “Editar”



Si quiere Ud. marcar una zona, mantenga pulsada la tecla de “MAYUSCULAS” (SHIFT) y pulse una de las teclas de la función de estado “Arriba” o “Abajo” o la correspondiente tecla del cursor.



La marcación de un bloque puede desactivarse por medio de la tecla “ESC”.

11.4.1 Cortar (“CTRL”–“X”)

Borra el sector de trayectoria marcado del programa, y lo coloca en el portapapeles preparado para insertar en otro lugar.

11.4.2 Copiar (“CTRL”–“C”)

Copia el sector de trayectoria marcado del programa, y lo coloca en el portapapeles preparado para insertar en otro lugar.

11.4.3 Pegar como ...

trayectoria original (“CTRL”+“V”)

Pega el contenido del portapapeles en la misma secuencia como efectuado el recorte o el copiado.

trayectoria inversa

Pega nuevamente el contenido del portapapeles en la secuencia inversa.

El último punto de la trayectoria del sector de trayectoria marcado y recortado o copiado, es colocado al comienzo del bloque insertado; el primer punto de la trayectoria al final.

Si por ejemplo, se ha programado por aprendizaje el recorrido hacia un dispositivo o útil, esta trayectoria puede insertarse fácilmente como recorrido de retorno. De este modo no es necesario efectuar la programación del camino de retorno.

11.4.4 Borrar

El sector de trayectoria marcado es borrado sin memorizarlo en el portapapeles.



El borrado de un sector de trayectoria marcado, no puede ser repuesto. Si Ud. ha borrado equivocadamente una parte del programa, entonces cierre el fichero sin memorizar las modificaciones, y a continuación ábralo nuevamente.

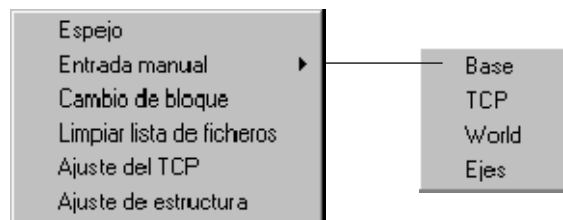
11.4.5 Seleccionar todo

Queda marcado el programa completo que se encuentra cargado en el editor.

11.5 Menú “Utilidades”

3 Utilidades

En este menú se encuentran opciones para la manipulación geométrica del programa.



Con las teclas del cursor “↑” o “↓” mueve Ud. la marca de inserción de campo de entrada a campo de entrada.



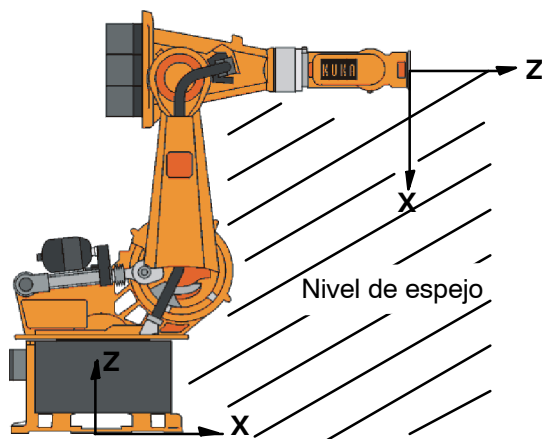
Con este softkey asume Ud. los valores declarados.



Este softkey finaliza la entrada sin guardar los valores.

11.5.1 Espejo

Con ayuda de esta función puede Ud. efectuar una simetría especular en el plano X-Z del sistema de coordenadas \$ROBROOT, de los puntos de la trayectoria programados. La condición es que como mínimo un paso de movimiento se encuentre marcado.



Después de haber seleccionado la opción, se presenta en pantalla una ventana de diálogo, en la cual Ud. debe indicar el nombre del fichero, en donde se ha de memorizar el programa cargado con los puntos de trayectoria simetrizados.



Pulsando la tecla de entrada se arranca el procedimiento y el programa es guardado bajo el nombre indicado. Este programa puede ser cargado, a continuación, en el editor.



Si quiere cancelar la acción, pulse la tecla “ESC”.

11.5.2 Entrada manual

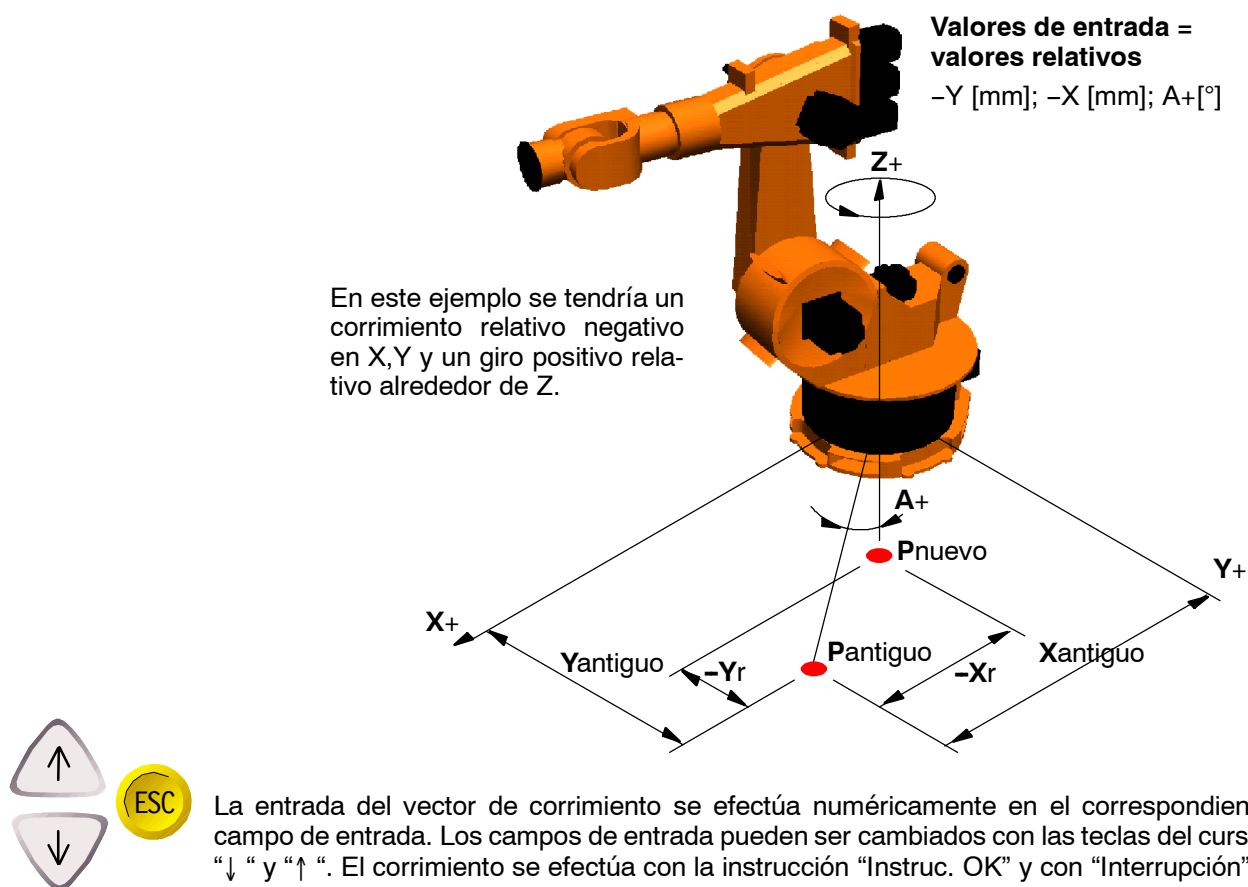
Con ayuda de esta función puede Ud. efectuar el corrimiento de la posición de los puntos de trayectoria programados en:

- el sistema de coordenadas de la pieza (BASE)
- el sistema de coordenadas de la herramienta (TCP)
- el sistema de coordenadas universales (WORLD) o
- coordenadas específicas del eje.

Después de seleccionar una de las posibles opciones, aparece en pantalla una ventana para la declaración de los valores deseados. Dependiendo del sistema de coordenadas declare aquí el corrimiento y el giro, o los ángulos de los ejes.

BASE

Bajo corrimiento cartesiano base de un punto se entiende un corrimiento relativo de un punto cualquiera en el espacio en el sistema de coordenadas originales (WORLD), que se encuentra en el pie del robot. Por medio del ejemplo aquí representado, quiere aclararse el corrimiento cartesiano base de un punto:



Indique los valores del corrimiento (X, Y, Z) en [mm] y el giro (A, B, C) en [Grados] en los campos de entrada ofrecidos.

Mover programa: BASE

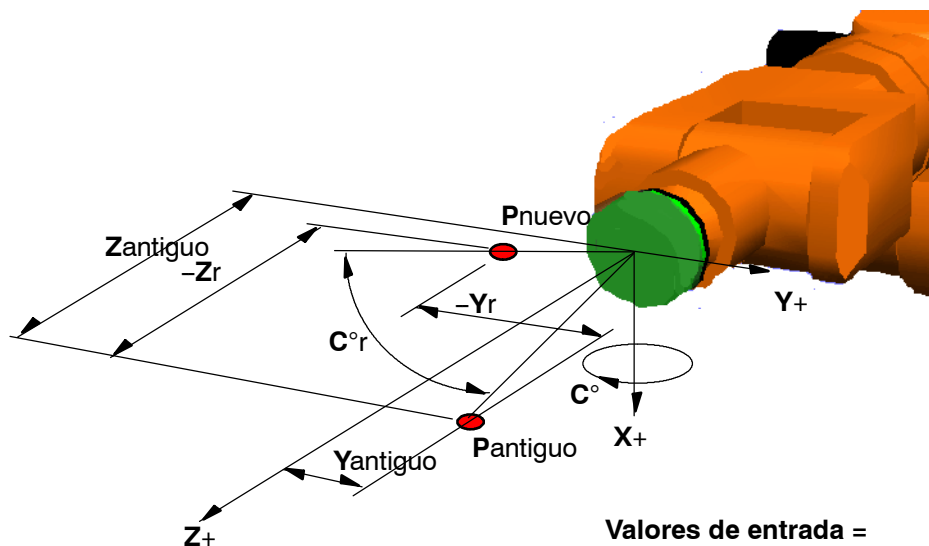
X= [mm] A= [deg.]

Y= [mm] B= [deg.]

Z= [mm] C= [deg.]

TCP

Un corrimiento cartesiano TCP de un punto significa, que un punto cualquiera en el espacio es desplazado de forma relativa respecto al sistema de coordenadas TOOL. Por medio del ejemplo aquí representado quiere aclararse el corrimiento cartesiano TCP de un punto:



En este ejemplo se tendría un corrimiento relativo negativo en X,Y y un giro positivo relativo alrededor de X [°].

¡Un corrimiento relativo en X no es tenido en cuenta!

Valores de entrada = valores relativos

-Y [mm]; -Z [mm]; C+ [°]

La entrada del vector de corrimiento se efectúa numéricamente en el correspondiente campo de entrada. Los campos de entrada pueden ser cambiados con las teclas del cursor "↓" y "↑". El corrimiento se efectúa con la instrucción "Instruc. OK" y con "Interrupción" o "Esc" desechado en cualquier momento.

Indique los valores del corrimiento (X, Y, Z) en [mm] y el giro (A, B, C) en [Grados] en los campos de entrada ofrecidos.

Mover programa: TCP

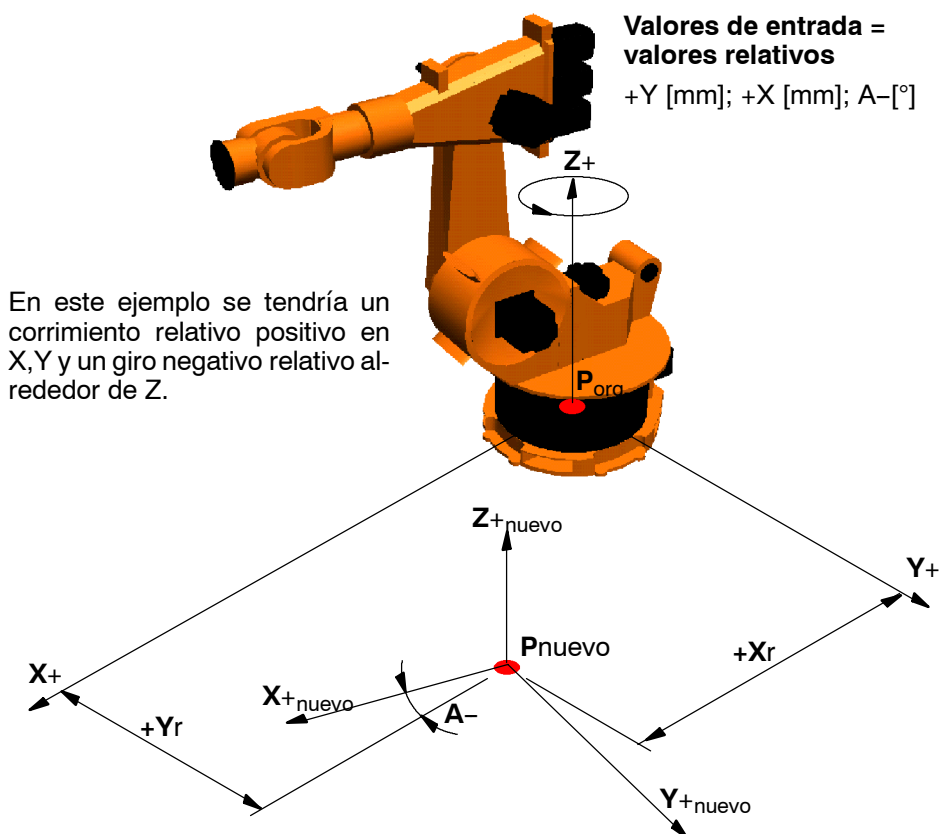
X= [mm] A= [deg.]

Y= [mm] B= [deg.]

Z= [mm] C= [deg.]

WORLD

Bajo corrimiento WORLD de un punto, se entiende que el sistema de coordenadas de origen, que de forma estándar se encuentra en el pie del robot, es desplazado relativamente. Por medio del ejemplo aquí representado pretende aclararse el corrimiento WORLD de un punto:



Indique los valores del corrimiento (X, Y, Z) en [mm] y el giro (A, B, C) en [Grados] en los campos de entrada ofrecidos.

Mover programa: WORLD

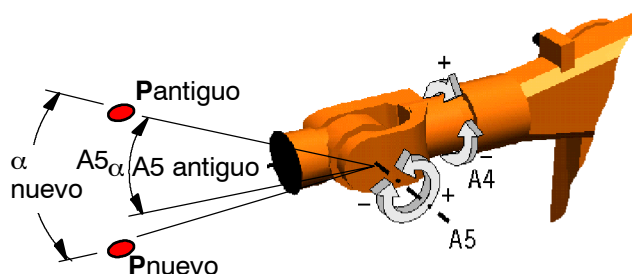
X= [mm] A= [deg.]

Y= [mm] B= [deg.]

Z= [mm] C= [deg.]

Ejes

Bajo corrimiento específico del eje de un punto se entiende, que un punto cualquiera en el espacio es alcanzado por medio de un movimiento relativo de los ejes. Por medio de la figura aquí representada, quiere efectuarse un corrimiento relativo del eje 5.



En este ejemplo se tendría un giro relativo en dirección positiva de A5 [°].
¡Un giro relativo de los demás ejes no es tenido en cuenta!

Valores de entrada = valores relativos
+A5 [°]

La declaración del giro del eje debe efectuarse en grados o en incrementos del correspondiente eje. Los campos de entrada pueden ser cambiados con las teclas del cursor “↓” y “↑”. El corrimiento se efectúa con la instrucción “Instruc. OK” y con “Interrupción” o “Esc” desechado en cualquier momento.

Indique los valores de los ángulos de los ejes (A1 ... A6) en [Grados].

Mover programa: AXIS

A1= <input type="text" value="0.0000"/> [deg.]	A4= <input type="text" value="0.0000"/> [deg.]
A2= <input type="text" value="0.0000"/> [deg.]	A5= <input type="text" value="0.0000"/> [deg.]
A3= <input type="text" value="0.0000"/> [deg.]	A6= <input type="text" value="0.0000"/> [deg.]



Las indicaciones también pueden efectuarse de forma incremental (E1-E6)[Inc].



Incrementos = Impulsos de giro de los accionamientos de los ejes

11.5.3 Cambio de bloque

Con ayuda de esta función puede Ud. modificar datos de movimiento en las partes marcadas en el programa. Para recordar: la marcación de bloques se efectúa pulsando una de las teclas de estado “Ultimo”, o “Siguiente” manteniendo pulsada la tecla de “MAYUSCULAS”.

Después de la llamada de la función, aparece la ventana que se muestra más abajo:

No	Tiempo	Mensaje	Result
004	10:37:21	Cambiar Bloque	Copy
003	10:37:21	Cambio de bloque	On
002	10:37:16	Fichero Abrir	OK
001	10:37:16	ExtEdit start	OK

Con las teclas del cursor “→” o “←” puede Ud. mover la marca de inserción de campo de entrada a campo de entrada.



Los campos de listas se abren con la combinación de teclas “ALT”+“↓”, o “ALT”+“↑”.

Modificar

Si Ud. ha efectuado las entradas deseadas, entonces pulse la tecla del softkey “Modificar”.

Cancelar

La función puede ser abandonada en cualquier momento pulsando la tecla del softkey “Cancelar”.

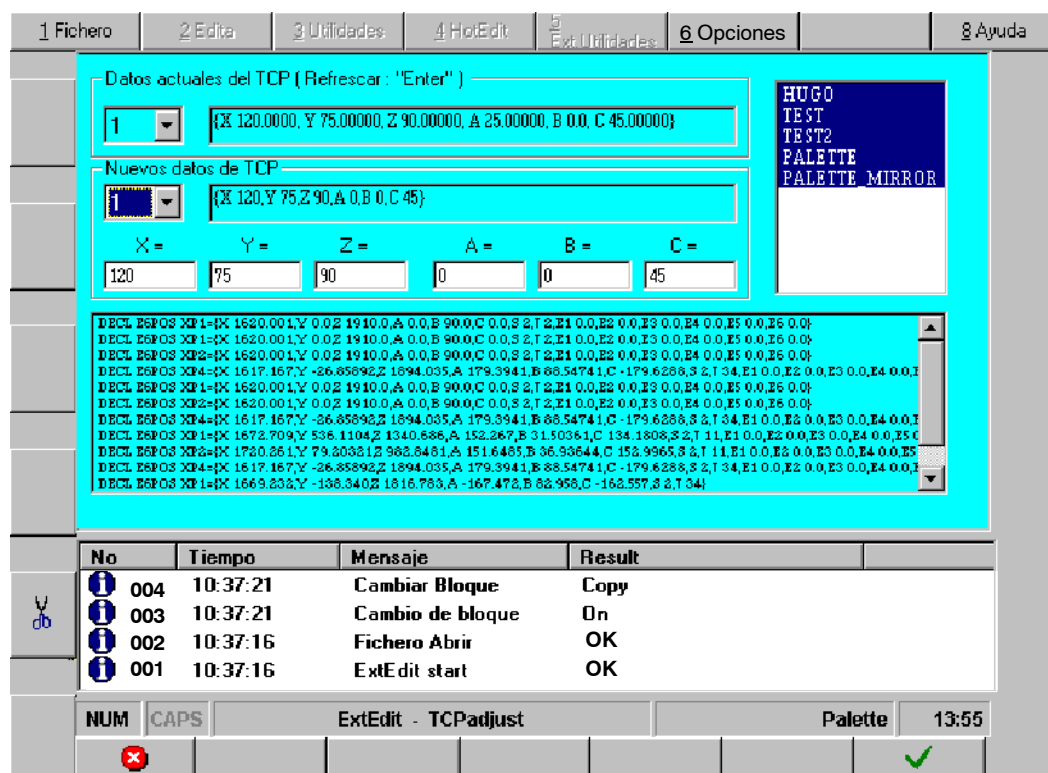
11.5.4 Limpiar lista de ficheros

Después del llamado de esta función se borran los puntos de trayectoria y parámetros de movimiento no referenciados de la lista de datos (*.DAT) pertenecientes al programa.

11.5.5 Ajuste del TCP o de BASE

Con ayuda de esta función puede Ud. adaptar el programa cargado en el editor a otro sistema de coordenadas de la herramienta (TOOL) resp. de la pieza (BASE).

Después del llamado de la función, aparece la ventana ilustrada más abajo. Como ejemplo, se seleccionó aquí la adaptación TCP.



No	Tiempo	Mensaje	Result
004	10:37:21	Cambiar Bloque	Copy
003	10:37:21	Cambio de bloque	On
002	10:37:16	Fichero Abrir	OK
001	10:37:16	ExtEdit start	OK

Se visualizan todos los programas con aquellos puntos a los cuales hace referencia la adaptación de Tool o de Base.

Para poder mover la marca de inserción de campo en campo, deben estar activadas las funciones de mando del cursor en el campo numérico. Para ello, pulse en forma corta, la tecla "NUM". A continuación puede Ud. con la tecla de mayúsculas ("SHIFT") desplazar la marca de entrada entre campos.



Los campos de listas se abren con la combinación de teclas "ALT"+"↓", o "ALT"+"↑".

Del campo de listas seleccione Ud. en el grupo "Nuevos datos de TCP" la herramienta a la cual se ha de adaptar el programa cargado. En los campos de entrada "X=", "Y=", "Z=", "A=", "B=" und "C=" pueden indicarse los nuevos datos de herramienta también en forma manual.

Si Ud. ha efectuado las modificaciones deseadas, pulse por favor, la tecla del softkey "Cambio". También puede abandonar la función en cualquier momento pulsando la tecla del softkey "Cancelar".

11.6 Menú “HotEdit”

4 HotEdit

El menú “HotEdit” posibilita efectuar el corrimiento de coordenadas del punto en el sistema de coordenadas Tool, Base y universales en línea (on line) durante la ejecución de un programa. El punto del menú “Limits” permite limitar este corrimiento.



11.6.1 Base, TCP y World

Desplace el cursor sobre el paso de movimiento que quiera desplazar, o bien, marque varios pasos de movimiento.

Después de seleccionar algunas de las opciones aparece en el borde inferior de la pantalla una ventana de diálogo, en donde pueden declararse los corrimientos (X, Y, Z) y los giros (A, B, C) referidos al sistema de coordenadas correspondiente. La ventana de diálogo de la figura sirve como ejemplo para las demás ventanas de diálogo.

Mover programa: BASE

X=	<input type="text" value="0.0000"/>	[mm]	A=	<input type="text" value="0.0000"/>	[deg.]
Y=	<input type="text" value="0.0000"/>	[mm]	B=	<input type="text" value="0.0000"/>	[deg.]
Z=	<input type="text" value="0.0000"/>	[mm]	C=	<input type="text" value="0.0000"/>	[deg.]

Con las teclas del cursor “↑” o “↓” mueve Ud. la marca de inserción de campo de entrada a campo de entrada.

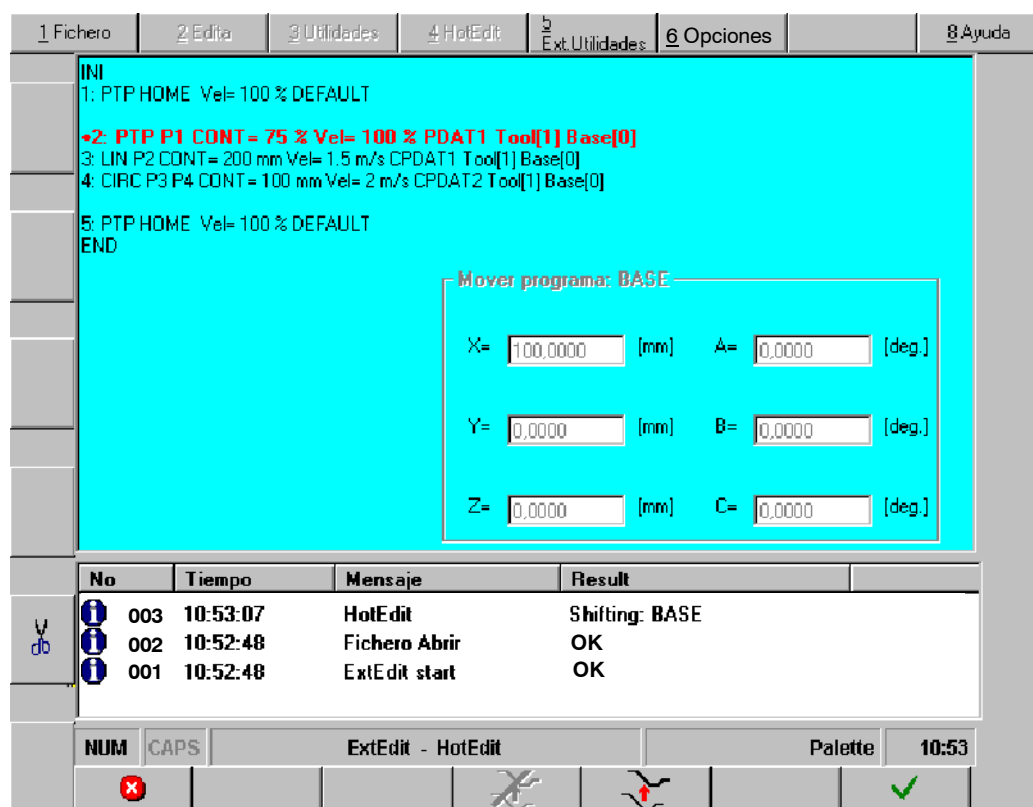


Si Ud. ha efectuado todas las declaraciones necesarias para el corrimiento (cifras enteras), pulse ahora la tecla del softkey “Ok”.



Si Ud. quiere finalizar la acción sin haber efectuado algún corrimiento en el programa, pulse la tecla del softkey “Cancelar”.

Debido a ello, la ventana de diálogo empalidece con fondo gris y se visualizan en pantalla dos nuevos softkeys adicionales.



Ahora puede Ud. asignar los valores declarados, pulsando la tecla del softkey correspondiente. Hasta este instante, la modificación de datos todavía no fue memorizada, y por ello todavía puede anularse la acción.



Si Ud. ha decidido de otra manera, puede anular la asignación. Para ello se dispone del softkey que se ve al lado. Este botón recién está disponible, después de haber efectuado la asignación de valores. Esta función es especialmente útil cuando se está ejecutando un programa y se puede constatar inmediatamente el efecto del corrimiento.



Si el corrimiento responde a las necesidades, puede Ud. asegurar las coordenadas del punto en el fichero "DAT".



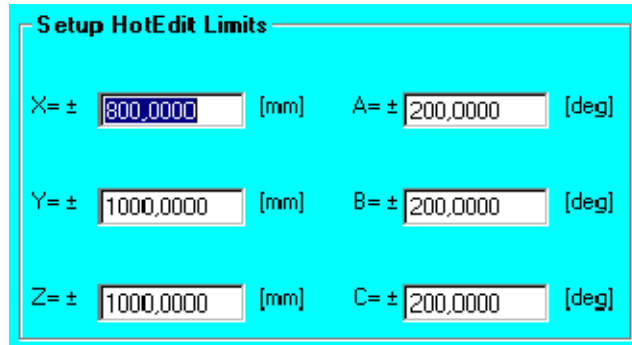
Con el softkey "Cancelar" puede Ud. finalizar la entrada en cualquier momento, sin memorizar ninguna modificación.



Si alguno de los valores declarados se encuentra fuera de las tolerancias, aparece en la ventana de corrimiento del programa una observación o un mensaje de fallo, que el valor declarado rebasa los límites de tolerancia.

11.6.2 Limits

Cuando se selecciona el punto del menú "Limits" aparece la ventana en la cual se encuentran declaradas las tolerancias de corrimiento, que pueden también ser modificadas.



Setup HotEdit Limits

X= ±	<input type="text" value="800.0000"/>	[mm]	A= ±	<input type="text" value="200.0000"/>	[deg]
Y= ±	<input type="text" value="1000.0000"/>	[mm]	B= ±	<input type="text" value="200.0000"/>	[deg]
Z= ±	<input type="text" value="1000.0000"/>	[mm]	C= ±	<input type="text" value="200.0000"/>	[deg]

Para efectuar una modificación, debe colocar Ud. primeramente el cursor sobre el lugar deseado, con ayuda de las teclas de flecha "↑" o "↓", y allí declarar la nueva tolerancia a través del bloque numérico.



Pulse la tecla del softkey "OK" para memorizar los valores y finalizar la acción.



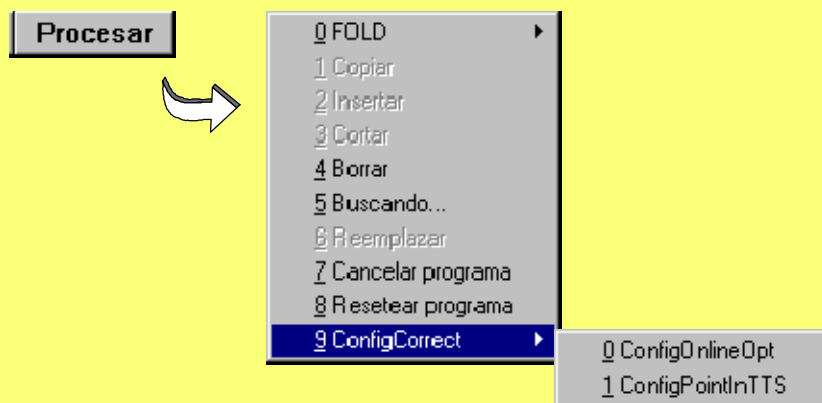
Si desea finalizar sin memorizar, pulse la tecla de "Cancelar".

11.6.3 TTS (Sistema de coordenadas de corrección)

Esta función sirve para la optimización de puntos en el espacio con ayuda del trípode de tecnología (TTS) y puede ser utilizada en instrucciones tecnológicas (por ej. soldadura al arco o aplicación de pegamentos). La corrección TTS está disponible a partir del nivel del "Experto" y puede ser utilizada en todos los modos de servicio.



En programas seleccionados no se dispone del menú "Puesta en servicio". En su lugar, utilice la instrucción del menú "Procesar" -> "ConfigCorrect", el cual se ofrece a partir del nivel del experto.



A continuación se abre el editor externo, y se pueden efectuar las correcciones necesarias. La corrección en línea (online) del punto recién se tendrá en cuenta, cuando el puntero de ejecución en avance, o bien, el de ejecución principal, no han alcanzado hasta ese momento el paso de la instrucción.

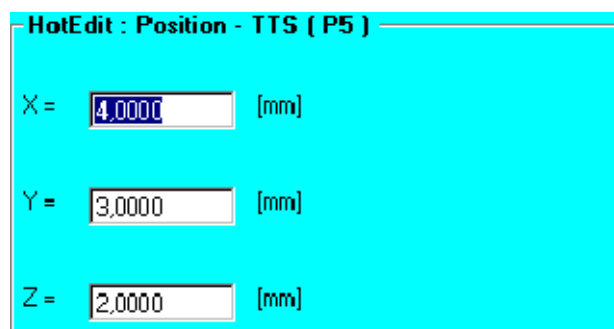
En cada corrección del punto se busca en la lista de datos local (*.DAT) los puntos necesarios. Si uno de los puntos necesarios no es encontrado, se emite el correspondiente mensaje de fallos.

La búsqueda del punto el cual indica la dirección de la trayectoria, se efectúa siempre desde el lugar donde se encuentra el punto a corregir, hacia adelante. La excepción la forma el punto final de un contorno de soldadura o de aplicación de un cordón de pegamento, dado que aquí se busca en los números de pasos próximos más bajos.

Un cambio de base entre dos pasos de movimiento es ignorado.

11.6.3.1 Position-TTS

Marque Ud. en el editor externo la instrucción de movimiento deseada y llame el comando "HotEdit" -> "TTS" -> "Position-TTS". Debido a ello se abre en la pantalla la siguiente ventana.



La opción "Position-TTS" sólo está disponible, cuando se ha seleccionado un paso de movimiento en una aplicación de soldadura o de pegamento.

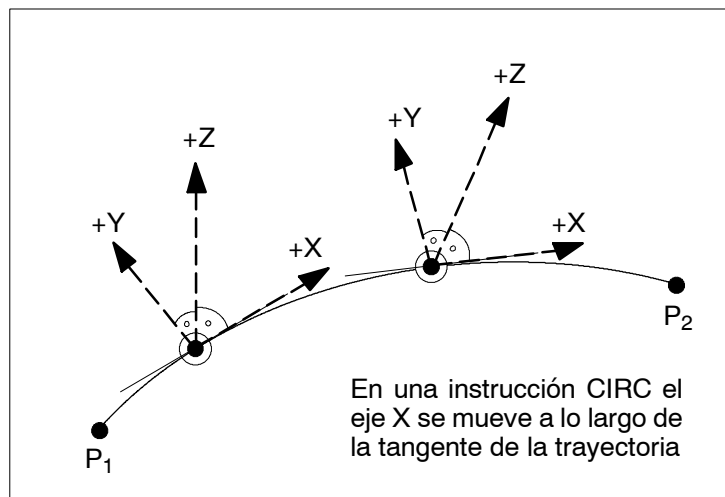
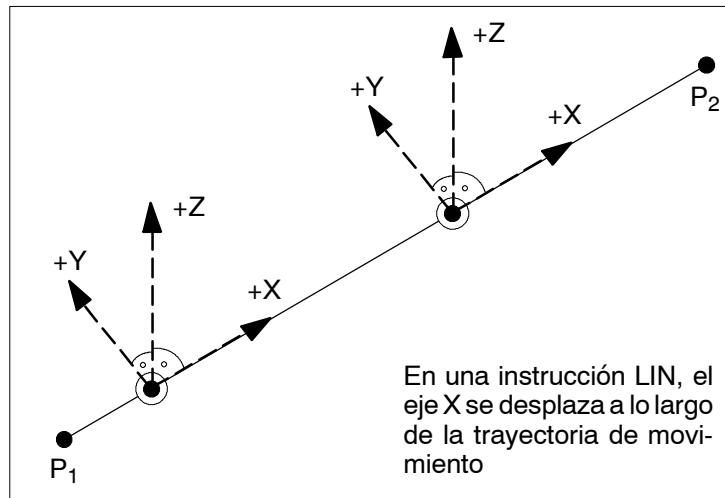


Si se han marcado varias instrucciones de movimiento, la corrección hace referencia siempre a la primera instrucción marcada.

Utilice Ud. las teclas del cursor “↑” o “↓”, para colocar el foco en el sitio deseado. Con las teclas del bloque numérico dé entrada a los nuevos valores.

Eje X

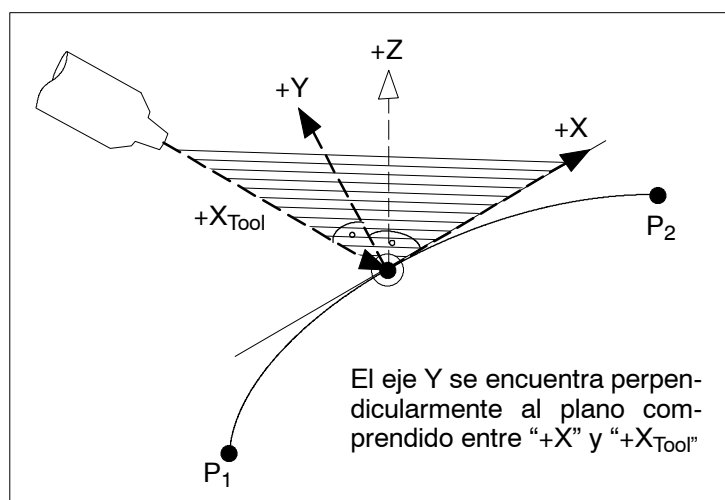
El eje X en el sistema de coordenadas TTS responde a un vector unitario a lo largo de la tangente a la trayectoria.



El eje X del sistema de coordenadas de la herramienta no debe encontrarse de forma paralela a la tangente de la trayectoria, caso contrario el sistema de coordenadas de corrección no puede ser creado. En este caso se emite el correspondiente mensaje de fallo.

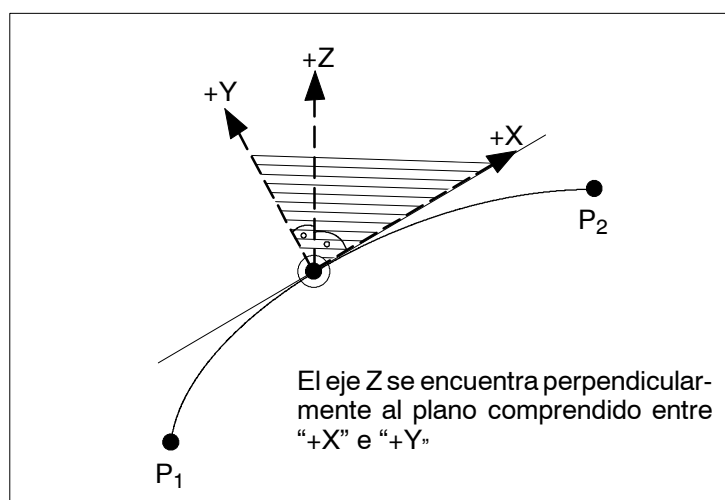
Eje Y

Del vector unitario a lo largo de la tangente de la trayectoria (+X) y el eje X del sistema de coordenadas de la herramienta (+X_{Tool}) se genera un plano. El eje Y se encuentra perpendicular a este plano.



Eje Z

Del vector unitario a lo largo de la tangente de la trayectoria (+X) y el eje Y (+Y) se genera un plano. El eje Z se encuentra perpendicularmente a este plano.



Corrección del punto PTP, LIN



Finalizada la entrada de las modificaciones en los campos de entrada, pulse la tecla del softkey "OK".



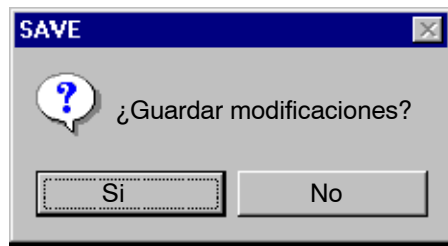
Ahora puede Ud. asignar los valores declarados, pulsando la tecla del softkey correspondiente. Hasta este instante, la modificación de datos todavía no fue memorizada, y por ello todavía puede anularse la acción.




Si Ud. ha decidido de otra manera, puede anular la asignación. Para ello se dispone del softkey que se ve al lado. Este botón recién está disponible, después de haber efectuado la asignación de valores.





Pulse nuevamente la tecla del softkey "OK" y confirme Ud. la próxima pregunta de seguridad, para poder memorizar las modificaciones.





 Con el softkey "Cancelar" puede Ud. finalizar la entrada en cualquier momento.


Corrección del punto CIRC

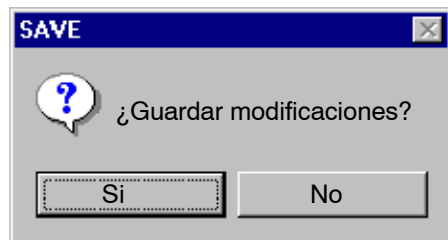
 Después de haber dado entrada a las modificaciones, asigne Ud. el punto intermedio (PI) a las correspondientes correcciones.


 Con este softkey selecciona Ud. el punto final (PF), después de haber dado entrada a las modificaciones en los campos de entrada.

 Ahora puede Ud. asignar los valores declarados, pulsando la tecla del softkey correspondiente. Hasta este instante, la modificación de datos todavía no fue memorizada, y por ello todavía puede anularse la acción.

 Si Ud. ha decidido de otra manera, puede anular la asignación. Para ello se dispone del softkey que se ve al lado. Este botón recién está disponible, después de haber efectuado la asignación de valores.

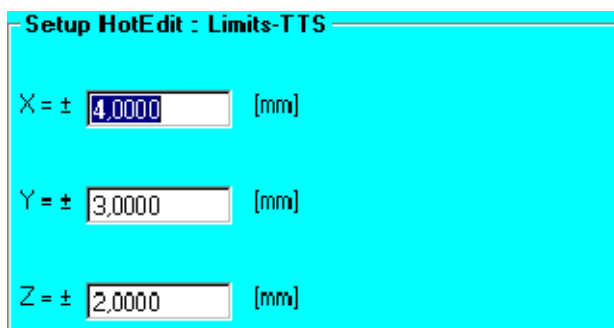
 Si el corrimiento responde a las necesidades, puede Ud. memorizar las modificaciones de las coordenadas del punto, por confirmación de la pregunta de seguridad.



 Con el softkey "Cancelar" puede Ud. finalizar la entrada en cualquier momento, sin memorizar ninguna modificación.

11.6.3.2 Limits-TTS

Aquí puede Ud. definir los valores límites de la corrección TTS. Estos valores límites son controlados automáticamente durante la corrección del punto.




Las tolerancias máximas permitidas se definen por medio del fichero "C:\KRC\Util\HotEdit.ini". Si se indica un valor mayor, se emite automáticamente un mensaje de fallo.



Para la corrección TTS, valores límites mayores de 5 mm carecen de sentido.

Utilice Ud. las teclas del cursor "↑" o "↓", para llevar el foco al lugar deseado. Con las teclas del bloque numérico, defina las nuevas tolerancias.



Pulse la tecla del softkey "OK" para memorizar las tolerancias.

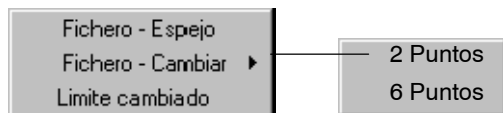


Si Ud. no desea memorizar estos valores, pulse "Interrumpir".

11.7 Menú “Ext Utilidades”



Aquí se ofrecen las funciones para la simetría a espejo, corrimiento y ajuste de los límites de carrera software.

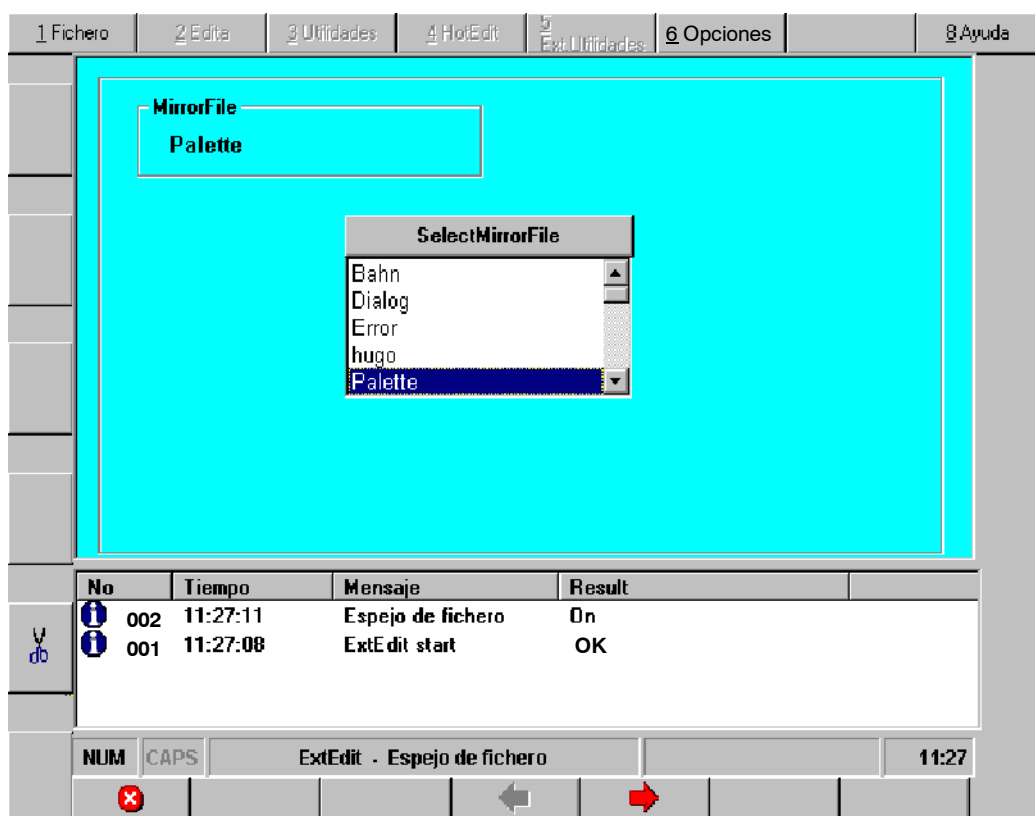


Este menú sólo está disponible mientras no se haya cargado ningún programa en el editor externo.

11.7.1 Fichero – Espejo

Contrariamente a lo que ocurre con la función “Espejo” (apartado 11.5.1), con la cual pueden simetrizarse puntos de trayectoria marcados en pasos de movimiento seleccionados, la función “Fichero–Espejo” permite la simetría de un programa de movimientos completo.

Después de haber arrancado la función, indique primeramente el fichero fuente, el cual se desea simetrizar.



Con las teclas del cursor “↑” o “↓” seleccione Ud. un fichero a simetrizar.



Si ha marcado el fichero deseado, pulse la tecla del softkey “Flecha a derecha”.



La función puede ser abandonada en cualquier momento pulsando la tecla del softkey “Cancelar”.



1 Fichero
2 Edita
3 Utilidades
4 HotEdit
5 Ext.Utilidades
6 Opciones
8 Ayuda

MirrorFile
Palette

Destination
Beispiel

No	Tiempo	Mensaje	Result
002	12:23:03	Espejo de fichero	On
001	12:23:00	ExtEdit start	OK

NUM CAPS ExtEdit - Espejo de fichero 12:23

Para el fichero simetrizado, indique un nombre compuesto de un máximo de 8 caracteres.



Con este softkey retorna Ud. a la última página, en la cual puede seleccionar otro fichero fuente.



Si Ud. le ha dado al fichero el nombre deseado, pulse la tecla de este softkey.

No	Tiempo	Mensaje	Result
002	12:27:17	Espejo de fichero	On
001	12:27:15	ExtEdit start	OK

Ahora tiene Ud. la posibilidad de agregar un comentario, que es visualizado en el navegador.



Con este softkey retorna Ud. a la última página, en la cual puede declarar el nombre del fichero a crear (simetrizado).



Si ha dado entrada al comentario deseado, pulse Ud. este softkey.

A continuación es creado el correspondiente programa. Ahora con ayuda del softkey “Flecha a izquierda” puede pasar páginas hacia atrás para, por ejemplo, crear otro programa o modificar un comentario.



Además, tiene Ud. la posibilidad de abandonar la función “Fichero–Espejo”. Utilice para ello el softkey “Cerrar”.

11.7.2 Fichero – Cambiar

Con esta función puede Ud. mover la posición de los puntos de la trayectoria marcados con la ayuda de un vector (2 puntos), y girar al mismo tiempo el sistema de coordenadas de referencia (6 puntos).

Cambio, 2 puntos

Para la funcionalidad del desplazamiento de 2 puntos, debe programarse por aprendizaje un fichero de referencia (nombre cualquiera; .src, .dat). En este fichero de referencia se memorizan dos puntos. Estos puntos definen el vector de desplazamiento (X, Y, C), medida en la cual se han de desplazar los puntos marcados.

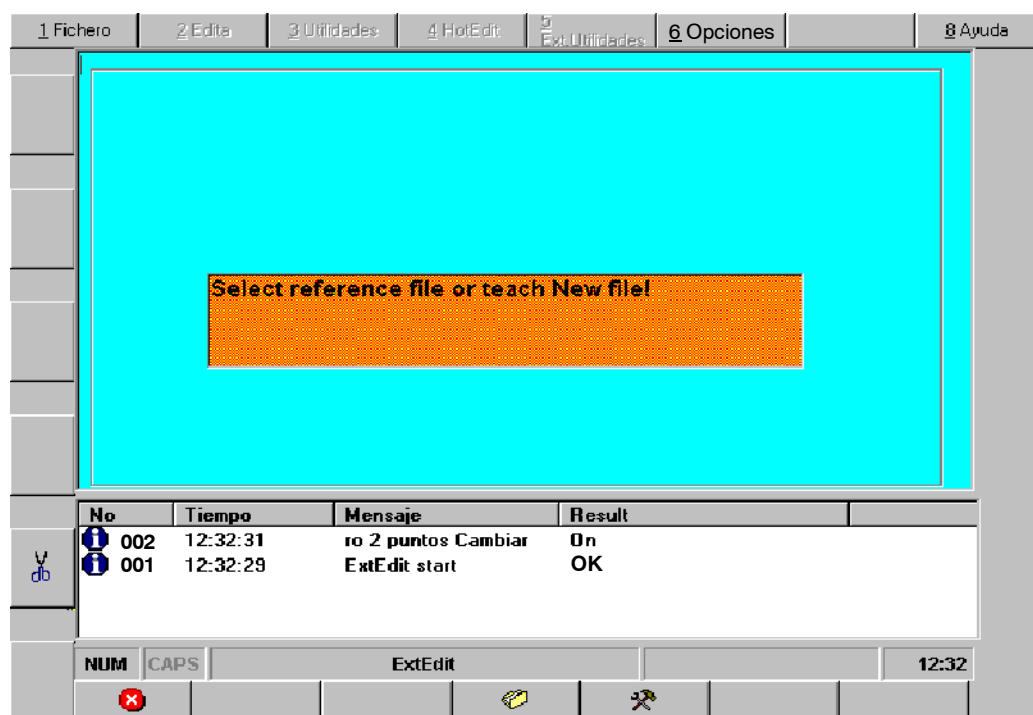
Cambio, 6 puntos

La funcionalidad del sistema de desplazamiento de 6 puntos, trabaja con el mismo principio. En el fichero de referencia correspondiente deben programarse por aprendizaje, seis puntos en total. Los primeros tres puntos definen la base fuente, los últimos tres, la base de destino. Para fuente y destino deben programarse puntos idénticos. Con estos valores se calculan las bases. El corrimiento y el giro entre bases fuente y de destino está determinado por el valor en que los puntos marcados deben ser desplazados.



El vector debe estar programado en un fichero ya antes del llamado de la función. Además deben haberse marcado los distintos pasos de movimiento que se han de mover.

Después de seleccionar la opción Corrimiento 2 puntos o bien, Corrimiento 6 puntos, el programa necesita la información respecto al fichero de referencia.



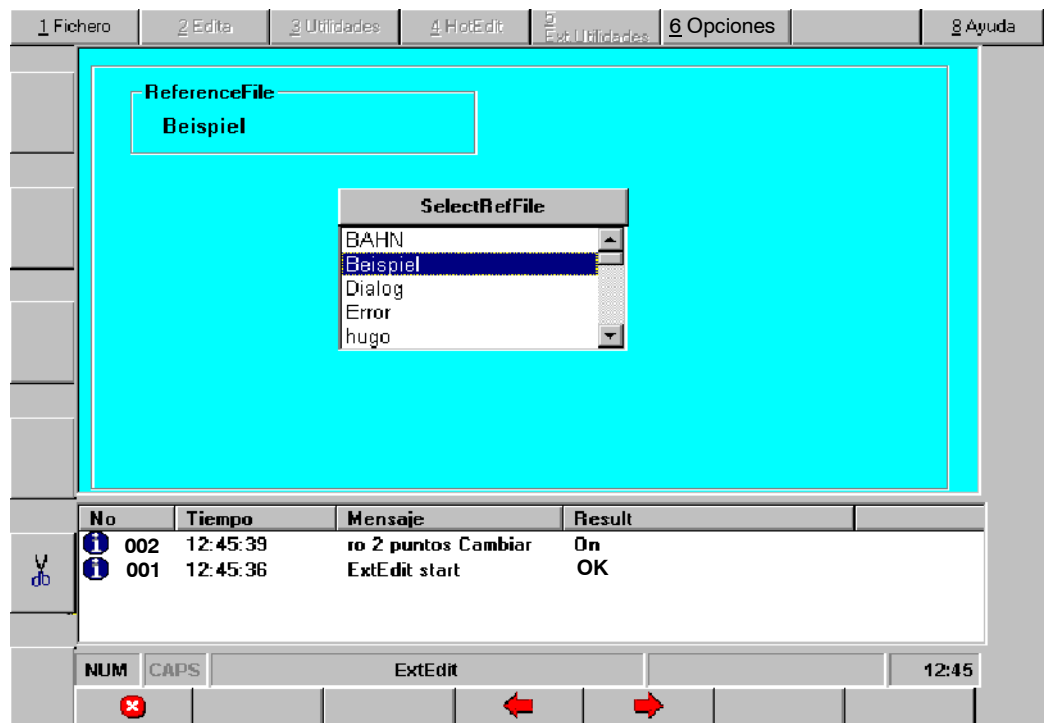
La función puede ser abandonada en cualquier momento pulsando la tecla del softkey "Cancelar".

Hasta aquí, Ud. dispone de las siguientes posibilidades:

11.7.2.1 Utilizar el fichero de referencia existente



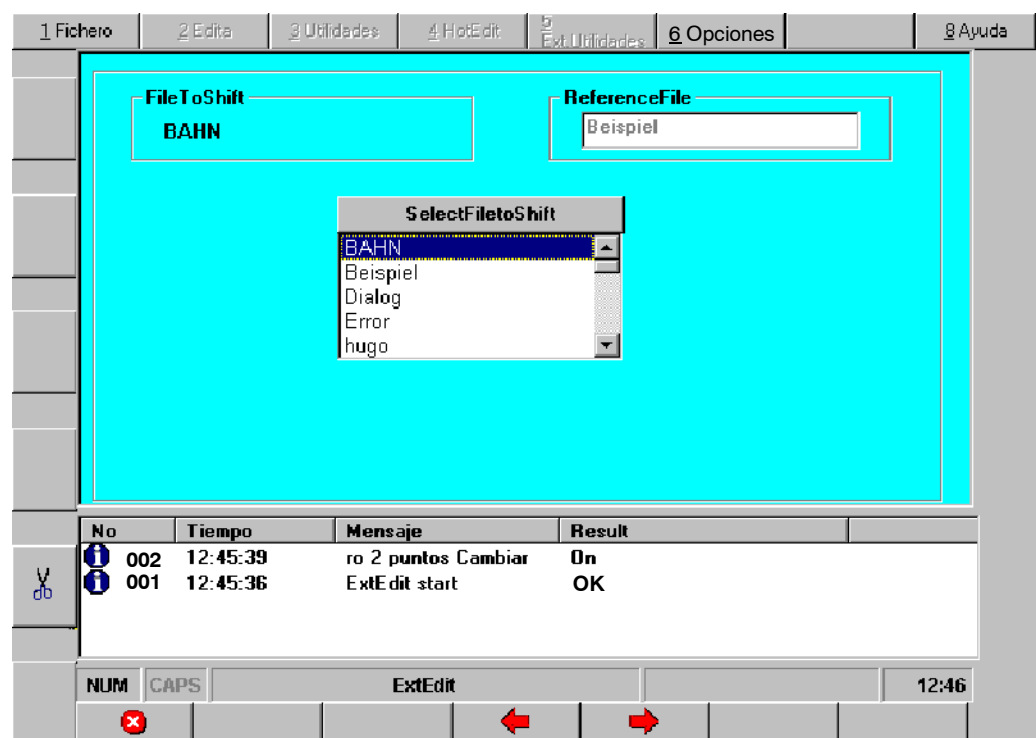
Con este softkey puede Ud. indicar un fichero de referencia así como también un fichero a desplazar.



Con ayuda de las teclas del cursor, seleccione primeramente el fichero de referencia y a continuación pulse la tecla del softkey "Flecha a derecha".



Con esta tecla del softkey se vuelve atrás a la página anterior.

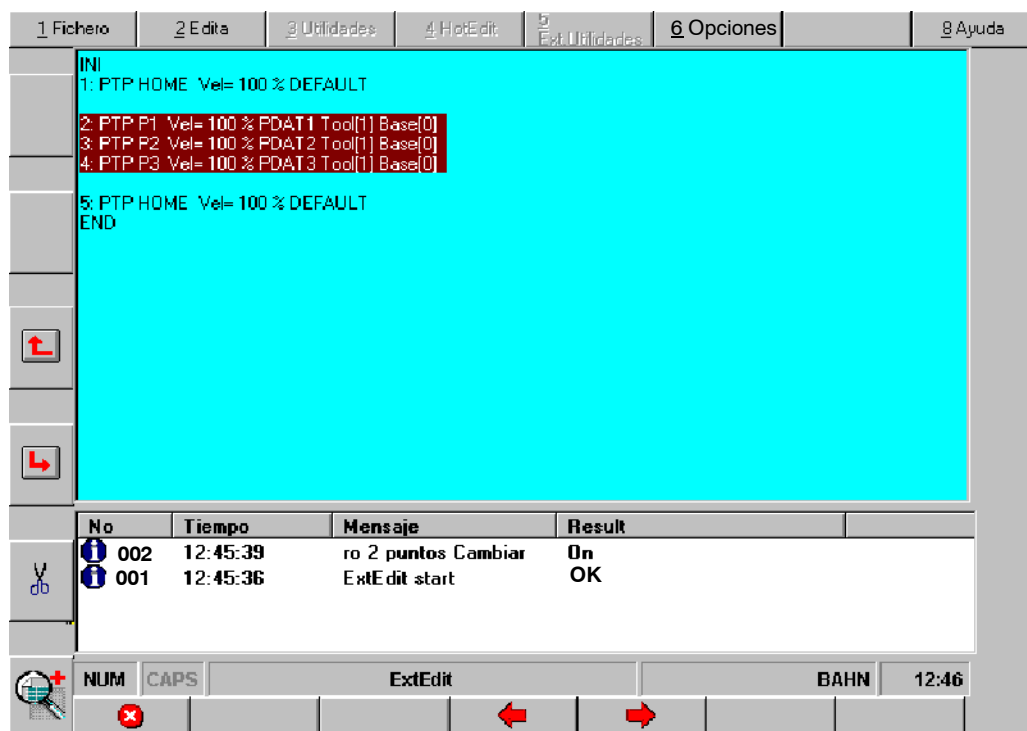


Utilice Ud. nuevamente las teclas del cursor, para marcar el fichero a desplazar. A continuación, pulse nuevamente la tecla del softkey "Flecha a derecha".



Con esta tecla del softkey se vuelve atrás a la página anterior.

A continuación se carga en el editor el fichero a desplazar. Marque aquí las instrucciones de movimiento, los cuales se han de desplazar.



The screenshot shows the KUKA ProHBExpert software interface. At the top, there is a menu bar with options: 1 Fichero, 2 Edita, 3 Utilidades, 4 HotEdit, 5 Ext. Utilidades, 6 Opciones, and 8 Ayuda. The main window is divided into two sections. The top section is a text editor with a cyan background, displaying a file named 'INI' with the following content:

```

1: PTP HOME Vel= 100 % DEFAULT
2: PTP P1 Vel= 100 % PDAT1 Tool[1] Base[0]
3: PTP P2 Vel= 100 % PDAT2 Tool[1] Base[0]
4: PTP P3 Vel= 100 % PDAT3 Tool[1] Base[0]
5: PTP HOME Vel= 100 % DEFAULT
END

```

The bottom section is a message log table with the following data:

No	Tiempo	Mensaje	Result
002	12:45:39	ro 2 puntos Cambiar	On
001	12:45:36	ExtEdit start	OK

At the bottom of the interface, there is a status bar with fields for NUM, CAPS, ExtEdit, BAHN, and a clock showing 12:46. There are also several softkey icons, including a left arrow and a right arrow.



Pulse nuevamente la tecla del softkey “Flecha a derecha”. A continuación los puntos seleccionados son desplazados en el fichero. El avance del proceso es visualizado en la ventana de mensajes.



Con esta tecla del softkey se vuelve atrás a la página anterior.

En la ventana de mensajes se muestra el resultado de la acción.

No	Tiempo	Mensaje	Result
003	13:05:46	ro 2 puntos Cambiar	Position XP1 to XP3 shifted
002	13:05:34	ro 2 puntos Cambiar	On
001	13:05:29	ExtEdit start	OK

11.7.2.2 Crear un nuevo fichero de referencia



Con este softkey se crea un nuevo fichero de referencia. A continuación se selecciona este nuevo programa de referencia, para que el operario programe por aprendizaje, los puntos necesarios.

```
1  INI
2
3  ; Teach here the first position
4
5  ; Teach here the second position
6
```

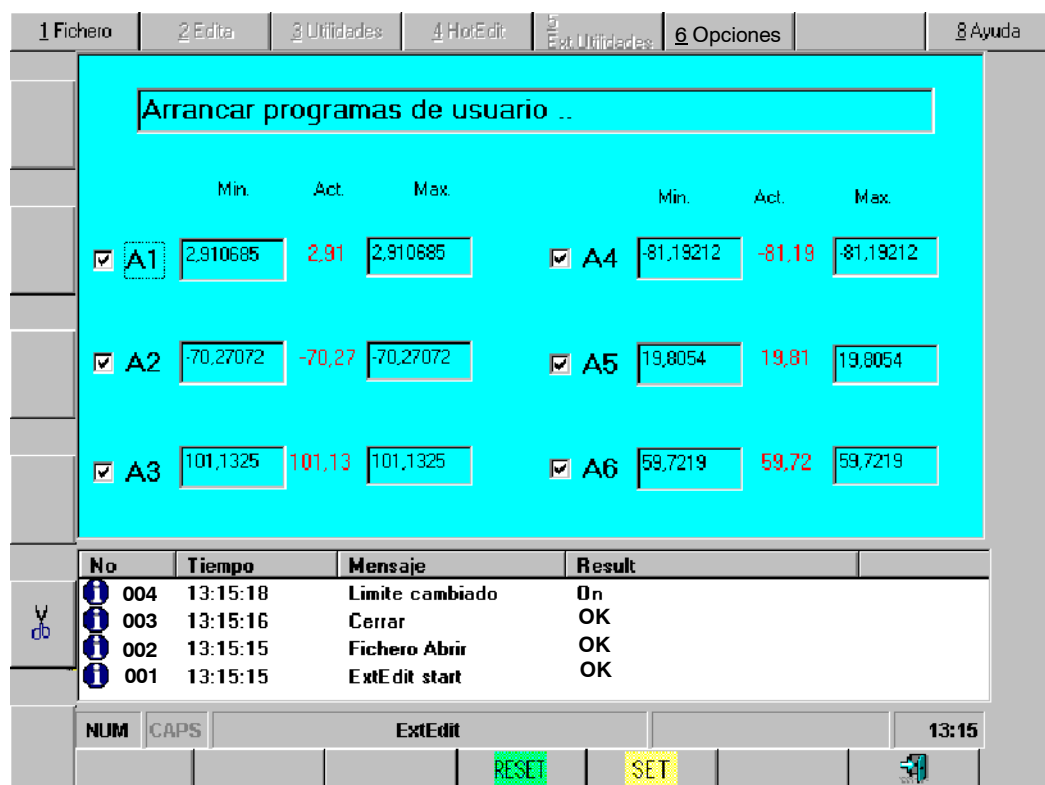
/R1/REF_2POINT.SRC Ln 1, Col 0

Después de la programación por aprendizaje, cancele el programa, y arranque nuevamente el editor externo. Siga el procedimiento tal como descrito en el apartado 11.7.2.1.

11.7.3 Limite cambiado

Con ayuda de esta función, es posible adaptar los límites de carrera software de los distintos ejes a los programas de movimiento.

Para ello, arranque Ud. la función “Limite cambiado” y cambie entonces con la tecla de la ventana, retrocediendo a la superficie de operación del software del robot. Allí arranque los programas de movimiento, cuyos valores de ejes son controlados en segundo plano. Si se han ejecutado todos los programas relevantes, llame nuevamente a la función. Los valores así determinados pueden ser puestos como finales de carrera software.



	Min.	Act.	Max.		Min.	Act.	Max.
<input checked="" type="checkbox"/> A1	2,910685	2,91	2,910685	<input checked="" type="checkbox"/> A4	-81,19212	-81,19	-81,19212
<input checked="" type="checkbox"/> A2	-70,27072	-70,27	-70,27072	<input checked="" type="checkbox"/> A5	19,8054	19,81	19,8054
<input checked="" type="checkbox"/> A3	101,1325	101,13	101,1325	<input checked="" type="checkbox"/> A6	59,7219	59,72	59,7219

No	Tiempo	Mensaje	Result
004	13:15:18	Limite cambiado	On
003	13:15:16	Cerrar	OK
002	13:15:15	Fichero Abrir	OK
001	13:15:15	ExtEdit start	OK

NUM CAPS ExtEdit 13:15

RESET SET



Las posiciones de los ejes son controlados en segundo plano mientras la función “Limite cambiado” esté activa.

Los valores máximos y mínimos que aparecen en cada caso, así como las posiciones actuales de los ejes A1 ... A6 se visualizan en los campos correspondientes.



Por medio de una casilla de chequeo pueden marcarse los ejes a los cuales se desea asignar los límites de carrera. Para seleccionar los ejes deseados, utilice las teclas del cursor. La tecla espaciadora oficia de llave de conmutación, para activar o desactivar el eje correspondiente. Un tilde colocado significa que para ese eje, debe activarse el límite de carrera software.



Con “Reset” pueden resetearse los valores de los límites de carrera software al valor original.



Pulsando la tecla del softkey “Set”, se definen nuevamente los límites de carrera software de los ejes seleccionados. Esta acción puede cancelarse y retornarse a la situación original en cualquier momento, y también más adelante, pulsando la tecla del softkey “Reset”.

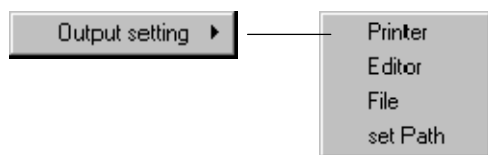


A los valores medidos de cada eje, se agrega un tampón de seguridad de 5°.

11.8 Menú “Opciones”

Opciones

El menú “Opciones” sirve a los efectos de determinar los ajustes básicos del editor externo.



11.8.1 Ouput setting (Declaraciones sobre la edición)

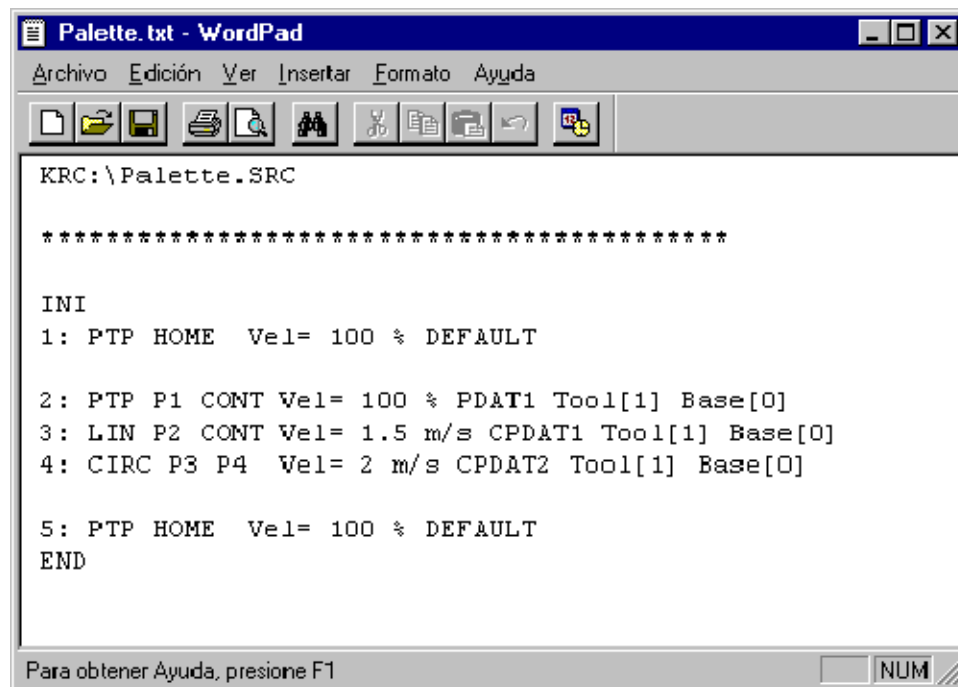
A través de este menú puede determinarse el medio de destino deseado, que encuentra su aplicación en “imprimir” (“mnuPrint”) -> “programa actual” (“mnuaktuelle”) o “imprimir” -> “todos usuarios” (“mnualles”).

Printer (impresora)

Si en el sistema operativo ha sido configurada una impresora, se utiliza éste para la salida de los datos. Si no ha sido configurada ninguna impresora, se edita, en cambio, el mensaje correspondiente en la ventana de mensajes.

Editor

El programa seleccionado es cargado en el editor “WordPad” puesto a disposición por el sistema operativo, para poder trabajar allí con él.



File (archivo)

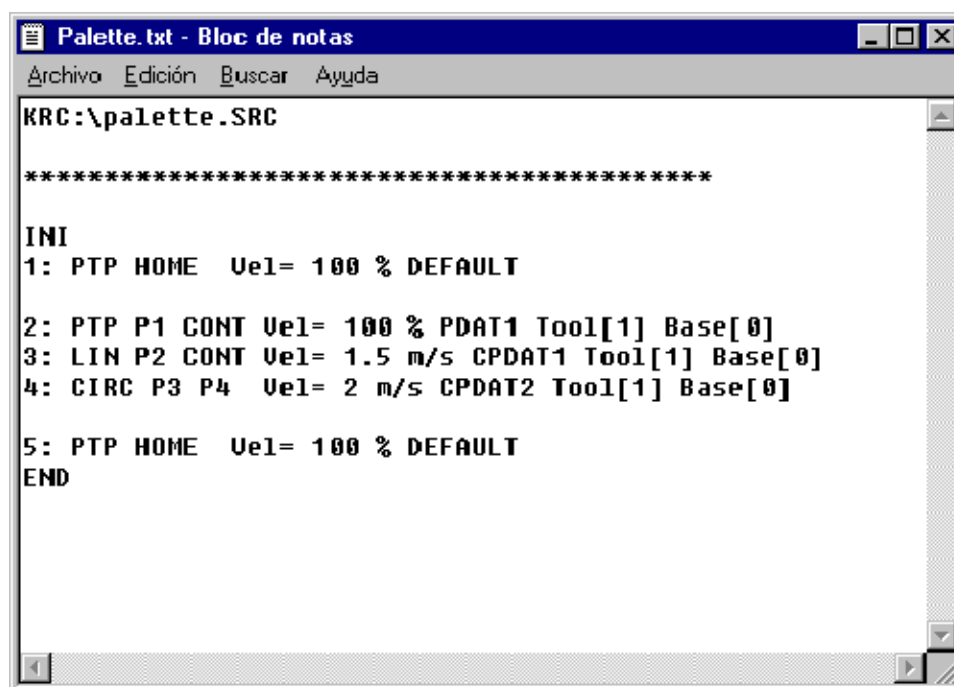
En la edición del programa actual a través de “Print” (imprimir), este programa es guardado como fichero de texto en el trayecto de búsqueda o bajo el nombre del fichero determinado en “Set Path” (crear archivo).

Si se editan todos los programas del usuario mediante “Print” (imprimir), en la carpeta definida bajo “Set Path” (Crear archivo) se crea el subdirectorio “USERPROG” y los programas se guardan allí como ficheros de texto.



La declaración estándar para “Set Path” (crear archivo) es “C:\”

Con un editor de textos se puede tener acceso a cada uno de los programas guardados y también modificarlos, en caso necesario.



```

Palette.txt - Bloc de notas
Archivo Edición Buscar Ayuda

KRC:\palette.SRC

*****

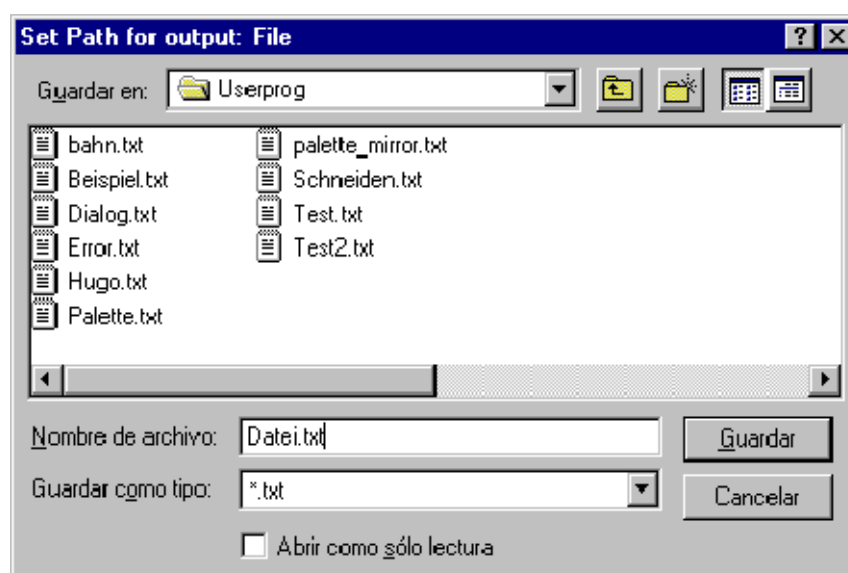
INI
1: PTP HOME Vel= 100 % DEFAULT

2: PTP P1 CONT Vel= 100 % PDAT1 Tool[1] Base[0]
3: LIN P2 CONT Vel= 1.5 m/s CPDAT1 Tool[1] Base[0]
4: CIRC P3 P4 Vel= 2 m/s CPDAT2 Tool[1] Base[0]

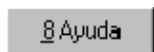
5: PTP HOME Vel= 100 % DEFAULT
END
  
```

Set Path (crear archivo)

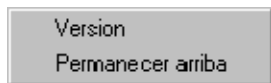
Esta opción sólo es necesaria cuando se quieren guardar programas como fichero en el disco duro. En la ventana de diálogo puede indicarse el directorio de destino así como también el nombre del fichero a imprimir. En la edición de todos los programas del usuario a través de “File” (archivo) se crea el subdirectorio “USERPROG” y los ficheros de texto se guardan allí.



11.9 Menú “Ayuda”



En este menú Ud. dispone de la posibilidad de interrogar la versión y una opción de visualización.



11.9.1 Versión

En la ventana de mensajes le es indicado el número de versión del editor externo.

No	Tiempo	Mensaje	Result	
003	18:24:55	Version	4.1.14	Número de versión
002	18:24:39	Fichero Abrir	OK	
001	18:24:39	ExtEdit start	OK	

11.9.2 Permanecer arriba

Después de haber seleccionado esta opción ya no aparece más en primer plano la superficie de operación del software del robot hasta finalizar el editor.



NOTAS:

Símbolos

.ERR, 12
 !, 106
 !-símbolo, 103
 ?, 107
 # – símbolo, 41
 #BASE, 65, 79
 #CONSTANT, 78
 #INITMOV, 78, 79
 #PATH, 79
 #TCP, 65
 #VAR, 78
 \$ – símbolo, 55
 \$ACC.CP, 77
 \$ACC.ORI1, 77
 \$ACC.ORI2, 77
 \$ACC_AXIS, 66
 \$ADVANCE, 86
 \$ALARM_STOP, 55, 56
 \$APO.CDIS, 93, 96
 \$APO.CORI, 93, 96
 \$APO.CPTP, 90
 \$APO.CVEL, 93, 96
 \$APO_DIS_PTP, 90
 \$BASE, 63, 70
 \$CIRC_TYPE, 79, 82
 \$CONFIG.DAT, 8, 57
 \$CUSTOM.DAT, 56
 \$CYCFLAG, 55
 \$FLAG, 54
 \$IBUS_ON, 56
 \$IPO_MODE, 65
 \$MACHINE.DAT, 8, 56
 \$NULLFRAME, 70
 \$NUM_AX, 56
 \$ORI_TYPE, 78, 82
 \$OUT_C, 133
 \$POS_ACT, 64, 65
 \$PRO_MODE, 22
 \$PSER, 56
 \$ROBCOR.DAT, 8, 57
 \$ROBROOT, 63, 70
 \$SET_IO_SIZE, 133
 \$TIMER, 54
 \$TIMER_FLAG, 54
 \$TIMER_STOP, 54
 \$TOOL, 63, 70
 \$VEL.CP, 77

\$VEL.ORI1, 77
 \$VEL.ORI2, 77
 \$VEL_AXIS, 66
 \$WORLD, 63, 70

A

A10.DAT, 8
 A10.SRC, 8
 A10_INI.DAT, 8
 A10_INI.SRC, 8
 A20.DAT, 8
 A20.SRC, 8
 A50.DAT, 8
 A50.SRC, 8
 Abrir todas FOLD, 19
 ABS(X), 52
 Acción de conmutación, 169
 Aceleración, 77
 Aceleración de eje, 68
 ACOS(x), 52
 Activar interrupciones, 156
 Agregado, 38
 Ajuste de BASE, 196
 Ajuste del TCP, 196
 Ambigüedades, 74
 Ángulo circular, 84
 ANIN, 141
 ANIN OFF, 141
 ANIN ON, 141
 ANIN/ANOUT, 138
 ANOUT OFF, 138
 ANOUT ON, 138
 Aproximación CIRC-CIRC, 96
 Aproximación CIRC-LIN, 96
 Aproximación LIN-CIRC, 98
 Aproximación LIN-LIN, 93
 Aproximación PTP – trayectoria, 99
 Aproximación PTP-PTP, 90
 Aproximación trayectoria – PTP, 100
 Archivos del sistema, 54
 Arco coseno, 52
 Arco seno, 52
 Arco tangente, 52
 ARCSPS.SUB, 8
 Área básica, 74
 Área por sobre la cabeza, 74
 Asignación de valor, 29
 Asistente KRL, 105

ATAN2(Y,X), 52
AXIS, 40, 60, 107

B

B_AND, 49, 51
B_EXOR, 49, 51
B_NOT, 49, 51
B_OR, 49, 51
BAS.SRC, 8, 72
BASE, 65
Bidimensional, 36
Bin Dec, 34
Bloquear / liberar, 156
BOOL, 33, 34
Borrar, 15
BOSCH.SRC, 8
BRAKE, 160
Bucle rechazante, 120
Bucles, 118
Bucles de conteo, 118
Bucles no rechazantes, 121
Bucles sinfín, 123

C

C_DIS, 93, 96, 109
C_ORI, 93, 96, 109
C_PTP, 90, 109
C_VEL, 93, 96, 109
CA, 84
Cadena cinemática, 64, 65
Cadenas de caracteres, 38
Cambio de bloque, 195
Cambio de herramienta, 102
Campo bidimensional, 36
Campo monodimensional, 35
Campo tridimensional, 37
Campos, 35
Cantidad, 52
CELL.DAT, 8
CELL.SRC, 8, 9
Cerrar todas FOLD, 19
CHAR, 33, 35
Cinemáticas de robot ambiguas, 74
CIRC, 84, 113
CIRC !, 103
CIRC REL, 113
CIRC_REL, 84

COI, 73
Coincidencia de paso, 73
Combinación lógica, 48
Combinación lógica de frames, 43
Combinación lógica O, 49
Combinación lógica O exclusivo, 49
Combinación lógica Y, 49
Comentarios, 27
Comienzo del posicionamiento aproximado, 93
Compilador, 12
Compilar, 12
Complementación, 49
Concepto de archivo, 9
CONFIRM, 126
Constante + referencia a la trayectoria, 79
Constante + referencia al espacio, 81
CONTINUE, 88
Contorno de posicionamiento aproximado, 89
Control de ejecución del programa, 115
Control de la orientación, 78
Copiar, 14
Corrección de programa, 14
Cortar, 15
COS(X), 52
Crear programas y editarlos, 11
Crear un programa nuevo, 11
Criterio de distancia, 93
Criterio de orientación, 93
Criterio de velocidad, 93
CSTEP, 22
CTRL-C, 14
CTRL-V, 15
CTRL-X, 15
Cursor de edición, 105

D

DECL, 31
Declaración, 10
DEF, 9, 145
DEFDAT, 175
DEFFCT, 146
Desactivar interrupciones, 156
Desplazamiento a Home, 73
DIGIN, 143
DIGIN OFF, 143
DIGIN ON, 143
Disparo por flancos, 156
DISTANCE, 166

Duración, 30

E

E6AXIS, 40, 107
E6POS, 107
Editar, 12
Editor, 14
Editor externo, 179
ELSE, 116
END, 10
ENDFOLD, 19
ENDFOR, 118
ENDLOOP, 123
ENDWHILE, 120
Enmascarar bits, 50
Entrada manual, 191
Entradas / salidas digitales, 131
Entradas analógicas, 141
Entradas digitales, 143
Entradas y salidas binarias, 128
ENUM, 40
Esconder partes de un programa, 19
Espejo, 190
Estado, 70
Estructura de archivos, 9
Estructura y creación de programas, 7
Estructuras, 38
Estructuras predefinidas, 40
EXIT, 123
EXT, 72

F

Filtrado de bits, 50
Flags, 55, 164
Flags cíclicos, 55
FLT_SERV.DAT, 8
FLT_SERV.SRC, 8
FOLD, 19
FOLD actual abr/cer, 19
FOR, 118
FRAME, 40, 107
Funciones, 9, 145
Funciones de bloque, 14

G

Giro, 70

GLOBAL, 177

Global, 146

GO, 22

GOTO, 115

H

H50.SRC, 8
H70.SRC, 8
HALT, 125
Herramienta fija, 65
Hex Dec, 34

I

Identificación de bloque, 117
IF, 116
Indicación de la posición, 107
Índice, 35
Índice de campo, 35
INI, 72
Insertar, 15
Instrucción, 10
Instrucciones de entrada/salida, 127
Instrucciones de espera, 124
Instrucciones de movimiento, 59
Instrucciones de salto, 115
INT, 33
Intercalar bits, 50
Interpolación referida a la base, 64
Interpolación referida a la garra, 65
INTERRUPT, 154
IR_STOPM.SRC, 8
ISTEP, 22

K

KUKA-Robot-Language, 105

L

Lenguaje máquina, 12
Limits-TTS, 204
Limpiar lista de ficheros, 195
LIN, 83, 111
LIN !, 103
LIN_REL, 83, 111
Lista de datos, 10
Lista de parámetros, 148
Listas de datos, 175
Listas de datos globales, 176

Listas de datos locales, 175

Local, 146

LOOP, 123

M

Manipulación de datos, 42

MARCA, 115

Medición rápida, 163

Modificación de programas, 14

Modos de ejecución del programa, 22

Movimientos circulares, 84

Movimientos con posicionamiento aproximado, 89

Movimientos lineales, 83

Movimientos punto a punto, 66

Movimientos sobre trayectorias, 77

MSG_DEMO.SRC, 8

MSTEP, 22

N

NEW_SERV.SRC, 8

Nombres, 29

O

Objetos de datos, 31

Operador, 42

Operador geométrico, 43

Operador geométrico “:”, 108

Operadores aritméticos, 42

Operadores de bits, 49

Operadores de comparación, 47

Operadores lógicos, 48

Operando, 42

OTTO, 175

P

P00.DAT, 9

P00.SRC, 9

Parada autom. del procesamiento en avance , 87

Parada del procesamiento en avance, 127

Parámetros de movimiento, 104

PERCEPT.SRC, 9

Perfil de marcha más elevado, 67

POS, 40, 107

Posición mecánica cero, 68

Posicionamiento aproximado, 89

Prioridad, 51, 156, 165, 169

Prioridades de los operadores, 51

Procesamiento en avance, 86

Programación de movimiento, 59

Programación por aprendizaje de los puntos, 103

PROKOR, 14

PSTEP, 22

PTP, 66, 69, 109

PTP !, 103

PTP sincrónico, 66

PTP_REL, 69, 109

PUBLIC, 176

PULSE, 136

Punto de la raíz de la muñeca, 74

Punto separador, 38

R

Raíz, 52

Ramificación condicionada, 116

Ramificaciones de programa, 115

REAL, 33, 34

Reemplazar, 16

REPEAT, 121

Reserva de lugar, 106

RESUME, 161

S

S y T, 73

Salidas analógicas, 138

Salidas de impulso, 136

Sección de declaraciones, 9

Sección de instrucciones, 9

Seno, Coseno, Tangente, 52

SIGNAL, 128

SIN(X), 52

Singularidades de la cinemática, 73

Sistema binario, 33

Sistema de coordenadas base, 64

Sistema de coordenadas de corrección, 200

Sistema de coordenadas de la herramienta, 63

Sistema de coordenadas del robot, 63

Sistema de coordenadas específico del eje, 59

Sistema de coordenadas universal, 63

Sistema decimal, 33

Sistema hexadecimal, 33

Sistemas de coordenadas, 59, 70

SPS.SUB, 8
SQRT(X), 52
Status, 73
STRUC, 38
Subprogramas, 9, 145
SWITCH, 117

T

TAN(X), 52
TCP, 65
Temporizador, 54
Tipo de datos, 31
Tipos de datos geométricos, 40
Tipos de datos simples, 33
Tipos de enumeración, 40
Tool Center Point, 66
Touch Up, 107
Transferencia de parámetros, 148
Transformación de avance, 60
Transformación de coordenadas cartesiano, 60
Transformación de las coordenadas, 60
Transformación de retroceso, 60
Translaciones, 62
Tratamiento de errores, 24
Tratamiento de interrupciones, 153
Tridimensional, 37
TRIGGER, 165
Trípode de tecnología, 200
TTS, 200
Turn, 73

U

UNTIL, 121
USER_GRP.DAT, 9
USER_GRP.SRC, 9
USERSPOT.SRC, 9

V

VAR, 106
Variable + referencia a la trayectoria, 79
Variable + referencia al espacio, 82
Variable global, 177
Variables del sistema, 54
Variables y declaraciones, 29
Variables y nombres, 29
Velocidad, 77
Velocidad de eje, 68

Version, 186
Vincular, 12
Vínculo, 12

W

WAIT, 124
WEAV_DEF.SRC, 9
WHILE, 120