

KUKA

Training

KUKA Roboter GmbH

Programación de robots 2

KUKA System Software 8

Documentación para la formación



Edición: 15.12.2011

Versión: P2KSS8 Roboterprogrammierung 2 V1 es



© Copyright 2011

KUKA Roboter GmbH
Zugspitzstraße 140
D-86165 Augsburg
Alemania

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

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 la edición siguiente.

Reservados los derechos a modificaciones técnicas que no tengan influencia en el funcionamiento.

Traducción de la documentación original

KIM-PS5-DOC

Publicación:	Pub COLLEGE P2KSS8 Roboterprogrammierung 2 (PDF-COL) es
Estructura de libro:	P2KSS8 Roboterprogrammierung 2 V2.3
Versión:	P2KSS8 Roboterprogrammierung 2 V1 es

Indice

1 Programación estructurada	5
1.1 Objetivo de la metodología de programación uniforme	5
1.2 Elementos auxiliares para la creación de programas de robot estructurados	5
1.3 Cómo crear un plan de ejecución del programa	10
1.4 Ejercicio: Crear la ejecución del programa	12
2 Introducción al nivel del experto	15
2.1 Utilizar el nivel experto	15
2.2 Ejercicio: Medición Tool y Base	17
2.3 Ejercicio: Navegador experto bucle sínfin	19
3 Variables y declaraciones	21
3.1 Gestión de datos en KRL	21
3.2 Trabajar con tipos de datos simples	24
3.2.1 Declaración de variables	24
3.2.2 Inicialización de variables con tipos de datos simples	27
3.2.3 Manipulación de valores de variables de tipos de datos simples con KRL	28
3.2.4 Ejercicio: Tipos de datos simples	31
3.3 Grupos / campos con KRL	32
3.4 Ejercicio: Campos con tipos de datos simples y bucle de conteo	36
3.5 Crear estructuras con KRL	38
3.6 Ejercicio: Crear estructuras con KRL	40
3.7 El tipo de datos de enumeración ENUM	42
3.8 Ejercicio: Crear un tipo de enumeración con KRL	43
4 Subprogramas y funciones	45
4.1 Trabajo con subprogramas locales	45
4.2 Trabajo con subprogramas globales	47
4.3 Transmitir parámetros a subprogramas	49
4.4 Ejercicio: Subprogramas con transferencia de parámetros	53
4.5 Programación de funciones	55
4.6 Trabajar con funciones estándar de KUKA	57
5 Programación de movimientos con KRL	59
5.1 Programar los movimientos mediante KRL	59
5.2 Programar los movimientos relativos mediante KRL	67
5.3 Calcular o manipular posiciones de robot	74
5.4 Modificar de forma adecuada bits de Status y Turn	75
5.5 Ejercicio: Paletizado y despaletizado	79
6 Trabajar con variables de sistema	83
6.1 Medición del tiempo ciclo mediante temporizador	83
6.2 Ejercicio: Medición del tiempo ciclo y optimización	84
7 Utilización de controles de ejecución de programa	87
7.1 Programar consultas o ramificaciones	87
7.2 Programar el distribuidor	88
7.3 Programar bucles	91

7.3.1	Programar un bucle sinfín	91
7.3.2	Programar bucles de conteo	93
7.3.3	Programar bucle finito	95
7.3.4	Programar bucle infinito	96
7.4	Programar funciones de espera	98
7.4.1	Función de espera dependiente del tiempo	98
7.4.2	Función de espera dependiente de una señal	99
7.5	Ejercicio: Técnicas de bucles	101
8	Funciones de conmutación con KRL	103
8.1	Programación de funciones de conmutación sencillas	103
8.2	Programación de funciones de conmutación referidas a la trayectoria TRIGGER WHEN DISTANCE	106
8.3	Programación de funciones de conmutación referidas a la trayectoria TRIGGER WHEN PATH	
110		
8.4	Ejercicio: Funciones tiempo-distancia en KRL	113
9	Programar con WorkVisual	115
9.1	Gestionar el proyecto con WorkVisual	115
9.1.1	Abrir proyecto con WorkVisual	115
9.1.2	Comparar proyectos con WorkVisual	119
9.1.3	Transmitir el proyecto a la unidad de control del robot (instalar)	123
9.1.4	Activar proyecto en la unidad de control del robot	127
9.2	Editar programas KRL con WorkVisual	130
9.2.1	Manipulación de ficheros	130
9.2.2	Manejo del editor KRL	136
Indice	145

1 Programación estructurada

1.1 Objetivo de la metodología de programación uniforme

Objetivo de la metodología de programación uniforme

Una metodología de programación sirve para:

- hacer frente a problemas complejos de un modo más sencillo mediante una estructura dividida de forma rigurosa
- representar de forma comprensible el proceso en cuestión (sin tener conocimientos de programación más avanzados)
- aumentar la efectividad en el mantenimiento, la modificación y la ampliación de programas

La planificación del programa con previsión tiene como consecuencia:

- poder dividir tareas complejas en tareas parciales simples
- reducir el tiempo total necesario para la programación
- permitir la posibilidad de sustitución de componentes con el mismo rendimiento
- poder desarrollar componentes de forma separada

Los 6 requisitos para un programa del robot:

1. Eficacia
2. Ausencia de errores
3. Inteligibilidad
4. Mantenimiento sencillo
5. Control visual completo
6. Rentabilidad

1.2 Elementos auxiliares para la creación de programas de robot estructurados

¿Qué utilidad tiene un comentario?

Los comentarios son complementos/observaciones dentro del lenguaje de programación. Todos los lenguajes de programación se componen de instrucciones para el ordenador (código) e indicaciones para el procesador de textos (comentarios). Si se continúa el procesamiento (se compila, interpreta, etc.) de un texto fuente, se ignorarán los comentarios del software de procesamiento y, por ello, no tienen ninguna influencia en el resultado.

En el controlador de KUKA se utilizan comentarios de líneas, es decir, los comentarios terminan automáticamente al final de la línea.

Los comentarios por sí solos no pueden hacer que un programa sea legible, pero pueden aumentar considerablemente la legibilidad de un programa bien estructurado. El programador tiene la posibilidad mediante los comentarios de añadir observaciones y explicaciones en el programa sin que sean registradas como sintaxis por el controlador.

El programador es responsable de que el contenido de los comentarios coincida con el estado actual de las indicaciones de programación. En caso de modificaciones del programa, también se deberá comprobar los comentarios y adaptarse si es necesario.

El contenido de un comentario y, de este modo, también su utilización se puede seleccionar libremente por el autor y no está sometido a ninguna sintaxis vinculante. Por regla general, los comentarios se realizan en lenguaje "humano", bien en la lengua materna del autor o en una lengua universal.

- Observaciones sobre el contenido o función de un programa
- El contenido y la utilización se pueden seleccionar libremente
- Se mejora la legibilidad de un programa

- Contribución a la estructuración de un programa
- La responsabilidad de la actualización corresponde al programador
- KUKA utiliza comentarios de líneas
- Los comentarios no son registrados como sintaxis por el controlador

¿Dónde y cuándo se utilizan los comentarios?

Informaciones sobre el texto fuente completo:

Al principio de un texto fuente el autor puede aplicar comentarios previos, incluyendo aquí el autor, la licencia, la fecha de creación, la dirección de contacto en caso de preguntas, la lista de otros ficheros necesarios, etc.

```
DEF PICK_CUBE()  
;Este programa recoge el cubo del depósito  
;Autor: Max Mustermann  
;Fecha de creación: 09.08.2011  
INI  
...  
END
```

Subdivisión del texto fuente:

Los títulos y los apartados se pueden identificar como tales. Para ello, no solo se utilizan frecuentemente medios lingüísticos, sino también medios gráficos que se pueden aplicar a lo largo del texto.

```
DEF PALLETIZE()  
*****  
/*Este programa paletiza 16 cubos sobre la mesa*  
/*Autor: Max Mustermann-----*  
/*Fecha de creación: 09.08.2011-----*  
*****  
INI  
...  
-----Cálculo de las posiciones-----  
...  
-----Paletizado de los 16 cubos-----  
...  
-----Despaletizado de los 16 cubos-----  
...  
END
```

Explicación de una línea individual:

De este modo se puede explicar el procedimiento o el significado de una parte del texto (p. ej. línea del programa) para que otros autores o el propio autor puedan entenderlo mejor posteriormente.

```
DEF PICK_CUBE()  
  
INI  
  
PTP HOME Vel=100% DEFAULT  
  
PTP Pre_Pos ; Desplazamiento a posición previa para agarrar  
  
LIN Grip_Pos ; Cubo desplazamiento a posición de agarre  
...  
END
```

Indicación sobre el trabajo a realizar:

Los comentarios pueden identificar fragmentos de códigos insuficientes o ser un comodín para fragmentos de códigos completamente ausentes.

```

DEF PICK_CUBE()

INI

;Aquí se debe insertar el cálculo de las posiciones de palets!

PTP HOME Vel=100% DEFAULT

PTP Pre_Pos ; Desplazamiento a posición previa para agarrar

LIN Grip_Pos ; Cubo desplazamiento a posición de agarre

;Aquí falta el cierre de la garra

END

```

Insertar punto y coma:

Si se borra provisionalmente un componente del código, aunque posiblemente se vuelve a introducir de forma posterior, se insertará un punto y coma. En cuanto está incluido en el comentario, el fragmento del código ya no es un código desde el punto de vista del compilador, es decir, prácticamente ya no está disponible.

```

DEF Palletize()

INI

PICK_CUBE()

;CUBE_TO_TABLE()

CUBE_TO_MAGAZINE()

END

```

¿Qué provoca la utilización de Folds en un programa del robot?

- En FOLDS se pueden ocultar partes del programa
- Los contenidos de FOLDS no son visibles para el usuario
- Los contenidos de FOLDS se procesan de forma totalmente normal en la ejecución del programa
- Mediante la utilización de Folds se puede mejorar la legibilidad de un programa

¿Qué ejemplos existen para la utilización de Folds?

En el controlador de KUKA ya se utilizan Folds por el sistema de forma estándar, p. ej. en la indicación de formularios en línea. Estos Folds facilitan el control visual completo de los valores introducidos en el formulario en línea y ocultan las partes del programa que no son relevantes para el operario.

Además, para el usuario existe la posibilidad de crear Folds propios (a partir de grupo de usuario experto). Estos Folds se pueden utilizar por el programador, p. ej. para comunicarle al operario lo que está ocurriendo en un lugar determinado del programa, pero manteniendo la sintaxis KRL real en segundo plano.

En general, los Folds se representan primero cerrados tras su creación.

```

DEF Main()
...
INI ; KUKA FOLD cerrado

SET_EA ; FOLD cerrado por usuario

PTP HOME Vel=100% DEFAULT ; KUKA FOLD cerrado

PTP P1 CONT Vel=100% TOOL[2]:Gripper BASE[2]:Table
...
PTP HOME Vel=100% Default

END

```

```

DEF Main()
...
INI ; KUKA FOLD cerrado

SET_EA ; FOLD creado por usuario
$OUT[12]=TRUE
$OUT[102]=FALSE
PART=0
Position=0

PTP HOME Vel=100% DEFAULT ; KUKA FOLD cerrado
...
PTP P1 CONT Vel=100% TOOL[2]:Gripper BASE[2]:Table

PTP HOME Vel=100% Default

END

```

```

DEF Main()
...
INI ; KUKA FOLD cerrado

SET_EA ; FOLD cerrado por usuario

PTP HOME Vel=100% DEFAULT ; KUKA FOLD abierto
$BWDSTART=FALSE
PDAT_ACT=PDEFAULT
FDAT_ACT=FHOME
BAS(#PTP_PARAMS,100)
$H_POS=XHOME
PTP XHOME
...

PTP P1 CONT Vel=100% TOOL[2]:Gripper BASE[2]:Table

PTP HOME Vel=100% Default

END

```

¿Por qué se trabaja con tecnología de subprogramas?

En la programación los subprogramas se utilizan principalmente para permitir una utilización múltiple de partes de tareas iguales y, de este modo, conseguir la prevención de repeticiones de códigos. Esto también permite ahorrar en espacio de almacenamiento, entre otros aspectos.

Otro motivo importante para la utilización de subprogramas es la estructuración resultante del programa.

Un subprograma debe realizar una tarea parcial acotada a sí misma y fácilmente descriptible.

Los subprogramas son actualmente más cortos y con una presentación clara en beneficio de la mantenibilidad y la eliminación de errores de programación, ya que el tiempo y la administración invertidos desde el punto de vista interno del ordenador para la llamada de subprogramas, ya no tiene prácticamente ninguna importancia en los ordenadores modernos.

- Posibilidad de utilización múltiple
- Prevención de repeticiones de códigos

- Ahorro de espacio de almacenamiento
- Los componentes se pueden desarrollar separados unos de otros
- Posibilidad de intercambio de componentes del mismo nivel de rendimiento en cualquier momento
- Estructuración del programa
- Tarea completa dividida en tareas parciales
- Mantenibilidad y eliminación de errores de programación mejoradas

Aplicación de subprogramas

```
DEF MAIN()
INI
LOOP
    GET_PEN()
    PAINT_PATH()
    PEN_BACK()
    GET_PLATE()
    GLUE_PLATE()
    PLATE_BACK()

    IF $IN[1] THEN
        EXIT
    ENDIF

ENDLOOP
END
```

¿Qué provoca la inserción de líneas de instrucciones?

Para aclarar la relación de los módulos del programa, se recomienda insertar secuencias de instrucciones encajadas en el texto de programación y escribir indicaciones de la misma profundidad de apilamiento directamente unas debajo de otras.

El efecto logrado es puramente visual, solo afecta al valor de un programa como medio de información de persona a persona.

```
DEF INSERT()
INT PART, COUNTER
INI
PTP HOME Vel=100% DEFAULT
LOOP
    FOR COUNTER = 1 TO 20
        PART = PART+1
        ; Imposible sangrar formularios inline!!!
    PTP P1 CONT Vel=100% TOOL[2]:Gripper BASE[2]:Table
        PTP XP5
    ENDFOR
    ...
ENDLOOP
```

¿Qué se consigue mediante una identificación adecuada de nombres de datos?

Para poder indicar correctamente la función de datos y señales en un programa del robot, se recomienda utilizar términos significativos en la asignación de los nombres. Aquí se incluyen p. ej.:

- Nombre de texto largo para señales de entrada y de salida
- Nombres de Tool y Base
- Declaraciones de señales para señales de entrada y de salida
- Nombres de puntos

1.3 Cómo crear un plan de ejecución del programa

¿Qué es un PEP? Un plan de ejecución del programa (PEP) es un diagrama de flujo para un programa que también se denomina como plan de estructura del programa. Es una representación gráfica para la transformación de un algoritmo en un programa y describe la secuencia de operaciones para la solución de una tarea. Los símbolos para los planes de ejecución del programa están regulados en la norma DIN 66001. Los planes de ejecución del programa también se utilizan a menudo por programas de ordenador para la representación de procesos y tareas.

En comparación con una descripción basada en un código, el algoritmo del programa presenta una legibilidad considerablemente más sencilla, ya que gracias a la representación gráfica, la estructura se puede reconocer notablemente mejor.

Los errores en la estructura y de programación se evitan de forma más sencilla en la transformación posterior en código de programación, ya que en caso de aplicación correcta de un PEP, es posible la transformación directa en un código de programación. Al mismo tiempo, con la creación de un PEP se obtiene una documentación del programa que se va a crear.

- Herramienta para la estructuración del proceso de un programa
- La ejecución del programa es fácilmente legible
- Los errores en la estructura se pueden reconocer de forma más sencilla
- Documentación simultánea del programa

Símbolos PEP

Comienzo o fin de un proceso o de un programa



Fig. 1-1

Combinación de indicaciones y operaciones

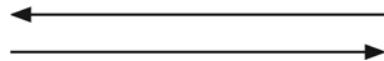


Fig. 1-2

Ramificación

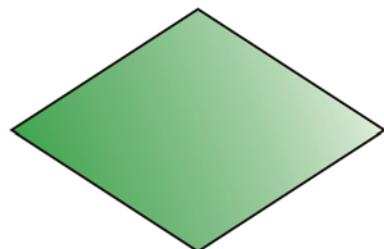


Fig. 1-3

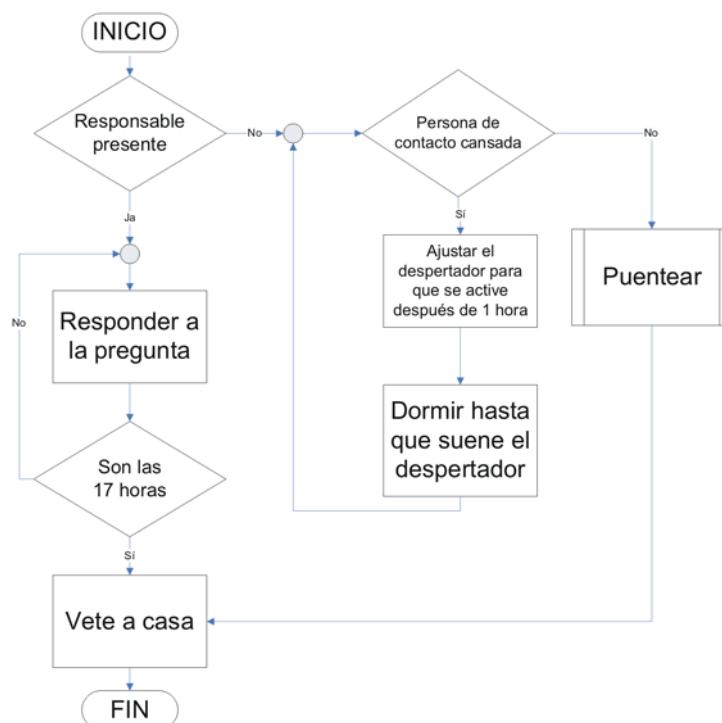
Indicaciones generales en el código de programa

**Fig. 1-4**

Llamada de subprograma

**Fig. 1-5**

Indicación de entrada/salida

**Fig. 1-6****Ejemplo PEP****Fig. 1-7****Cómo crear un PEP**

Partiendo de la presentación de usuario, el problema se perfecciona paso por paso, hasta que se tenga un control visual completo de los componentes elaborados de este modo para poder adaptarlos en KRL. Los proyectos que se crean en los pasos de desarrollo seguidos se diferencian por la precisión de detalles creciente.

1. Subdivisión aproximada del proceso completo en aprox. 1-2 páginas
2. División de la tarea completa en tareas parciales pequeñas
3. Subdivisión aproximada de las tareas parciales
4. Perfeccionamiento de la subdivisión de las tareas parciales
5. Aplicación en código KRL

1.4 Ejercicio: Crear la ejecución del programa

Objetivo del ejercicio	Después de terminar con éxito este ejercicio, se dispone de la competencia necesaria para efectuar las siguientes tareas: <ul style="list-style-type: none">■ Dividir la tarea completa en tareas parciales■ Perfeccionamiento de la subdivisión aproximada■ Creación de un PEP (plan de ejecución del programa)
Requisitos	Las siguientes condiciones son necesarias para efectuar este ejercicio con éxito: <ul style="list-style-type: none">■ Conocimientos teóricos sobre la metodología de programación
Tarea	Descripción de la instalación La tarea del robot consiste en la recogida de piezas de plástico de una máquina de moldeo por inyección. Las piezas se agarran con ventosas de vacío y se apilan sobre una cinta de avance intermitente junto a la máquina de moldeo por inyección.

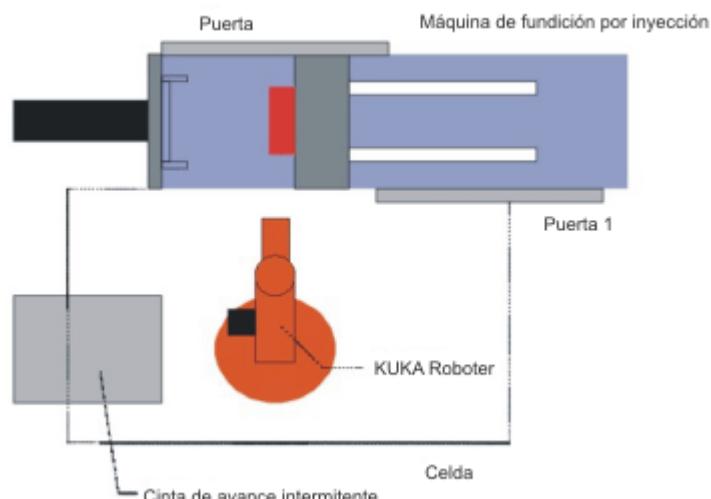


Fig. 1-8

Descripción de la ejecución

1. Despues de que la máquina de moldeo por inyección ha producido una pieza, abre la puerta.
2. El robot se desplaza hasta la posición de recepción tras el control del interruptor de final de carrera de la puerta y agarra el componente.
3. El expulsor expulsa el componente de la herramienta.
4. El robot se desplaza ahora fuera de la máquina y el expulsor vuelve a retroceder.
5. En cuanto el robot ha salido de forma segura de la máquina, se puede cerrar la puerta y producirse una pieza nueva.
6. El componente terminado se deposita a continuación en el lugar libre en la cinta de avance intermitente.

7. Seguidamente la cinta de avance intermitente se desplaza hasta que la posición de descarga esté de nuevo libre.

Tarea

1. Dividir la tarea en módulos de programas adecuados
2. Perfeccionar de nuevo la división aproximada
3. Crear un plan de ejecución del programa

Lo que deberá saberse ahora:

1. ¿Qué requisitos se exigen a un programa?

.....

2. ¿Qué ventajas ofrece la buena legibilidad de un programa?

.....

3. ¿Qué hay que tener en cuenta en la utilización de comentarios?

.....

4. ¿Qué ventajas ofrece la utilización de folds?

.....

2 Introducción al nivel del experto

2.1 Utilizar el nivel experto

Descripción	La unidad de control del robot ofrece diferentes grupos de usuarios con diferentes funciones. Se pueden seleccionar los siguientes grupos de usuarios :
■ Operarios	Grupo para el operario. Esto es el grupo de usuario por defecto.
■ Usuario	Grupo para el operario. (Los grupos de usuarios del operario y del usuario están declarados por defecto para el mismo grupo destinatario.)
■ Experto	Grupo de usuario para el programador. Este grupo de usuario está protegido por una contraseña.
■ Administrador	Mismas funciones a las del grupo de usuarios "Experto". A modo adicional es posible integrar plug-ins en la unidad de control del robot. Este grupo de usuario está protegido por una contraseña.
■ Técnico de mantenimiento de seguridad	Este grupo usuario puede activar y configurar la configuración de seguridad del robot. Este grupo de usuario está protegido por una contraseña.
■ Técnico de mantenimiento de seguridad	Este grupo de usuario solo es relevante cuando se utiliza KUKA.SafeOperation o KUKA.SafeRangeMonitoring. El grupo de usuario está protegido por una contraseña.
Funciones avanzadas del grupo de usuario Experto:	
■ Protegido por contraseña (predeterminada: kuka)	
■ La programación en el editor mediante KRL es posible	
■ Vista en detalle para módulos disponible	
■ Mostrar/ocultar la línea DEF	
■ Abrir y cerrar pliegues (FOLD)	
■ Mostrar la vista en detalle en el programa	
■ En la creación del programa se puede seleccionar entre plantillas predefinidas	
■ El grupo de usuario Experto se vuelve a abandonar automáticamente,	
■ cuando se cambia al modo de servicio AUT o AUT EXT	
■ cuando durante un tiempo determinado no se realiza ninguna operación en la interfaz de usuario (300 s)	
Funciones	Crear programas mediante plantillas
■ Cell : El programa Cell disponible solo se puede sustituir o crearse de nuevo en caso de que se borre Cell.	
■ Expert : Módulo compuesto por ficheros SRC y DAT en los que solo está disponible el encabezamiento y el final del programa.	
■ Expert Submit : Fichero Submit (SUB) adicional compuesto de encabezamiento y final del programa.	
■ Function : Creación de función de fichero SRC, en el que solo se crea el encabezamiento de la función con una variable BOOL. El final de la función está disponible, pero el retorno debe ser programado previamente.	

- **Modul:** Módulo compuesto de ficheros SRC y DAT, en los que está disponible el encabezamiento y el final del programa y la estructura base (INI y 2x PTP HOME).
- **Submit:** Fichero Submit (SUB) adicional compuesto de encabezamiento y final del programa y estructura base (DECLARATION,INI,LOOP/ENDLOOP).

El **filtro** determina cómo se deben mostrar los programas en la lista de ficheros. Se puede elegir entre los siguientes **filtros**:

- **Detalle**

Los programas se visualizan como ficheros SRC y DAT (ajuste por defecto).

- **Módulos**

Los programas se muestran como módulos.

Mostrar/ocultar la **línea DEF**

- Por defecto, la **línea DEF** está oculta. Solo se pueden efectuar declaraciones en un programa una vez que la **línea DEF** sea visible.
- La **línea DEF** se muestra y oculta independientemente para cada programa abierto y seleccionado. Si está activada una vista en detalle, la **línea DEF** está visible y no es necesario mostrarla expresamente.

Abrir/cerrar **FOLD**

- Los **FOLDs** siempre están cerrados para el usuario y se pueden abrir como experto
- El experto también puede programar **FOLDs** propios
- La **sintaxis** para un FOLD es la siguiente:

```
; FOLD Nombre
Instrucciones
; ENDFOLD <Nombre>
```

Las filas ENDFOLD pueden asignarse más fácilmente si se introduce aquí el nombre del FOLD. Pueden activarse los FOLDs.

Procedimiento para activar el nivel experto y subsanar errores

Activar el nivel experto

1. Seleccionar en el menú principal **Configuración > Grupo usuario**.
2. Registrarse como **experto**: Pulsar **Iniciar sesión**. Marcar el grupo de usuario **Experto** y confirmar con **Iniciar sesión**.
3. Se solicita lo siguiente: Introducir la contraseña (predeterminada: kuka) y confirmar con **Iniciar sesión**.

Subsanar el error en el programa

1. Seleccionar un módulo defectuoso en el navegador



Fig. 2-1: Programa incorrecto

2. Seleccionar el menú **Lista de errores**

3. Se abre la indicación de errores (*nombre del programa.ERR*)
4. Seleccionar el error, se representa una descripción detallada abajo en la indicación de errores
5. En la ventana de indicación de errores pulsar la tecla **Visualizar** y saltar al programa defectuoso
6. Subsanar el error
7. Abandonar el editor y guardar

2.2 Ejercicio: Medición Tool y Base

Objetivo del ejercicio

Después de terminar con éxito este ejercicio, se dispone de la competencia necesaria para efectuar las siguientes tareas:

- Medición de la herramienta con el método de 4 puntos XYZ y el método World 5D
- Medición de la herramienta mediante la entrada de valores numéricos
- Medición Base según el método de 3 puntos

Requisitos

Las siguientes condiciones son necesarias para efectuar este ejercicio con éxito:

- Conocimientos teóricos sobre la medición de una herramienta
- Conocimientos teóricos sobre la medición de una base

Tarea

Medición garra

- Medir la garra con la ayuda de la "entrada numérica".

- Asignar el número de herramienta 3 y la denominación "Garra".
- Los datos de medición son:

X	Y	Z	A	B	C
163 mm	146 mm	146 mm	45°	0°	180°

- Datos de carga de la garra:

M	X	Y	Z	A	B	C
6,68 kg	21 mm	21 mm	98 mm	0°	0°	0°

JX	JY	JZ
0,1 kgm ²	0,4 kgm ²	0,46 kgm ²

Medición clavija

- Utilizar la clavija superior del depósito de clavijas y sujetarla manualmente en la garra.
- Medir la clavija con el método 4 puntos XYZ y el método ABC-World 5D.
- Utilizar como número de herramienta el 2 y como denominación "Clavija1".

Medición "base azul"

- La medición de la BASE número 2 se realiza con la Clavija1 y el métodos de 3 puntos. Como denominación se utiliza "base azul".

Lo que deberá saberse ahora:

1. ¿Qué ventajas ofrece la utilización de una base sobre la mesa?
-
-

2. ¿Dónde se encuentra el punto de referencia para la medición de una herramienta?

.....
.....

3. ¿Cuántos sistemas de base diferentes presenta el software de KUKA?

.....
.....

4. ¿Cuál es la diferencia entre el método ABC-World 5D y 6D?

.....
.....

5. ¿Dónde se encuentra una base sin medir (estado de suministro)?

.....
.....

2.3 Ejercicio: Navegador experto bucle sinfín

Objetivo del ejercicio Después de completar correctamente este ejercicio, se dispondrá de la competencia necesaria para efectuar las siguientes tareas:

- Creación de módulos en el nivel experto
- Combinación de módulos individuales en un programa principal
- Utilización de un bucle sinfín

Requisitos Los siguientes requisitos son necesarios para completar este ejercicio correctamente:

- Conocimientos teóricos sobre el navegador en el nivel experto
- Conocimientos teóricos sobre la utilización de subprogramas globales
- Conocimientos teóricos sobre un bucle sinfín

Formulación

1. Crear un plan de ejecución del programa (PEP) antes del inicio de la programación.
2. Crear dos módulos adecuados (cubo_dep/cubo_rec).
3. La llamada y el bucle sinfín deben tener lugar en el módulo DEPÓSITO.
4. Compruebe el programa en los modos de servicio T1, T2 y Automático.
Se deben tener en cuenta las prescripciones de seguridad enseñadas.

Lo que se debe saber tras el ejercicio:

1. ¿Cuál es la diferencia al crear un programa entre la selección de MODUL y EXPERT?

.....
.....
.....
.....
.....

2. ¿Qué se entiende por el modo de ejecución del programa ISTEP?

.....
.....
.....

3. ¿Cuál es la diferencia entre SELECCIONAR y ABRIR un programa?

.....
.....
.....

4. ¿Cuál es el aspecto de la sintaxis para un "pliegue"?

.....
.....
.....

5. ¿Qué consecuencias tiene la modificación posterior de la posición HOME?

.....
.....
.....

3 Variables y declaraciones

3.1 Gestión de datos en KRL

Trabajar con variables con KRL

Generalidades sobre variables

- En la programación de robots con KRL, una variable es en sentido amplio de la palabra, simplemente un recipiente para operandos ("valores") que aparecen en el desarrollo de un proceso del robot.
- Una variable dispone de una dirección determinada asignada en la memoria de un ordenador.
- Una variable se denomina mediante un nombre que es una palabra clave de KUKA.
- Cada variable está unida a un tipo determinado de archivos.
- La declaración del tipo de archivos es necesaria antes de la utilización.
- En KRL se distingue entre variables locales y globales.

Vida útil de las variables en KRL

- Con la vida útil se entiende el tiempo en el que la variable tiene reservado la posición de memoria.
- Las variables de duración temporal vuelven a liberar su posición de memoria al abandonar el programa o la función.
- Las variables de la lista de datos obtienen el valor actual su posición de memoria de forma permanente.

Validez de las variables en KRL

- Las variables declaradas como locales solo están disponibles y son visibles en este programa.
- Las variables globales están registradas en una lista de datos central (global).
- Las variables globales también se pueden registrar en un lista de datos local y en la declaración se les aplica la palabra clave **global**.

Tipos de datos con KRL

- un tipo de datos denomina un resumen de objetos en relación con una cantidad
- tipos de datos estándar predefinidos
- tipos de datos estándar autodefinidos
- tipos de datos de KUKA predefinidos

Utilización de variables con KRL

Convenciones sobre los nombres

- Los nombres en KRL pueden tener una longitud máxima de 24 caracteres.
- Los nombres en KRL pueden contener letras (A-Z), cifras (0-9) y los caracteres especiales "_" y "\$".
- Los nombres en KRL no deben comenzar con cifras.
- Los nombres en KRL no deben ser palabras clave.
- La utilización de mayúsculas y minúsculas es indiferente.

Tipos de datos con KRL

- Tipos de datos estándar predefinidos

Tipos de datos simples	Número entero	Número con coma flotante	Valores lógicos	Carácter individual
Palabra clave	INT	REAL	BOOL	CHAR
Rango de valores	$-2^{31} \dots (2^{31}-1)$	$\pm 1.1 \cdot 10^{-38} \dots \pm 3.4 \cdot 10^{38}$	TRUE / FALSE	Juego de caracteres ASCII
Ejemplos	-199 o 56	-0,0000123 o 3,1415	TRUE o FALSE	"A" o "q" o "7"

■ Campos / Grupo

```
Voltage[10] = 12.75
Voltage[11] = 15.59
```

- Guardar varias variables del mismo tipo de datos mediante el índice
- La inicialización o la modificación de valores se realiza mediante índice
- El tamaño máximo del campo depende de las necesidades de almacenamiento del tipo de datos

■ Tipo de datos de enumeración

```
color = #red
```

- Todos los valores del tipo de enumeración se definen en la creación con nombre (en forma de texto)
- El sistema también determina un orden
- El número máximo de elementos depende del espacio de almacenamiento

■ Tipo de datos combinado / estructura

```
Date = {day 14, month 12, year 1996}
```

- Tipo de datos combinado de componentes de diferentes tipos de datos
- Los componentes pueden estar compuestos por tipos de datos simples, pero también por estructuras
- Es posible el acceso a componentes individuales

Vida útil / validez

■ Las variables que se crean en el fichero SRC se denominan variables de duración temporal y

- no siempre se pueden mostrar
- solo son válidas en la parte del programa declarada
- vuelven a liberar su espacio de almacenamiento al alcanzar la última línea del programa (línea END)

■ Variables en el fichero DAT local

- se pueden mostrar siempre durante la ejecución del programa del fichero SRC correspondiente
- están disponibles en el fichero SRC completo, por tanto también en subprogramas locales
- también se pueden crear como variables globales
- reciben el valor actual en el fichero DAT y comienzan con el valor guardado en la nueva llamada

■ Variables en el fichero del sistema \$CONFIG.DAT

- están disponibles en todos los programas (global)
- se pueden mostrar siempre, incluso si no está activo ningún programa
- reciben el valor actual en el fichero \$CONFIG.DAT

Doble declaración de variables

- una doble declaración se crea siempre con la utilización de la misma secuencia de caracteres (nombre)
- **no** se trata de una doble declaración cuando se utiliza el mismo nombre en diferentes ficheros SRC o DAT
- las dobles declaraciones en el mismo fichero SRC y DAT no están permitidas y generan un mensaje de error
- las dobles declaraciones en el fichero SRC y DAT y en el fichero \$CONFIG.DAT están permitidas
 - durante la ejecución de la rutina de programación, en la que ha sido declarada la variable, solo se modifica el valor local y no el valor en el fichero \$CONFIG.DAT
 - durante la ejecución de la rutina de programación "ajena" solo se recurre al valor procedente del fichero \$CONFIG.DAT y se modifica

Datos de sistema de KUKA

- existen datos de sistema de KUKA como
 - tipo de datos de enumeración, como el modo de servicio (mode_op)
 - estructura, como fecha/hora (date)
- las informaciones del sistema se obtienen de las variables de sistema de KUKA
 - leen las informaciones del sistema actuales
 - modifican las configuraciones del sistema actuales
 - están predefinidas y comienzan con el carácter "\$"
 - \$DATE (hora y fecha actuales)
 - \$POS_ACT (posición actual del robot)
 - ...

3.2 Trabajar con tipos de datos simples

A continuación se explica la creación, la inicialización y la modificación de variables. En este caso solo se utilizan los tipos de datos simples.

Tipos de datos simples con KRL

- Números enteros (INT)
- Números con coma flotante (REAL)
- Valores lógicos (BOOL)
- Carácter individual (CHAR)

3.2.1 Declaración de variables

Creación de variables

Declaración de variables

- la declaración se deberá realizar siempre antes de la utilización
- cada variable deberá asignarse a un tipo de datos
- para la asignación de nombres se deberán respetar las convenciones sobre nombres
- la palabra clave para la declaración es **DECL**
- la palabra clave DECL puede suprimirse en los cuatro tipos de datos simples
- las asignaciones de valor se realizan con el puntero de ejecución en avance
- la declaración de variables se puede realizar de diferentes modo, ya que de aquí resulta la vida útil y la validez de las variables correspondientes
 - Declaración en el fichero SRC
 - Declaración en el fichero DAT local
 - Declaración en el fichero \$CONFIG.DAT
 - Declaración en el fichero DAT local con la palabra clave "global"
- creación de constantes
 - Las constantes se crean con la palabra clave **CONST**
 - Las constantes solo se pueden crear en listas de datos

Principio de la declaración de variables

Estructura del programa en el fichero SRC

- En la sección de declaración se deben declarar variables
- La sección de inicialización comienza con la primera asignación de valor, a menudo con la línea "INI"
- En la sección de instrucciones se asignan o se modifican valores

```
DEF main( )
; Sección de declaraciones
...
; Sección de inicialización
INI
...
; Sección de instrucciones
PTP HOME Vel=100% DEFAULT
...
END
```

Modificar la vista estándar

- Mostrar la línea DEF solo es posible como experto
- Necesario para acceder en los módulos delante de la línea "INI" en la sección de declaración
- La visibilidad de la línea DEF- y END también es importante para la transferencia de variables a subprogramas

Planificar la declaración de variables

- Determinar la vida útil
 - Fichero **SRC**: La variable de duración temporal "se extingue" al final de la rutina de programación
 - Fichero **DAT**: La variable se conserva después de la finalización
- Determinar la validez/disponibilidad
 - Local en el fichero **SRC**: Solo disponible en la rutina de programación en la que ha sido declarada. De este modo, la variable solo está disponible entre las líneas DEF y END locales (programa principal o subprograma local)
 - Local en el fichero **DAT**: Válida en el programa completo, por tanto también en todos los subprogramas locales
 - **\$CONFIG.DAT**: Disponible de forma global, por tanto en todas las rutinas de programación es posible un acceso de lectura y de escritura
 - Local en el fichero **DAT** como variable global: Disponible de forma global, en todas las rutinas de programación es posible un acceso de lectura y de escritura, en cuanto se aplica al fichero DAT la palabra clave PUBLIC y en la declaración se aplica adicionalmente GLOBAL
- Determinar el tipo de datos
 - BOOL: Resultados clásicos "SÍ"/"NO"
 - REAL: Resultados de operaciones del ordenador para poder evitar errores de redondeo
 - INT: Variables de conteo clásicas para bucles de conteo o contador de piezas
 - CHAR: solo un carácter
La cadena o el texto solo se puede realizar como campo CHAR
- Asignación de nombre y declaración
 - Utilizar DECL para crear una legibilidad sencilla del programa
 - Utilizar nombres de variables adecuados y autoexplicativos
 - No utilizar nombres o abreviaturas crípticos
 - Utilizar una longitud de nombre adecuada, no gastar cada vez 24 caracteres

Procedimiento en la declaración de variables con un tipo de datos simple

Crear la variable en el fichero SRC

1. Grupo usuario Experto
2. Dejar mostrar la línea DEF
3. Abrir el fichero SRC en el editor
4. Realizar la declaración de las variables

```
DEF MY_PROG ( )
DECL INT counter
DECL REAL price
DECL BOOL error
DECL CHAR symbol
INI
...
END
```

5. Cerrar el programa y guardar

Crear la variable en el fichero DAT

1. Grupo usuario Experto
2. Abrir el fichero DAT en el editor
3. Realizar la declaración de las variables

```
DEFDAT MY_PROG
EXTERNAL DECLARATIONS
DECL INT counter
DECL REAL price
DECL BOOL error
DECL CHAR symbol
...
ENDDAT
```

4. Cerrar la lista de datos y guardar

Crear variables en el fichero \$CONFIG.DAT

1. Grupo usuario Experto
2. Abrir en la carpeta SYSTEM el fichero \$CONFIG.DAT en el editor

```
DEFDAT $CONFIG
BASISTECH GLOBALS
AUTOEXT GLOBALS
USER GLOBALS
ENDDAT
```

3. Seleccionar el Fold "USER GLOBALS" y abrir con la tecla de función programable "Abrir/Cerrar Fold"
4. Realizar la declaración de las variables

```
DEFDAT $CONFIG ( )
...
;=====
; Userdefined Types
;=====
;=====
; Userdefined Externals
;=====
;=====
; Userdefined Variables
;=====
DECL INT counter
DECL REAL price
DECL BOOL error
DECL CHAR symbol
...
ENDDAT
```

5. Cerrar la lista de datos y guardar

Crear la variable global en el fichero DAT

1. Grupo usuario Experto
2. Abrir el fichero DAT en el editor
3. Ampliar lista de datos en el encabezamiento del programa con la palabra clave PUBLIC

```
DEFDAT MY_PROG PUBLIC
```

4. Realizar la declaración de las variables

```
DEFDAT MY_PROG PUBLIC
EXTERNAL DECLARATIONS
DECL GLOBAL INT counter
DECL GLOBAL REAL price
DECL GLOBAL BOOL error
DECL GLOBAL CHAR symbol
...
ENDDAT
```

5. Cerrar la lista de datos y guardar

3.2.2 Inicialización de variables con tipos de datos simples

Descripción de la inicialización con KRL

- Despues de cada declaración, la variable solo tiene reservado un espacio de almacenamiento, el valor siempre es un valor válido
- En el fichero SRC se realiza la declaración y la inicialización siempre en dos líneas separadas
- En el fichero DAT se realiza la declaración y la inicialización siempre en una línea
La constante se debe inicializar de forma inmediata en la declaración
- La sección de inicialización comienza con la primera asignación de valor

Principio de la inicialización

Inicialización de números enteros

- Inicialización como número decimal

```
value = 58
```

- Inicialización como número binario

```
value = 'B111010'
```

Binario	2^5	2^4	2^3	2^2	2^1	2^0
Dec	32	16	8	4	2	1

Cálculo: $1*32+1*16+1*8+0*4+1*2+0*1 = 58$

- Inicialización hexadecimal

```
value = 'H3A'
```

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

Cálculo: $3*16 +10 = 58$

Procedimiento en la inicialización con KRL

Declaración e inicialización en el fichero SRC

1. Abrir el fichero SRC en el editor
2. La declaración se ha realizado
3. Realizar la inicialización

```
DEF MY_PROG ( )
DECL INT counter
DECL REAL price
DECL BOOL error
DECL CHAR symbol
INI
counter = 10
price = 0.0
error = FALSE
symbol = "X"
...
END
```

4. Cerrar el programa y guardar

Declaración e inicialización en el fichero DAT

1. Abrir el fichero DAT en el editor
2. La declaración se ha realizado
3. Realizar la inicialización

```

DEFDAT MY_PROG
EXTERNAL DECLARATIONS
DECL INT counter = 10
DECL REAL price = 0.0
DECL BOOL error = FALSE
DECL CHAR symbol = "X"
...
ENDDAT

```

4. Cerrar la lista de datos y guardar

Declaración en el fichero DAT e inicialización en el fichero SRC

1. Abrir el fichero DAT en el editor
2. Realizar la declaración

```

DEFDAT MY_PROG
EXTERNAL DECLARATIONS
DECL INT counter
DECL REAL price
DECL BOOL error
DECL CHAR symbol
...
ENDDAT

```

3. Cerrar la lista de datos y guardar
4. Abrir el fichero SRC en el editor
5. Realizar la inicialización

```

DEF MY_PROG ( )
...
INI
counter = 10
price = 0.0
error = FALSE
symbol = "X"
...
END

```

6. Cerrar el programa y guardar

Declaración e inicialización de una constante

1. Abrir el fichero DAT en el editor
2. Realizar la declaración y la inicialización

```

DEFDAT MY_PROG
EXTERNAL DECLARATIONS
DECL CONST INT max_size = 99
DECL CONST REAL PI = 3.1415
...
ENDDAT

```

3. Cerrar la lista de datos y guardar

3.2.3 Manipulación de valores de variables de tipos de datos simples con KRL

Listado de las posibilidades para la modificación de valores de variables con KRL

Los valores de variables se modifican de diferente manera en las rutinas de programación (fichero SRC) y dependiendo de la tarea. A continuación se comentarán los métodos más comunes. La manipulación mediante operaciones de bit y funciones estándar es posible, pero aquí no se profundizará en ello.

Manipulación de datos mediante

- Tipos de cálculo básicos
 - (+) Suma
 - (-) Resta
 - (*) Multiplicación

- (/*I*) División
- Operaciones de comparación
 - (==) idéntico / igualdad
 - (<>) desigual
 - (>) mayor que
 - (<) menor que
 - (>=) mayor o igual que
 - (<=) menor o igual que
- Operaciones lógicas
 - (**NOT**) Inversión
 - (**AND**) Y lógico
 - (**OR**) O lógico
 - (**EXOR**) O excluyente
- Operaciones de bit
 - (**B_NOT**) Inversión por bits
 - (**B_AND**) Combinación de Y por bits
 - (**B_OR**) Combinación de O por bits
 - (**B_EXOR**) Combinación de O excluyente por bits

Funciones estándar

- Función absoluta
- Función raíz
- Función de seno y coseno
- Función de tangente
- Función de arcocoseno
- Función de arcotangente
- Varias funciones para la manipulación de cadenas

Relaciones en la manipulación de datos

Modificación de valores con la utilización del tipo de datos REAL y INT

- Redondear hacia arriba / hacia abajo

```
; Declaración
DECL INT A,B,C
DECL REAL R,S,T
; Inicialización
A = 3          ; A=3
B = 5.5        ; B=6 (a partir de x.5 redondea hacia arriba)
C = 2.25       ; C=2 (redondear por defecto)
R = 4          ; R=4.0
S = 6.5        ; S=6.5
T = C          ; T=2.0 (se utiliza el valor redondeado por defecto)
```

- Resultados de operaciones aritméticas (+;-;*)

Operandos	INT	REAL
INT	INT	REAL
REAL	REAL	REAL

```

; Declaración
DECL INT D,E
DECL REAL U,V
; Inicialización
D = 2
E = 5
U = 0.5
V = 10.6
; Sección de instrucciones (manipulación de datos)
D = D*E ; D = 2 * 5 = 10
E = E+V ; E= 5 + 10.6 = 15.6 -> redondear hacia arriba E=16
U = U*V ; U= 0.5 * 10.6 = 5.3
V = E+V ; V= 16 + 10.6 = 26.6

```

■ Resultados de operaciones aritméticas (/)

Particularidades en las operaciones aritméticas con valores enteros:

- En caso de resultados intermedios de operaciones enteras puras, se eliminarán todas las posiciones decimales
- En la asignación de valor a una variable entera, el resultado se redondeará de acuerdo con las reglas normales de cálculo

```

; Declaración
DECL INT F
DECL REAL W
; Inicialización
F = 10
W = 10.0
; Sección de instrucciones (manipulación de datos)
; INT / INT -> INT
F = F/2 ; F=5
F = 10/4 ; F=2 (10/4 = 2.5 -> cortar posición de coma)
; REAL / INT -> REAL
F = W/4 ; F=3 (10.0/4=2.5 -> redondear hacia arriba)
W = W/4 ; W=2.5

```

Operaciones de comparación

Con las operaciones de comparación se pueden formar expresiones lógicas. El resultado de una comparación siempre es del tipo de datos BOOL.

Operador / KRL	Descripción	Tipos de datos admisibles
==	idéntico/igual-dad	INT, REAL, CHAR, BOOL
<>	desigual	INT, REAL, CHAR, BOOL
>	mayor que	INT, REAL, CHAR
<	menor que	INT, REAL, CHAR
>=	mayor o igual que	INT, REAL, CHAR
<=	menor o igual que	INT, REAL, CHAR

```

; Declaración
DECL BOOL G,H
; Inicialización/sección de instrucciones
G = 10>10.1 ; G=FALSE
H = 10/3 == 3 ; H=TRUE
G = G<>H ; G=TRUE

```

Operaciones lógicas

Con las operaciones lógicas se pueden formar expresiones lógicas. El resultado de una operación de este tipo siempre es del tipo de datos BOOL.

Operaciones		NOT A	A AND B	A OR B	A EXOR B
A=TRUE	B=TRUE	FALSE	TRUE	TRUE	FAL-SE
A=TRUE	B=FALS E	FALSE	FALSE	TRUE	TRUE
A=FALSE	B=TRUE	TRUE	FALSE	TRUE	TRUE
A=FALSE	B=FALS E	TRUE	FALSE	FALSE	FAL-SE

```
; Declaración
DECL BOOL K,L,M
; Inicialización/sección de instrucciones
K = TRUE
L = NOT K ; L=FLASE
M = (K AND L) OR (K EXOR L) ; M=TRUE
L = NOT (NOT K) ; L=TRUE
```

Los operadores se ejecutan en el orden de su prioridad

Prioridad	Operador
1	NOT (B_NOT)
2	Multiplicación (*); división (/)
3	Suma (+), resta (-)
4	AND (B_AND)
5	EXOR (B_EXOR)
6	OR (B_OR)
7	cualquier comparación (==; <>; ...)

```
; Declaración
DECL BOOL X, Y
DECL INT Z
; Inicialización/sección de instrucciones
X = TRUE
Z = 4
Y = (4*Z+16 <> 32) AND X ; Y=FALSE
```

Procedimiento para la manipulación de datos

1. Especificar el tipo de datos para la variable o variables
2. Determinar la validez y la vida útil de las variables
3. Realizar la declaración de variables
4. Inicializar la variable
5. Manipular la variable en las rutinas de programación, es decir, siempre en el fichero SRC
6. Cerrar el fichero SRC y guardar

3.2.4 Ejercicio: Tipos de datos simples

Objetivo del ejercicio

Después de completar correctamente este ejercicio, se dispondrá de la competencia necesaria para efectuar las siguientes tareas:

- Utilización de tipos de datos simples
- Declaración, inicialización y utilización de variables
- Utilización correcta de variables en relación con su vida útil

Requisitos	Los siguientes requisitos son necesarios para completar este ejercicio correctamente: <ul style="list-style-type: none">■ Conocimientos teóricos sobre tipos de datos simples y su manipulación
Formulación	Ampliar los programas creados con lo siguiente: <ul style="list-style-type: none">■ Una variable cuenta el número de cubos transportados desde la última selección del programa.■ Una variable debe guardar el número absoluto de todos los cubos transportados hasta el momento.■ Una variable debe contar el peso total (en kg) de los cubos transportados. Un cubo pesa 0,329 kg.■ Una variable que en el proceso de recogida esté en TRUE, de lo contrario en FALSE.■ Una variable que contenga la letra "O" con la garra abierta y con la garra cerrada contenga la letra "G". En el estado indefinido, la variable recibe la asignación del valor "X".
	Utilizar nombres de variables y tipos de datos adecuados. También es importante declarar la variable en el lugar correcto. <ul style="list-style-type: none">■ Determinar dónde se declaran las variables.■ Ampliar el plan de ejecución del programa existente con esta nueva variable.■ Tener en cuenta las diferentes inicializaciones de las variables.■ Compruebe el programa en los modos de servicio T1, T2 y Automático. Se deben tener en cuenta las prescripciones de seguridad enseñadas.
	Lo que se debe saber tras el ejercicio:
1. ¿Cuál es la longitud máxima para un nombre de variable?
2. ¿Qué tipos de datos simples existen?
3. ¿Dónde se declaran las variables en el fichero SRC?
4. ¿Qué vida útil tiene una variable declarada en el fichero \$CONFIG.DAT?
5. Declarar un número con coma flotante con el nombre "Value" en el fichero DAT con el valor 138,74.

3.3 Grupos / campos con KRL

Descripción de campos con KRL	Los campos, o también denominados grupos, ofrecen espacio de almacenamiento para varias variables del mismo tipo de datos, que se diferencian mediante un índice
--------------------------------------	--

- El espacio de almacenamiento de los campos es finito, es decir, el tamaño máximo del campo depende de las necesidades de almacenamiento del tipo de datos
- En la declaración se deberá conocer el tamaño de campo y el tipo de datos
- El índice de inicio en KRL comienza siempre con 1
- La inicialización siempre se puede realizar individualmente
- Una inicialización en el fichero SRC también se puede realizar mediante un bucle

Dimensiones del campo

- Campo de 1 dimensiones

```
dimension1[4]= TRUE
```

- Campo de 2 dimensiones

```
dimension2[2,1]= 3.25
```

- Campo de 3 dimensiones

```
dimension1[3,4,1]= 21
```

- El campo de 4 dimensiones o superior **no** es compatible en KRL

Relaciones en la utilización de campos

La vida útil y la validez de las variables de campo es la misma que en la utilización de variables de tipo de datos simple

.

Declaración de campo

- Creación en el fichero SRC

```
DEF MY_PROG ( )
DECL BOOL error[10]
DECL REAL value[50,2]
DECL INT parts[10,10,10]
INI
...
END
```

- Creación en la lista de datos (también \$CONFIG.DAT)

```
DEFDAT MY_PROG
EXTERNAL DECLARATIONS
DECL BOOL error[10]
DECL REAL value[50,2]
DECL INT parts[10,10,10]
...
ENDDAT
```

Declarar el campo e inicializar en el fichero SRC

- Cada campo por llamada de índice individual

```
DECL BOOL error[10]
error[1]=FALSE
error[2]=FALSE
error[3]=FALSE
error[3]=FALSE
error[4]=FALSE
error[5]=FALSE
error[6]=FALSE
error[7]=FALSE
error[8]=FALSE
error[9]=FALSE
error[10]=FALSE
```

- Mediante bucles adecuados

```

DECL BOOL error[10]
DECL INT x
FOR x = 1 TO 10
error[x]=FALSE
ENDFOR

```



Tras la finalización del bucle x tiene el valor 11

Inicializar el campo en la lista de datos

- Cada campo por llamada de índice individual e indicación de espera posterior en la lista de datos

```

DEFDAT MY_PROG
EXTERNAL DECLARATIONS
DECL BOOL error[10]
error[1]=FALSE
error[2]=FALSE
error[3]=FALSE
error[4]=FALSE
error[5]=FALSE
error[6]=FALSE
error[7]=FALSE
error[8]=FALSE
error[9]=FALSE
error[10]=FALSE

```

- Declaración no permitida e inicialización en la lista de datos

```

DEFDAT MY_PROG
EXTERNAL DECLARATIONS
DECL BOOL error[10]
DECL INT size = 32
error[1]=FALSE
error[2]=FALSE
error[3]=FALSE
error[4]=FALSE
error[5]=FALSE
error[6]=FALSE
error[7]=FALSE
error[8]=FALSE
error[9]=FALSE
error[10]=FALSE

```



Genera diez mensajes de error "Paso de valor inicial no está en la sección inicial"

Declarar campos en la lista de datos e inicializar en el fichero SRC

- Si un campo se crea de este modo en la lista de datos, los datos actuales **no** se podrán consultar en la lista de datos; los valores actuales **solo** se pueden comprobar a través de la visualización de variables.

```

DEFDAT MY_PROG
EXTERNAL DECLARATIONS
DECL BOOL error[10]

```

```
DEF MY_PROG ( )
INI
Fehler[1]=FALSE
Fehler[2]=FALSE
Fehler[3]=FALSE
...
Fehler[10]=FALSE
```

o bien

```
DEF MY_PROG ( )
INI
FOR x = 1 TO 10
Fehler[x]=FALSE
ENDFOR
```

Inicialización mediante bucles

- Campo de 1 dimensiones

```
DECL INT parts[15]
DECL INT x
FOR x = 1 TO 15
parts[x] = 4
ENDFOR
```

- Campo de 2 dimensiones

```
DECL INT parts_table[10,5]
DECL INT x, y
FOR x = 1 TO 10
FOR y = 1 TO 5
parts_table[x, y] = 6
ENDFOR
ENDFOR
```

- Campo de 3 dimensiones

```
DECL INT parts_palette[5,4,3]
DECL INT x, y, z
FOR x = 1 TO 5
FOR y = 1 TO 4
FOR z = 1 TO 3
parts_palette[x, y, z] = 12
ENDFOR
ENDFOR
ENDFOR
```

Procedimiento para la utilización de grupos

1. Determinar el tipo de datos para el campo
2. Determinar la validez y la vida útil del campo
3. Realizar la declaración del campo
4. Inicializar los elementos del campo
5. Manipular el campo en las rutinas de programación, es decir, siempre en el fichero SRC
6. Cerrar el fichero SRC y guardar

```
DEF MY_PROG ( )
DECL REAL palette_size[10]
DECL INT counter
INI
; Inicialización
FOR counter = 1 TO 10
    palette_size[counter] = counter * 1.5
ENDFOR
...
; Cambiar valor individualmente
palette_size[8] = 13
...
; Comparación de valores
IF palette_size[3] > 4.2 THEN
    ...
```

3.4 Ejercicio: Campos con tipos de datos simples y bucle de conteo

Objetivo del ejercicio	Después de completar correctamente este ejercicio, se dispondrá de la competencia necesaria para efectuar las siguientes tareas:
	<ul style="list-style-type: none">■ Declarar, inicializar campos con tipos de datos simples■ Procesar elementos de campo individuales■ Trabajar con la indicación de variables (configuración/indicación)
Requisitos	Los siguientes requisitos son necesarios para completar este ejercicio correctamente: <ul style="list-style-type: none">■ Conocimientos de tipos de datos simples■ Conocimientos teóricos sobre campos■ Conocimientos teóricos sobre el bucle FOR■ Conocimientos teóricos sobre la indicación de variables
Formulación	Ampliar los programas creados con lo siguiente <ul style="list-style-type: none">■ Crear un campo de una dimensión con el tamaño 12. El contenido de los campos debe ser la letra "O" o "X".■ Antes del inicio del proceso, todos los campos deben estar ocupados con "O".■ Tras el depósito del 1º cubo, el 1º cubo deberá recibir la letra "X". Para el depósito posterior se deberá proceder del mismo modo.■ Comprobar el índice de campo actual y el contenido del campo.

Inicialización previa:

1	2	3	4	5	6	7	8	9	10	11	12
O	O	O	O	O	O	O	O	O	O	O	O

Transcurso del programa:

1	2	3	4	5	6	7	8	9	10	11	12
X	X	X	X	X	X	O	O	O	O	O	O

6 cubos depositados

Fig. 3-1

- Determinar dónde se declaran las variables.
- Ampliar el plan de ejecución del programa existente con este campo.
- Crear un grupo nuevo para la indicación de las variables de campo necesarias.
- Compruebe el programa en los modos de servicio T1, T2 y Automático. Se deben tener en cuenta las prescripciones de seguridad enseñadas.

Lo que se debe saber tras el ejercicio:

1. ¿Dónde se dimensiona el tamaño de un campo?

.....
.....

2. ¿Qué diferencia la declaración de un campo en el fichero SRC y en el fichero DAT?

.....
.....

3. ¿Qué mensaje de error aparece si se sobrepasa el índice de campo?

.....
.....

4. Declarar un campo de 3 dimensiones con el nombre de Precio del armario. El precio del armario se calcula de los componentes longitud, anchura y profundidad. ¿El surtido comprende 5 longitudes diferentes, 3 anchuras diferentes y 2 profundidades diferentes?

.....
.....

5. ¿Qué se puede activar con la tecla de función programable "Info start." en la indicación de variables?

.....
.....

3.5 Crear estructuras con KRL

Variables con varias informaciones individuales

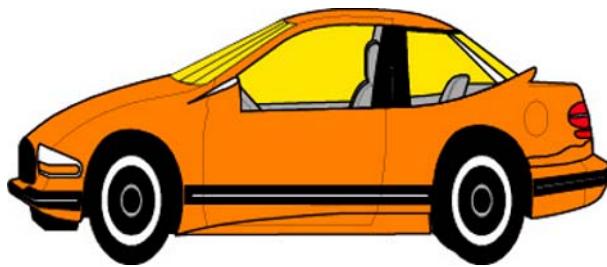


Fig. 3-2

Tipo de datos combinado: **Estructura**

- Con los campos se pueden combinar variables del mismo tipo de datos. En el mundo real, muchas veces también existen variables de tipos de datos diferentes.
- Así, un coche tiene una potencia de motor o un número de kilómetros de tipo Integer (entero). Para el precio se ofrece el tipo Real. Por el contrario, la existencia de una instalación de aire acondicionado es más bien del tipo de datos Bool.
- Todo junto describe un coche.
- Una estructura se puede definir por sí misma con la palabra clave STRUC.
- Una estructura es un conjunto de diferentes tipos de datos.

```
STRUC CAR_TYPE INT motor, REAL price, BOOL air_condition
```

- Una estructura debe estar definida previamente, y posteriormente se puede seguir utilizando.

Utilización de una estructura

Disponibilidad/definición de una estructura

- En una estructura se pueden utilizar los tipos de datos simples INT, REAL, BOOL y CHAR

```
STRUC CAR_TYPE INT motor, REAL price, BOOL air_condition
```

- En una estructura se pueden integrar campos CHAR

```
STRUC CAR_TYPE INT motor, REAL price, BOOL air_condition, CHAR car_model[15]
```

- En una estructura se pueden utilizar también estructuras conocidas, como una posición POS

```
STRUC CAR_TYPE INT motor, REAL price, BOOL air_condition, POS car_pos
```

- Despues de la definición de la estructura se debe declarar una variable de trabajo correspondiente

```
STRUC CAR_TYPE INT motor, REAL price, BOOL air_condition
DECL CAR_TYPE my_car
```

Inicialización /modificación de una estructura

- La inicialización se puede realizar mediante paréntesis
- En la inicialización mediante paréntesis solo se pueden utilizar constantes (valores fijos)
- El orden de la asignación de valores no se debe tener en cuenta

```
my_car = {motor 50, price 14999.95, air_condition = TRUE}
```

```
my_car = {price 14999.95, motor 50, air_condition = TRUE}
```

- En una estructura no se deben indicar todos los elementos de la estructura
- Una estructura se inicializa con un elemento de la estructura
- Los valores no inicializados están o se ajustan como desconocidos

```
my_car = {motor 75} ; Precio no conocido
```

- La inicialización también se puede realizar mediante el separador de puntos

```
my_car.price = 9999.0
```

- En la inicialización mediante el separador de puntos también se puede insertar variables

```
my_car.price = value_car
```

- Un elemento de la estructura se puede modificar individualmente en cualquier momento mediante el separador de puntos

```
my_car.price = 12000.0
```

Validez/vida útil

- Las estructuras creadas localmente no son válidas al alcanzar la línea END
- Las estructuras que se utilicen en varios programas deben declararse en el fichero \$CONFIG.DAT

Denominación

- No se pueden utilizar palabras clave
- Las estructuras autodefinidas deberán terminar con **TYPE** para una mejor visibilidad

KUKA trabaja mucho con las estructuras predefinidas que se encuentran guardadas en el sistema. Se pueden encontrar ejemplos en relación con las posiciones y la programación de mensajes

Estructuras predefinidas de KUKA a partir del sector de las posiciones

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

Inicialización de una estructura con una posición

- En la inicialización mediante paréntesis solo se pueden utilizar constantes (valores fijos)

```
STRUC CAR_TYPE INT motor, REAL price, BOOL air_condition, POS
car_pos
DECL CAR_TYPE my_car
my_car = {price 14999.95, motor 50, air_condition = TRUE, car_pos {X
1000, Y 500, A 0}}
```

- La inicialización también se puede realizar mediante el separador de puntos

```
my_car.price = 14999.95
my_car.car_pos = {X 1000, Y 500, A 0}
```

- En la inicialización mediante el separador de puntos también se puede insertar variables

```
my_car.price = 14999.95
my_car.car_pos.X = x_value
my_car.car_pos.Y = 750
```

Creación de una estructura

1. Definición de la estructura

```
STRUC CAR_TYPE INT motor, REAL price, BOOL air_condition
```

2. Declaración de las variables de trabajo

```
DECL CAR_TYPE my_car
```

3. Inicialización de las variables de trabajo

```
my_car = {motor 50, price 14999.95, air_condition = TRUE}
```

4. Modificación de los valores y/o comparación de valores de las variables de trabajo

```
my_car.price = 5000.0
```

```
my_car.price = value_car
```

```
IF my_car.price >= 20000.0 THEN
...
ENDIF
```

3.6 Ejercicio: Crear estructuras con KRL

Objetivo del ejercicio

Después de completar correctamente este ejercicio, se dispondrá de la competencia necesaria para efectuar las siguientes tareas:

- Creación de estructuras propias (declaración, inicialización)
- Trabajar con el separador de puntos

Requisitos

Los siguientes requisitos son necesarios para completar este ejercicio correctamente:

- Conocimientos teóricos sobre tipos de datos y estructuras simples

Formulación

Crear un programa con el nombre de estructura.

- Generar una estructura con el nombre BOX_Tipo. En esta estructura se deben poder incluir los siguientes valores característicos:
 - Longitud
 - Ancho
 - Altura
 - Estado de la pintura (sí/no)
 - Número de botellas
- Asignar a la variable CAJA la estructura BOX_Tipo e inicializar con los siguientes valores de inicio:
 - Longitud 25,2
 - Anchura 18,5
 - Altura 0,5
 - Contenido: 4 botellas
 - No pintada
- En la primera operación de procesamiento se incluyen 8 botellas más. En el 2º paso la CAJA aún se pinta. Entre el paso uno y dos se debe planificar un tiempo de espera de 3 segundos.
- Solicitar que se muestre el contenido de la variable CAJA durante el proceso.
- Determinar dónde se declaran las variables.

- Ampliar el plan de ejecución del programa existente con este campo.
- Crear un grupo nuevo para la indicación de las variables de campo necesarias.
- Compruebe el programa en los modos de servicio T1, T2 y Automático.
Se deben tener en cuenta las prescripciones de seguridad enseñadas.

Lo que se debe saber tras el ejercicio:

1. ¿Qué es una estructura?

.....
.....

2. ¿Qué estructuras predefinidas de KUKA existen?

.....
.....

3. ¿Cuál es la diferencia entre una estructura POS y una estructura E6POS?

.....
.....

4. Nombrar una estructura FRAME conocida.

.....
.....

5. ¿Para qué se necesita el separador de puntos?

.....
.....

3.7 El tipo de datos de enumeración ENUM

Forma de texto como valor de variable

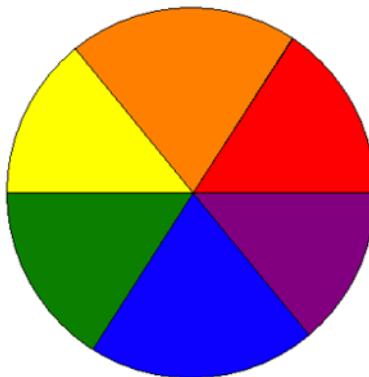


Fig. 3-3

- El tipo de datos de enumeración se compone de una cantidad limitada de constantes, como verde, amarillo o azul

```
ENUM COLOR_TYPE green, blue, red, yellow
```

- Las constantes son nombres libremente seleccionables
- Las constantes se determinan por programadores
- Un tipo de enumeración debe estar definido previamente, y posteriormente se puede seguir utilizando
- Una variable de trabajo, como color de la caja, del tipo COLOR_TYPE solo puede aceptar siempre un valor de una constante
- La asignación de valor de una constante se realiza siempre con el símbolo #

Utilización de un tipo de datos de enumeración

Disponibilidad / utilización

- Solo se pueden utilizar constantes conocidas
- Un tipo de enumeración se pueden ampliar tantas veces como sea necesario
- Un tipo de enumeración se puede utilizar individualmente

```
ENUM COLOR_TYPE green, blue, red, yellow
```

- Un tipo de enumeración se puede integrar en una estructura

```
ENUM COLOR_TYPE green, blue, red, yellow
STRUC CAR_TYPE INT motor, REAL price, COLOR_TYPE car_color
```

Validez/vida útil

- Los tipos de datos de enumeración creados localmente no son válidos al alcanzar la línea END
- Los tipos de datos de enumeración que se utilicen en varios programas deben declararse en el fichero \$CONFIG.DAT

Denominación

- Los nombres de tipos de enumeración y sus constantes deben ser auto-explicativos
- No se pueden utilizar palabras clave
- Los tipos de enumeración autodefinidos deberán terminar con **TYPE** para una mejor visibilidad

Creación de un tipo de datos de enumeración

1. Definición de las variables de las variables de enumeración y las constantes

```
ENUM LAND_TYPE de, be, cn, fr, es, br, us, ch
```

2. Declaración de las variables de trabajo

```
DECL LAND_TYPE my_land
```

3. Inicialización de las variables de trabajo

```
my_land = #be
```

4. Comparación de valores de las variables de trabajo

```
IF my_land == #es THEN  
...  
ENDIF
```

3.8 Ejercicio: Crear un tipo de enumeración con KRL

Objetivo del ejercicio

Después de completar correctamente este ejercicio, se dispondrá de la competencia necesaria para efectuar las siguientes tareas:

- Crear una variable ENUM propia (declaración)
- Trabajar con variables ENUM

Requisitos

Los siguientes requisitos son necesarios para completar este ejercicio correctamente:

- Conocimientos teóricos sobre tipos de datos y estructuras simples
- Conocimientos teóricos sobre variables ENUM sencillas

Formulación

Ampliar la estructura del programa con lo siguiente:

1. Generar una variable ENUM cuyo contenido incluya los colores rojo, amarillo, verde y azul.
2. Crear una variable de lámpara para asignarle el color azul.
3. En el segundo paso, asignar a la lámpara el color amarillo.
4. Ampliar la estructura BOX_Tipo con la unidad de color y asignar a la CAJA en principio el color rojo, seguidamente amarillo y finalmente el color verde.
5. Solicitar que se muestre el contenido de la variable CAJA durante el proceso.

Lo que se debe saber tras el ejercicio:

1. ¿Quién determina el número y el nombre de las constantes en el tipo de enumeración ENUM?
-
-

2. ¿Cuándo se utiliza el símbolo "#"?
-
-

3. ¿Qué es incorrecto en la siguiente generación?
ENUM Día_tipo lu, ma, mi, ju, vi, sá, do
-
-

4 Subprogramas y funciones

4.1 Trabajo con subprogramas locales

Definición de subprogramas locales

- Los subprogramas locales se encuentran después del programa principal y están identificados con DEF nombre_subprograma() y END

```
DEF MY_PROG( )
; Esto es el programa principal
...
END

_____

DEF LOCAL_PROG1( )
; Esto es el subprograma local 1
...
END

_____

DEF LOCAL_PROG2( )
; Esto es el subprograma local 2
...
END

_____

DEF LOCAL_PROG3( )
; Esto es el subprograma local 3
...
END
```

- Un fichero SRC puede componerse de hasta 255 subprogramas locales
- Los subprogramas locales se pueden llamar varias veces
- Además del nombre del programa es necesario el paréntesis
- Tras la ejecución de un subprograma local tiene lugar un retroceso a la siguientes instrucción después de la llamada del subprograma

Relaciones durante el trabajo con subprogramas locales

```
DEF MY_PROG( )
; Esto es el programa principal
...
LOCAL_PROG1( )
...
END

_____

DEF LOCAL_PROG1( )
...
LOCAL_PROG2( )
...
END

_____

DEF LOCAL_PROG2( )
...
END
```

- Pueden estar encajados como máximo 20 subprogramas
- Las coordenadas de puntos se guardan en la lista DAT correspondiente y están disponibles para el fichero completo

```

DEF MY_PROG( )
; Esto es el programa principal
...
PTP P1 Vel=100% PDAT1
...
END

DEF LOCAL_PROG1( )
...
; esta es la posición idéntica al programa princ
PTP P1 Vel=100% PDAT1
...
END

```

```

DEFDAT MY_PROG( )
...
DECL E6POS XP1={X 100, Z 200, Z 300 ... E6 0.0}
...
ENDDAT

```

- Mediante RETURN se puede finalizar un subprograma y provoca un retroceso en el módulo del programa llamado

```

DEF MY_PROG( )
; Esto es el programa principal
...
LOCAL_PROG1( )
...
END

DEF LOCAL_PROG1( )
...
IF $IN[12]==FALSE THEN
RETURN ; Retroceso en programa principal
ENDIF
...
END

```

Procedimiento para la creación de subprogramas locales

1. Grupo usuario Experto
2. Dejar mostrar la línea DEF
3. Abrir el fichero SRC en el editor

```

DEF MY_PROG( )
...
END

```

4. Pasar con el cursor **debajo** de la línea END
5. Asignar un nuevo encabezamiento del programa local con DEF, nombre del programa y paréntesis

```

DEF MY_PROG( )
...
END
DEF PICK_PART( )

```

6. Finalizar un subprograma nuevo con una instrucción END

```

DEF MY_PROG( )
...
END
DEF PICK_PART( )
END

```

7. Tras pulsar Return se inserta una barra trasversal entre el programa principal y el subprograma

```
DEF MY_PROG( )
...
END

-----
```

```
DEF PICK_PART( )
END
```

8. El programa principal y el subprograma se pueden seguir procesando ahora
9. Cerrar el programa y guardar

4.2 Trabajo con subprogramas globales

Definición de subprogramas globales

- Los subprogramas globales disponen de ficheros SRC y DAT independientes

```
DEF GLOBAL1( )
...
END
```

```
DEF GLOBAL2( )
...
END
```

- Los subprogramas globales se pueden llamar varias veces
- Tras la ejecución de un subprograma local tiene lugar un retroceso a la siguientes instrucción después de la llamada del subprograma

```
DEF GLOBAL1( )
...
GLOBAL2( )
...
END
```

```
DEF GLOBAL2( )
...
GLOBAL3( )
...
END
```

- Pueden estar encajados como máximo 20 subprogramas
- Las coordenadas de puntos se guardan en la lista DAT correspondiente y solo están disponibles para el programa correspondiente

```
DEF GLOBAL1( )
...
PTP P1 Vel=100% PDAT1
END
```

```
DEFDAT GLOBAL1( )
DECL E6POS XP1={X 100, Z 200, Z 300 ... E6 0.0}
ENDDAT
```

Diferentes coordenadas para P1 en Global2 ()

```
DEF GLOBAL2( )
...
PTP P1 Vel=100% PDAT1
END
```

```
DEFDAT GLOBAL2( )
DECL E6POS XP1={X 800, Z 775, Z 999 ... E6 0.0}
ENDDAT
```

- Mediante RETURN se puede finalizar un subprograma y provoca un retroceso en el módulo del programa llamado

```
DEF GLOBAL1( )
...
GLOBAL2( )
...
END
```

```
DEF GLOBAL2( )
...
IF $IN[12]==FALSE THEN
RETURN ; Retroceso en GLOBAL1( )
ENDIF
...
END
```

Procedimiento para la programación con subprogramas globales

1. Grupo usuario Experto
2. Crear un nuevo programa

```
DEF MY_PROG( )
...
END
```

3. Crear el segundo programa de nuevo

```
DEF PICK_PART( )
...
END
```

4. Abrir el fichero SRC del programa MY_PROG en el editor
5. Programar la llamada del subprograma mediante el nombre del programa y paréntesis

```
DEF MY_PROG( )
...
PICK_PART( )
...
END
```

6. Cerrar el programa y guardar

4.3 Transmitir parámetros a subprogramas

Descripción de la transferencia de parámetros

- Sintaxis

```
DEF MY_PROG( )
...
CALC (K, L)
...
END
```

```
DEF CALC(R:IN, S:OUT)
...
END
```

- Para transmitir parámetros a subprogramas existen dos posibilidades
 - como parámetros **IN**
 - como parámetros **OUT**
- La transferencia de parámetros se puede realizar tanto a subprogramas locales como globales
- Transferencia de parámetros como parámetros **IN** (*Call by value*):
 - el valor de la variable permanece invariable en el programa principal, es decir, se continúa trabajando con el valor anterior del programa principal
 - el subprograma solo puede leer el valor de la variable, pero no puede escribirlo
- Transferencia de parámetros como parámetros **OUT** (*Call by reference*):
 - el valor de la variable se modifica en el programa principal, es decir, se acepta la valencia del subprograma
 - el subprograma lee el valor, lo modifica y escribe el valor nuevo de vuelta
- Transferencia de parámetros a subprogramas locales

Principio de la transferencia de parámetros

```

DEF MY_PROG( )
DECL REAL r,s
...
CALC_1(r)
...
CALC_2(s)
...
END

_____
DEF CALC_1(num1:IN)
; Valor "r" transferido a num1 sólo para lectura
DECL REAL num1
...
END

_____
DEF CALC_2(num2:OUT)
; Valor "s" transferido a num2, modificado y reescrito
DECL REAL num2
...
END

```

■ Transferencia de parámetros a subprogramas globales

```

DEF MY_PROG( )
DECL REAL r, s
...
CALC_1(r)
...
CALC_2(s)
...
END

```

```

DEF CALC_1(num1:IN)
; Valor "r" transferido a num1 sólo para lectura
DECL REAL num1
...
END

```

```

DEF CALC_2(num2:OUT)
; Valor "s" transferido a num2, modificado y reescrito
DECL REAL num2
...
END

```

- La transferencia de valores a tipos de datos iguales siempre es posible
- Transferencia de valores a otros tipos de datos:

```

DEF MY_PROG( )
DECL DATATYPE1 value
CALC(value)
END

_____
DEF CALC(num:IN)
DECL DATATYPE2 num
...
END

```

DATATYPE 1	DATATYPE 2	Observación
BOOL	INT, REAL, CHAR	ERROR (...parámetro incompatible)
INT	REAL	Valor INT se utiliza como valor REAL
INT	CHAR	Se utiliza el carácter de la tabla ASCII
CHAR	INT	Se utiliza el valor INT de la tabla ASCII

DATATYPE 1	DATATYPE 2	Observación
CHAR	REAL	Se utiliza el valor REAL de la tabla ASCII
REAL	INT	Los valores REAL se redondean
REAL	CHAR	Los valores REAL se redondean, se utiliza el carácter de la tabla ASCII

■ Transferencia de varios parámetros

```

DEF MY_PROG( )
DECL REAL w
DECL INT a, b
...
CALLOC(w, b, a)
...
CALLOC(w, 30, a)
...
END

DEF CALC(ww:OUT, bb:IN, aa:OUT)
;1.) w <-> ww, b -> bb, a <-> aa
;2.) w <-> ww, 30 -> bb, a <-> aa
DECL REAL ww
DECL INT aa, bb
...
END

```



También es posible no transmitir valores, en caso de que en el subprograma también se calcule sin estos valores. Ejemplo: CALCULA(s,, a)

■ Transferencia de parámetros con campos

- Los campos solo se pueden transmitir completos a un campo nuevo
- Los campos solo se pueden transmitir con el parámetro OUT (Call by reference)

```

DEF MY_PROG( )
DECL CHAR name[10]
...
name="PETER"
RECHNE(name[])
...
END

DEF RECHNE(my_name[]:OUT)
; Generar campo en el subprograma siempre sin tamaño de campo
; El tamaño de campo se adapta al campo de salida
DECL CHAR my_name[]
...
END

```



Transferencia de campos **completos**: CAMPO_1D[] (1 dimensión), CAMPO_2D[,] (2 dimensiones), CAMPO_3D[,,] (3 dimensiones)

- Los elementos de campo individuales también se pueden transmitir

```
DEF MY_PROG( )
DECL CHAR name[10]
...
name="PETER"
CALC(name[1])
...
END

DEF RECHNE(symbol:IN)
; Se transfiere sólo un carácter
DECL CHAR symbol
...
END
```



En la transferencia de elementos de campo individuales solo podrá existir una variable y ningún campo como destino. Aquí solo se transmite la letra "P" al subprograma.

Procedimiento para la transferencia de parámetros

Consideraciones previas

1. Determinar los parámetros que son necesarios en el subprograma
2. Determinar el tipo de transferencia de parámetros (parámetros IN u OUT)
3. Determinar los tipos de datos iniciales y de destino (lo ideal es el mismo tipo de datos)
4. Determinar el orden de la transferencia de parámetros



A tener en cuenta: El primer parámetro enviado se escribe sobre el primer parámetro en el subprograma, el segundo en el segundo parámetro en el subprograma, etc.

1. Cargar un programa principal en el editor.
2. Declarar, inicializar y posiblemente manipular variables en el programa principal
3. Crear la llamada del subprograma con la llamada de variables
4. Cerrar el programa principal y guardar
5. Cargar el subprograma en el editor
6. Completar la línea DEF con variables y con IN/OUT
7. Declarar, inicializar y posiblemente manipular variables en el subprograma
8. Cerrar el subprograma y guardar

Ejemplo completo:

```

DEF MY_PROG( )
DECL REAL w
DECL INT a, Anzahl
w = 1.5
a = 3
b = 5
CALC(w, b, a)
; Valores actuales
; w = 3.8
; a = 13
; b = 5
END

_____  

DEF CALC(ww:OUT, bb:IN, aa:OUT)
; w <-> ww, b -> bb, a <-> aa

DECL REAL ww
DECL INT aa, bb
ww = ww + 2.3 ; ww = 1.5 + 2.3 =3.8 ->w
bb = bb + 5 ; bb = 5 + 5 = 10
aa = bb + aa ; aa = 10 + 3= 13 -> a
END

```

4.4 Ejercicio: Subprogramas con transferencia de parámetros

Objetivo del ejercicio	Después de completar correctamente este ejercicio, se dispondrá de la competencia necesaria para efectuar las siguientes tareas:
	<ul style="list-style-type: none"> ■ Tecnología de subprogramas con transferencia de parámetros
Requisitos	Los siguientes requisitos son necesarios para completar este ejercicio correctamente: <ul style="list-style-type: none"> ■ Conocimientos de tecnología de subprogramas ■ Conocimientos teóricos sobre la transferencia de parámetros ■ Conocimientos teóricos sobre funciones
Formulación	Tarea: Completar los resultados una vez ejecutados los programas.

```
DEF transmit_var( )  
  
    INT A, B, C, D, E, F  
  
    A=50  
    B=48  
    C=200  
    D=300  
    E=8  
    F=5  
  
    Calc (C, D, A, F, B, E)  
  
    ;A = ..... B = ..... C = .....  
  
    ;D = ..... E = ..... F = .....  
  
    END  
  
    ;Sub program  
DEF Calc (X1:OUT, X2:IN, X3:OUT, X4:IN, X6:IN, X5:OUT)  
INT X1, X2, X3, X4, X5, X6  
  
    ; Calculation  
X1 = X1+211 ; X1 = .....  
X2 = X2-312 ; X2 = .....  
X3 = X2 +X3 ; X3 = .....  
X4 = X4 * X5 ; X4 = .....  
X5 = X5 * 8 ; X5 = .....  
X6 = X6 - 40 ; X6 = .....  
  
END
```

Lo que se debe saber tras el ejercicio:

1. ¿Cuántos subprogramas pueden estar encajados?

.....
.....

2. ¿A qué fichero DAT accede un subprograma local?

.....
.....

3. ¿Dónde se declara una variable que debe ser conocida en el subprograma global y también en el programa principal?

.....
.....

4. ¿Con qué instrucción se transmite un valor de retorno de una función al programa principal?

.....
.....

5. ¿Cuál es la diferencia entre "parámetros IN" y "parámetros OUT" en la transferencia de parámetros?

.....
.....

4.5 Programación de funciones

Definición de funciones con KRL

- Una función es un subprograma que envía de vuelta un valor determinado al programa principal
- A menudo son necesarios determinados valores de entrada para poder calcular el valor de retorno
- En el encabezamiento de la función se determina el tipo de datos que se reescriben en el programa principal
- El valor que se va a transmitir se transmite con la instrucción RETURN(return_value)
- Existen funciones locales y globales
- Sintaxis de una función

```
DEFFCT DATATYPE NAME_FUNCTION( )
...
RETURN (return_value)
ENDFCT
```

Principio de las funciones con KRL

- El nombre del programa es al mismo tiempo el nombre de variable de un tipo de datos determinado
- Llamada de una función global

```
DEF MY_PROG( )
DECL REAL result, value
...
result = CALC(value)
...
END
```

```
DEFFCT REAL CALC(num:IN)
DECL REAL return_value, num
...
RETURN(return_value)
ENDFCT
```



La instrucción RETURN (return_value) deben tener lugar antes de la instrucción ENDFCT.

- Llamada de una función local

```
DEF MY_PROG( )
DECL REAL result, value
...
result = CALC(value)
...
END
```

```
DEFFCT REAL CALC(num:IN)
DECL REAL return_value, num
...
RETURN(return_value)
ENDFCT
```

- Utilización de parámetros IN / OUT en la transferencia de valores
 - Transferencia de valores como parámetro IN

```
DEF MY_PROG( )
DECL REAL result, value
value = 2.0
result = CALC(value)
; value = 2.0
; result = 1000.0
END
```

```
DEFFCT REAL CALC(num:IN)
DECL REAL return_value, num
num = num + 8.0
return_value = num * 100.0
RETURN(return_value)
ENDFCT
```



El valor de transferencia `value` no se modifica.

- Transferencia de valores como parámetro OUT

```
DEF MY_PROG( )
DECL REAL result, value
value = 2.0
result = CALC(value)
; value = 10,0
; result = 1000.0
END
```

```
DEFFCT REAL CALC(num:OUT)
DECL REAL return_value, num
num = num + 8.0
return_value = num * 100.0
RETURN(return_value)
ENDFCT
```



El valor de transferencia `value` se reescribe modificado.

Procedimiento para la programación de funciones

1. Determinar el valor que debe suministrar la función (tipo de datos de retorno)
2. Determinar los parámetros que son necesarios en la función (tipo de datos de transferencia)
3. Determinar el tipo de transferencia de parámetros (parámetros IN u OUT)
4. Determinar si es necesaria una función local o global
5. Cargar el programa principal en el editor
6. Declarar, inicializar y posiblemente manipular variables en el programa principal
7. Crear la llamada de función
8. Cerrar el programa principal y guardar
9. Crear la función (global o local)
10. Cargar la función en el editor
11. Completar la línea DEFFCT con el tipo de datos, las variables y con IN/OUT
12. Declarar, inicializar y manipular variables en la función
13. Crear la línea RETURN(return_value)
14. Cerrar la función y guardar

4.6 Trabajar con funciones estándar de KUKA

Listado de las funciones estándar de KUKA

Funciones matemáticas:

Descripción	Función KRL
Cantidad	ABS(x)
Raíz	SQRT(x)
Seno	SIN(x)
Coseno	COS(x)
Tangente	TAN(x)
Arcocoseno	ACOS(x)
Arco tangente	ATAN2(y,x)

Funciones para variables de cadena:

Descripción	Función KRL
Determinación de la longitud de cadena en la declaración	StrDeclLen(x)
Longitud de una variable de cadena tras de la inicialización	StrLen(x)
Borrado del contenido de variables de cadena	StrClear(x)
Ampliar la variable de cadena	StrAdd(x,y)
Comparación del contenido de variables de cadena	StrComp(x,y,z)
Copiar variable de cadena	StrCopy(x,y)

Funciones para la emisión de mensaje:

Descripción	Función KRL
Transmitir mensaje	Set_KrlMsg(a,b,c,d)
Transmitir diálogo	Set_KrlDlg(a,b,c,d)
Comprobar mensaje	Exists_KrlMsg(a)
Comprobar diálogo	Exists_KrlDlg(a,b)
Borrar mensaje	Clear_KrlMsg(a)
Lectura de la memoria intermedia de mensajes	Get_MsgBuffer(a)

Principio para la utilización de funciones estándar de KUKA

Cada función estándar se llama con parámetros de transmisión:

- con valor fijo

```
result = SQRT(16)
```

- variables de tipo de datos simple

```
result = SQRT(x)
```

- variable compuesta de campos

```
result = StrClear(Name[])
```

- variable compuesta de tipos de datos de enumeración

- variable compuesta de estructuras

- con varias variables diferentes

```
result = Set_KrlMsg(#QUIT, message_parameter, parameter[], option)
```



message_paramter, parameter[1...3] y option son estructuras pre-definidas de KUKA

Cada función necesita una variable adecuada en la que se pueda guardar el resultado de esta función:

- Las funciones matemáticas devuelven un valor REAL
- Las funciones de cadena devuelven un valor BOOL o INT

```
; Borrar un string  
result = StrClear(Name[])
```

- Las funciones de mensaje devuelven un valor BOOL o INT

```
; Borrar un mensaje (BOOL: borrado?)  
result = Clear_KrlMsg(Rueckwert)
```

5 Programación de movimientos con KRL

5.1 Programar los movimientos mediante KRL

Definición de un movimiento	Indicaciones necesarias para un movimiento <ul style="list-style-type: none"> ■ Tipo de movimiento - PTP, LIN, CIRC ■ Posición de destino y posible posición auxiliar ■ Parada exacta o posicionamiento aproximado ■ Posible distancia de aproximación ■ Velocidad - PTP (%) y movimiento de trayectoria (m/s) ■ Aceleración ■ Herramienta - TCP y carga ■ Base de trabajo ■ Herramienta guiada por robot o externa ■ Control de orientación para movimientos de trayectoria ■ Posible distancia de aproximación ■ Ángulo circular para un movimiento circular CIRC 				
Principio de la programación de movimientos	<p>Tipo de movimiento PTP</p> <ul style="list-style-type: none"> ■ PTP <i>Punto de destino <C_PTP <Posicionamiento aproximado de trayectoria>></i> <table border="1"> <thead> <tr> <th>Elemento</th> <th>Descripción</th> </tr> </thead> <tbody> <tr> <td><i>Punto de destino</i></td> <td> Tipo: POS, E6POS, AXIS, E6AXIS, FRAME El punto de destino puede indicarse de forma cartesiana o específica del eje. Las coordenadas cartesianas hacen referencia al sistema de coordenadas BASE. Si no se indican todos los componentes del punto de destino, la unidad de control adopta los valores de la posición anterior para los componentes que faltan. </td> </tr> </tbody> </table>	Elemento	Descripción	<i>Punto de destino</i>	Tipo: POS, E6POS, AXIS, E6AXIS, FRAME El punto de destino puede indicarse de forma cartesiana o específica del eje. Las coordenadas cartesianas hacen referencia al sistema de coordenadas BASE. Si no se indican todos los componentes del punto de destino, la unidad de control adopta los valores de la posición anterior para los componentes que faltan.
Elemento	Descripción				
<i>Punto de destino</i>	Tipo: POS, E6POS, AXIS, E6AXIS, FRAME El punto de destino puede indicarse de forma cartesiana o específica del eje. Las coordenadas cartesianas hacen referencia al sistema de coordenadas BASE. Si no se indican todos los componentes del punto de destino, la unidad de control adopta los valores de la posición anterior para los componentes que faltan.				

Elemento	Descripción
C_PTP	<p>Consigue que se dé un posicionamiento aproximado del punto de destino.</p> <p>Para el posicionamiento aproximado PTP-PTP es suficiente la indicación C_PTP. Para el posicionamiento aproximado PTP-CP, esto es, cuando tras el paso PTP aproximado se da un paso LIN o CIRC se debe indicar adicionalmente un <i>Posicionamiento aproximado de trayectoria</i>.</p>
Posicionamiento aproximado de trayectoria	<p>Sólo para el posicionamiento aproximado PTP-CP. Este parámetro determina cuando se iniciará como muy pronto el posicionamiento aproximado. Los datos posibles son:</p> <ul style="list-style-type: none"> ■ C_DIS Parámetros de distancia (Default): El posicionamiento aproximado comienza como muy pronto cuando la distancia al punto de destino queda por debajo del valor de \$APO.CDIS. ■ C_ORI Parámetros de orientación: El posicionamiento aproximado comienza como muy pronto cuando el ángulo de orientación dominante queda por debajo del valor de \$APO.CORI. ■ C_VEL Parámetros de velocidad: El posicionamiento aproximado comienza como muy pronto cuando la velocidad durante la fase de frenado queda por debajo del valor de \$APO.CVEL.

- El robot se aproxima a una posición del fichero DAT; la posición ha sido programada previamente mediante formulario en línea y realiza un posicionamiento aproximado a este punto P3

PTP XP3 C_PTP

- El robot se aproxima a una posición introducida
 - Específico del eje (AXIS o E6AXIS)

PTP {A1 0, A2 -80, A3 75, A4 30, A5 30, A6 110}

- Posición (con herramienta actualmente activa y base)

PTP {X 100, Y -50, Z 1500, A 0, B 0, C 90, S 3, T3 35}

- El robot se desplaza solo con la introducción de una o varias unidades

PTP {A1 30} ; sólo A1 desplazado a 30°

PTP {X 200, A 30} ; sólo en X a 200mm y A a 30°

Tipo de movimiento LIN

- LIN Punto de destino <Posicionamiento aproximado de trayectoria>

Elemento	Descripción
<i>Punto de destino</i>	<p>Tipo: POS, E6POS, FRAME</p> <p>Si no se indican todos los componentes del punto de destino, la unidad de control adopta los valores de la posición anterior para los componentes que faltan.</p> <p>Las informaciones sobre el estado y el giro dentro de un punto de destino del tipo POS o E6POS no se tienen en cuenta para los movimientos LIN (al igual que ocurre con los movimientos CIRC).</p> <p>Las coordenadas hacen referencia al sistema de coordenadas BASE.</p>
<i>Posicionamiento aproximado de trayectoria</i>	<p>Este parámetro consigue que se dé un posicionamiento aproximado del punto de destino. Además determina cuando se iniciará como muy pronto el posicionamiento aproximado. Los datos posibles son:</p> <ul style="list-style-type: none"> ■ C_DIS Parámetros de distancia: El posicionamiento aproximado comienza como muy pronto cuando la distancia al punto de destino queda por debajo del valor de \$APO.CDIS. ■ C_ORI Parámetros de orientación: El posicionamiento aproximado comienza como muy pronto cuando el ángulo de orientación dominante queda por debajo del valor de \$APO.CORI. ■ C_VEL Parámetros de velocidad: El posicionamiento aproximado comienza como muy pronto cuando la velocidad durante la fase de frenado queda por debajo del valor de \$APO.CVEL.

- El robot se desplaza hasta una posición calculada y realiza un posicionamiento aproximado al punto DEPÓSITO[4]

LIN ABLAGE[4] C_DIS

Tipo de movimiento CIRC

- CIRC *Punto auxiliar, punto de destino<, CA Ángulo circular> <Posicionamiento aproximado de trayectoria>*

Elemento	Descripción
<i>Punto auxiliar</i>	<p>Tipo: POS, E6POS, FRAME</p> <p>Si no se indican todos los componentes del punto auxiliar, la unidad de control adopta los valores de la posición anterior para los componentes que faltan.</p> <p>Por principio, el ángulo de orientación y las informaciones sobre el estado y el giro dentro de un punto auxiliar no se tienen en cuenta.</p> <p>El punto auxiliar no puede ser aproximado. Siempre se ejecutará de forma exacta.</p> <p>Las coordenadas hacen referencia al sistema de coordenadas BASE.</p>
<i>Punto de destino</i>	<p>Tipo: POS, E6POS, FRAME</p> <p>Si no se indican todos los componentes del punto de destino, la unidad de control adopta los valores de la posición anterior para los componentes que faltan.</p> <p>Las informaciones sobre el estado y el giro dentro de un punto de destino del tipo POS o E6POS no se tienen en cuenta para los movimientos CIRC (al igual que ocurre con los movimientos LIN).</p> <p>Las coordenadas hacen referencia al sistema de coordenadas BASE.</p>
<i>Ángulo circular</i>	<p>Indica el ángulo total del movimiento circular. Permite prolongar el movimiento más allá del punto de destino programado o acortarlo. Esto significa que el punto de destino real no incluirá el punto de destino programado.</p> <p>Unidad: Grados. Ninguna limitación; se puede programar en particular un ángulo circular de más de 360°.</p> <ul style="list-style-type: none"> ■ Ángulo circular positivo: La trayectoria circular se desplaza en dirección punto inicial > punto auxiliar > punto de destino. ■ Ángulo circular negativo: La trayectoria circular se desplaza con dirección punto inicial > punto de destino > punto auxiliar.
<i>Posicionamiento aproximado de trayectoria</i>	<p>Este parámetro consigue que se dé un posicionamiento aproximado del punto de destino. Además determina cuando se iniciará como muy pronto el posicionamiento aproximado. Los datos posibles son:</p> <ul style="list-style-type: none"> ■ C_DIS Parámetros de distancia: El posicionamiento aproximado comienza como muy pronto cuando la distancia al punto de destino queda por debajo del valor de \$APO.CDIS. ■ C_ORI Parámetros de orientación: El posicionamiento aproximado comienza como muy pronto cuando el ángulo de orientación dominante queda por debajo del valor de \$APO.CORI. ■ C_VEL Parámetros de velocidad: El posicionamiento aproximado comienza como muy pronto cuando la velocidad durante la fase de frenado queda por debajo del valor de \$APO.CVEL.

- El robot se aproxima a las posiciones del fichero DAT; las posiciones han sido programadas previamente mediante formulario en línea y se desplaza con un ángulo circular de 190°

CIRC XP3, XP4, CA 190

- Ángulo circular CA



En el punto de destino real se adopta la orientación que se programó en el punto de destino programado.

- Ángulo circular positivo (CA>0): El circuito se desplaza en el sentido programado: Punto inicial - punto auxiliar - punto de destino

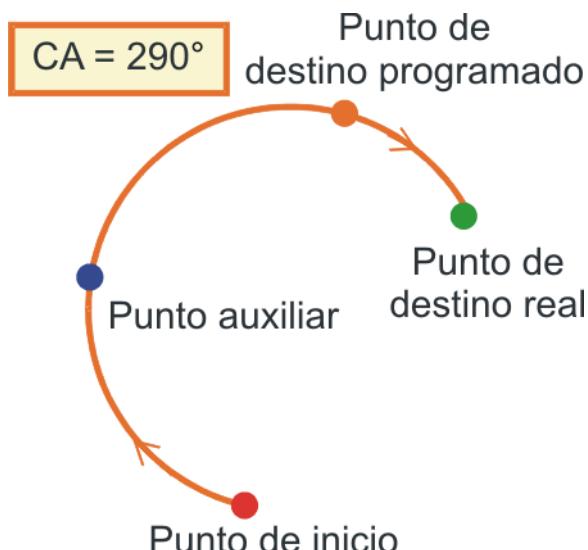


Fig. 5-1: Ángulo circular CA = +290°

- Ángulo circular negativo (CA<0): El circuito se desplaza en sentido contrario al sentido programado: Punto inicial - punto destino - punto de auxiliar



Fig. 5-2: Ángulo circular CA = -290°

Función de los parámetros de movimiento

Preajustes para la programación de movimientos

- Se pueden aplicar ajustes existentes:

- de la ejecución de la líneaINI
- del último formulario en línea
- de los últimos ajustes de las variables de sistema correspondientes
- Modificar o inicializar las variables de sistema correspondientes

Variables de sistema de los parámetros de movimiento

- **Herramienta:** \$TOOL y \$LOAD
 - Activación del TCP medido

```
$TOOL = tool_data[x] ; x = 1...16
```

- Activación de los datos de carga correspondientes

```
$LOAD = load_data[x] ; x = 1...16
```

- **Base de referencia / base de trabajo:** \$BASE

- Activación de la base medida

```
$BASE = base_data[x] ; x = 1...16
```

- **Herramienta guiada por robot o externa:** \$IPO_MODE

- Herramienta guiada por robot

```
$IPO_MODE = #BASE
```

- Herramienta externa

```
$IPO_MODE = #TCP
```

- **Velocidad**

- Con movimiento PTP

```
$VEL_AXIS[x] ; x=1...8 para cada eje
```

- Con movimientos de trayectoria LIN o CIRC

```
$VEL.CP = 2.0 ; [m/s] Velocidad de trayectoria
```

```
$VEL.ORI1 = 150 ; [°/s] Velocidad de basculamiento
```

```
$VEL.ORI2 = 200 ; [°/s] Velocidad de rotación
```



La dirección de avance de la herramienta es en la mayoría de los casos el eje X. La velocidad de giro es el giro alrededor de este eje X con ángulo C. En la velocidad de basculamiento se gira alrededor de los otros dos ángulos (A y B).

- **Aceleración**

- Con movimiento PTP

```
$ACC_AXIS[x] ; x=1...8 para cada eje
```

- Con movimientos de trayectoria LIN o CIRC

```
$ACC.CP = 2.0 ; [m/s] Aceleración de trayectoria
```

```
$ACC.ORI1 = 150 ; [°/s] Aceleración de basculamiento
```

```
$ACC.ORI2 = 200 ; [°/s] Aceleración de rotación
```

- **Distancia de aproximación**

- Solo con movimiento PTP: **C_PTP**

```
PTP_XP3_C_PTP  
$APO_CPTP = 50 ; Valor de posicionamiento aproximado en [%] con C_PTP
```

- Con movimientos de trayectoria LIN, CIRC y con PTP: **C_DIS**

El alejamiento respecto del punto de destino debe quedar por debajo del valor \$APO.CDIS

```
PTP XP3 C_DIS
LIN XP4 C_DIS
$APO.CDIS = 250.0 ; [mm] Distancia
```

- Con movimientos de trayectoria LIN, CIRC: **C_ORI**
El ángulo de orientación dominante debe quedar por debajo del valor \$APO.CORI

```
LIN XP4 C_ORI
$APO.CORI = 50.0 ; [°] Ángulo
```

- Con movimientos de trayectoria LIN, CIRC: **C_VEL**
La velocidad en la fase de frenado hacia el punto de destino debe quedar por debajo del valor \$APO.CVEL

```
LIN XP4 C_VEL
$APO.CVEL = 75.0 ; [%] Porcentaje
```

- **Control de la orientación:** Solo con LIN y CIRC
 - Con LIN y CIRC: **\$ORI_TYPE**

```
$ORI_TYPE = #CONSTANT
```

Durante el movimiento de trayectoria, la orientación se mantiene constante. Para el punto final se ignora la orientación programada.

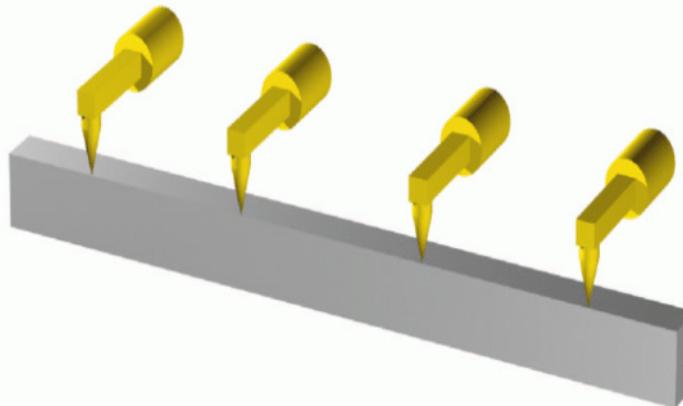


Fig. 5-3: Control orientación constante

```
$ORI_TYPE = #VAR
```

Durante el movimiento de trayectoria se transforma la orientación continuamente hacia la orientación del punto de destino.

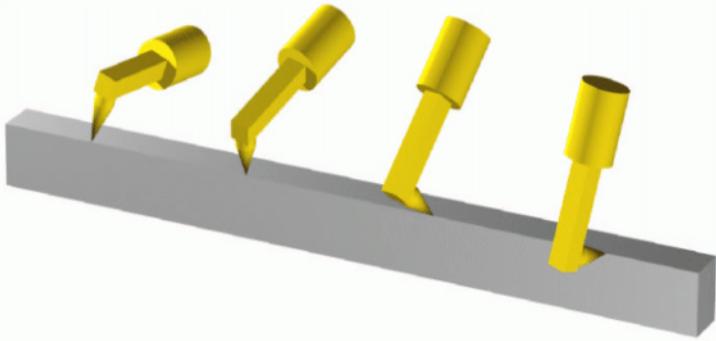


Fig. 5-4: Estándar o PTP manual

\$ORI_TYPE = #JOINT

Durante el movimiento de trayectoria, la orientación de la herramienta es modificada ininterrumpidamente desde la posición inicial hasta la posición final. Ello se logra mediante el desplazamiento lineal del eje axial de la muñeca. Esta opción permite evitar los problemas que se derivan de la singularidad de la muñeca, ya que no implica ningún control de orientación mediante giro o basculación sobre la dirección de avance de la herramienta.

■ Solo con CIRC: **\$CIRC_TYPE**



La variable **\$CIRC_TYPE** **carece de significado** cuando se efectúa un desplazamiento lineal del ángulo del eje de la muñeca con **\$ORI_TYPE = #JOINT**.

\$CIRC_TYPE = #PATH

Control de orientación referido a la trayectoria durante el movimiento circular

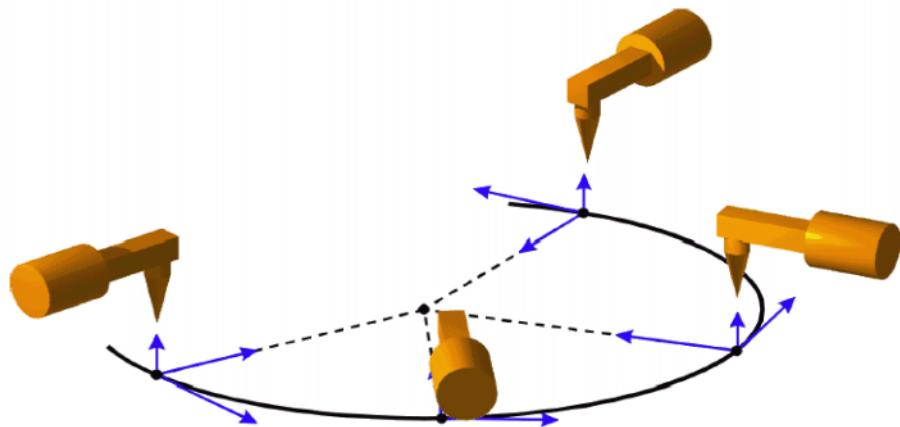


Fig. 5-5: Orientación constante, referida a la trayectoria

\$CIRC_TYPE = #BASE

Control de orientación referido al espacio durante el movimiento circular

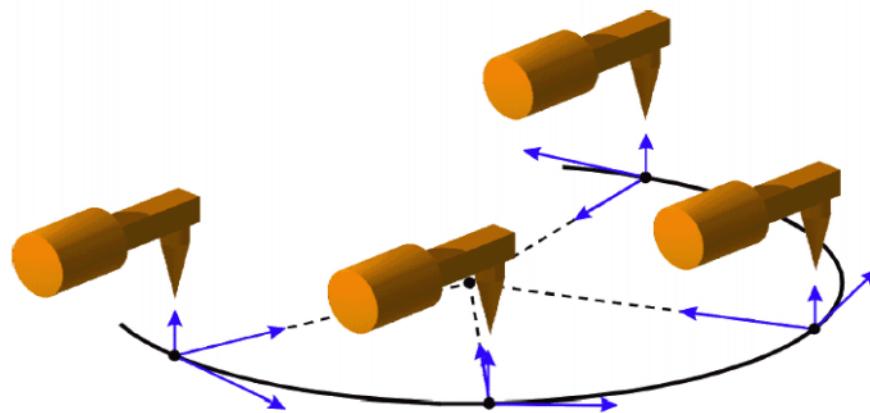


Fig. 5-6: Orientación constante, referida a la base

Procedimiento para programar movimientos mediante KRL

1. Cargar el programa en el editor como experto mediante el pulsador **Abrir**
2. Comprobar los ajustes predefinidos para la programación de movimientos y aceptar o inicializar de nuevo:
 - Herramienta (\$TOOL y \$LOAD)
 - Ajustes base (\$BASE)
 - Herramienta guiada por robot o externa (\$IPO_MODE)
 - Velocidad
 - Aceleración
 - Posible distancia de aproximación
 - Posible control de la orientación
3. Crear la instrucción de movimiento, compuesta de:
 - Tipo de movimiento (PTP, LIN, CIRC)
 - Punto de destino (con CIRC también punto auxiliar)
 - Para CIRC posible ángulo circular (CA)
 - Activar el posicionamiento aproximado (C_PTP, C_DIS, C_ORI, C_VEL)
4. En caso de nuevo movimiento de vuelta al punto 3
5. Cerrar el editor y guardar

5.2 Programar los movimientos relativos mediante KRL

Descripción

- Movimiento absoluto

```
PTP {A3 45}
```

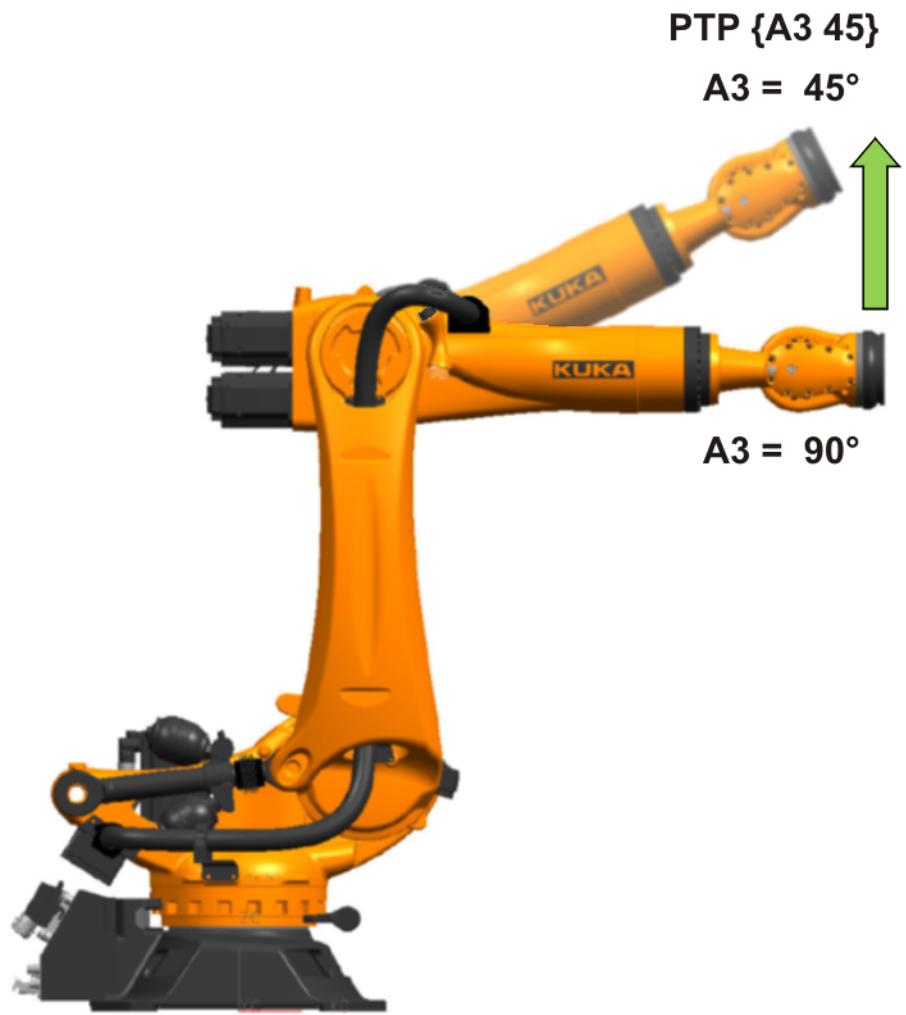


Fig. 5-7: Movimiento absoluto del eje A3

Aproximación de la posición de destino mediante valores absolutos. Aquí se posiciona el eje A3 a 45°.

- Movimiento relativo

```
PTP_REL {A3 45}
```

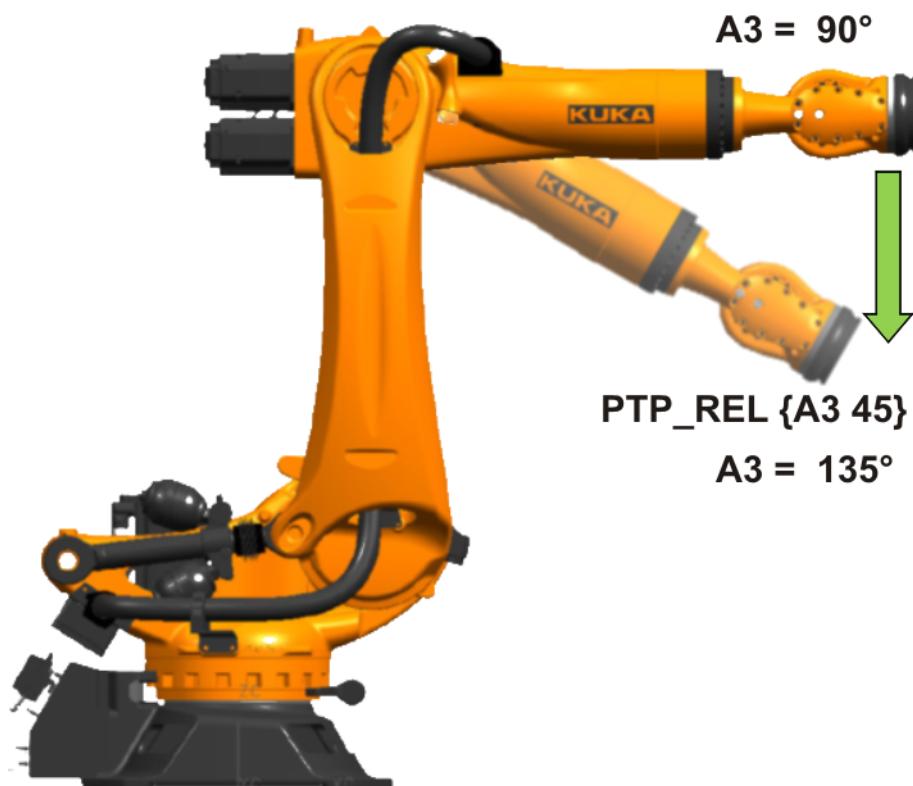


Fig. 5-8: Movimiento relativo del eje A3

Aproximación de la posición de destino, mediante el avance desde la posición actual de acuerdo con el valor indicado. Aquí el eje A3 se avanza 45°.

Existen movimientos relativos para:

- Movimiento PTP
- Movimiento LIN
- Movimiento CIRC

Principio de un movimiento relativo



La instrucción REL siempre hace referencia a la posición actual del robot. Si una instrucción REL se interrumpe, el robot por tanto efectúa, desde la posición de interrupción, todo el movimiento REL completo otra vez.

Movimiento relativo PTP_REL

- **PTP_REL** *Punto de destino <C_PTP <Posicionamiento aproximado de trayectoria>>*

Elemento	Descripción
<i>Punto de destino</i>	<p>Tipo: POS, E6POS, AXIS, E6AXIS</p> <p>El punto de destino puede indicarse de forma cartesiana o específica del eje. La unidad de control interpreta las coordenadas en relación a la posición actual. Las coordenadas cartesianas hacen referencia al sistema de coordenadas BASE.</p> <p>Si no se indican todos los componentes del punto de destino, la unidad de control pondrá a 0 los componentes que falten. Esto quiere decir que los valores absolutos de estos componentes no variarán.</p>
C_PTP	<p>Consigue que se dé un posicionamiento aproximado del punto de destino.</p> <p>Para el posicionamiento aproximado PTP-PTP es suficiente la indicación C_PTP. Para el posicionamiento aproximado PTP-CP, esto es, cuando tras el paso PTP aproximado se da un paso LIN o CIRC se debe indicar adicionalmente un <i>Posicionamiento aproximado de trayectoria</i>.</p>
<i>Posicionamiento aproximado de trayectoria</i>	<p>Sólo para el posicionamiento aproximado PTP-CP. Este parámetro determina cuando se iniciará como muy pronto el posicionamiento aproximado. Los datos posibles son:</p> <ul style="list-style-type: none"> ■ C_DIS Parámetros de distancia (Default): El posicionamiento aproximado comienza como muy pronto cuando la distancia al punto de destino queda por debajo del valor de \$APO.CDIS. ■ C_ORI Parámetros de orientación: El posicionamiento aproximado comienza como muy pronto cuando el ángulo de orientación dominante queda por debajo del valor de \$APO.CORI. ■ C_VEL Parámetros de velocidad: El posicionamiento aproximado comienza como muy pronto cuando la velocidad durante la fase de frenado queda por debajo del valor de \$APO.CVEL.

- El eje 2 se mueve 30 grados en dirección negativa. El resto de ejes no se mueve.

```
PTP_REL {A2 -30}
```

- El TCP se desplaza desde la posición actual 100 mm en dirección X y 200 mm en dirección Z negativa. Y, A, B, C y S permanecen constantes. T se calcula en el trayecto más corto.

```
PTP_REL {X 100,Z -200}
```

Movimiento relativo LIN_REL

- LIN_REL *Punto de destino <Posicionamiento aproximado de trayectoria>*
<#BASE | #TOOL>

Elemento	Descripción
<i>Punto de destino</i>	<p>Tipo: POS, E6POS, FRAME</p> <p>El punto de destino debe indicarse en coordenadas cartesianas. La unidad de control interpreta las coordenadas en relación a la posición actual. Las coordenadas pueden hacer referencia a los sistemas de coordenadas BASE o TOOL.</p> <p>Si no se indican todos los componentes del punto de destino, la unidad de control pondrá automáticamente a 0 los componentes que falten. Esto quiere decir que los valores absolutos de estos componentes no variarán.</p> <p>En los movimientos lineales LIN no se tendrán en cuenta los datos de estado y giro dentro de un punto de destino de tipo POS o E6POS.</p>
<i>Posicionamiento aproximado de trayectoria</i>	<p>Este parámetro consigue que se dé un posicionamiento aproximado del punto de destino. Además determina cuando se iniciará como muy pronto el posicionamiento aproximado. Los datos posibles son:</p> <ul style="list-style-type: none"> ■ C_DIS Parámetros de distancia: El posicionamiento aproximado comienza como muy pronto cuando la distancia al punto de destino queda por debajo del valor de \$APO.CDIS. ■ C_ORI Parámetros de orientación: El posicionamiento aproximado comienza como muy pronto cuando el ángulo de orientación dominante queda por debajo del valor de \$APO.CORI. ■ C_VEL Parámetros de velocidad: El posicionamiento aproximado comienza como muy pronto cuando la velocidad durante la fase de frenado queda por debajo del valor de \$APO.CVEL.
#BASE, #TOOL	<ul style="list-style-type: none"> ■ #BASE Ajuste predeterminado. Las coordenadas del punto de destino hacen referencia al sistema de coordenadas BASE. ■ #TOOL Las coordenadas del punto de destino hacen referencia al sistema de coordenadas TOOL. <p>La indicación #BASE o #TOOL sólo hace referencia a la instrucción LIN_REL a la cual pertenece. No repercute en las instrucciones subsiguientes.</p>

- En el sistema de coordenadas BASE, el TCP se desplaza de la posición actual 100 mm en dirección X y 200 mm en dirección Z negativa. Y, A, B, C y S permanecen constantes. T se obtiene con el movimiento.

LIN_REL {X 100,Z -200} ; El ajuste por defecto es #BASE

- En el sistema de coordenadas TOOL, el TCP se desplaza de la posición actual 100 mm en dirección X negativa. Y, Z, A, B, C y S permanecen constantes. T se obtiene con el movimiento.

Este ejemplo es apropiado para mover hacia atrás la herramienta en la dirección de avance. La condición necesaria es que la dirección de avance de la herramienta haya sido medida en dirección X.

```
LIN_REL {X -100} #TOOL
```

Movimiento relativo CIRC_REL

- CIRC_REL *Punto auxiliar, punto de destino<, CA Ángulo circular> <Posicionamiento aproximado de trayectoria>*

Elemento	Descripción
<i>Punto auxiliar</i>	<p>Tipo: POS, E6POS, FRAME</p> <p>El punto auxiliar debe indicarse con coordenadas cartesianas. La unidad de control interpreta las coordenadas en relación a la posición actual. Las coordenadas hacen referencia al sistema de coordenadas BASE.</p> <p>Si se indican \$ORI_TYPE, estado y/o giro se ignorarán estos datos.</p> <p>Si no se indican todos los componentes del punto auxiliar la unidad de control pondrá a 0 los componentes que falten. Esto quiere decir que los valores absolutos de estos componentes no variarán.</p> <p>No se tendrán en cuenta ni ángulo de orientación, ni los datos de estado y giro dentro de un punto auxiliar.</p> <p>El punto auxiliar no puede ser aproximado. Siempre se ejecutará de forma exacta.</p>
<i>Punto de destino</i>	<p>Tipo: POS, E6POS, FRAME</p> <p>El punto de destino debe indicarse en coordenadas cartesianas. La unidad de control interpreta las coordenadas en relación a la posición actual. Las coordenadas hacen referencia al sistema de coordenadas BASE.</p> <p>Si no se indican todos los componentes del punto de destino, la unidad de control pondrá a 0 los componentes que falten. Esto quiere decir que los valores absolutos de estos componentes no variarán.</p> <p>No se tendrán en cuenta los datos de estado y giro dentro de un punto de destino de tipo POS o E6POS.</p>

Elemento	Descripción
Ángulo circular	<p>Indica el ángulo total del movimiento circular. Permite prolongar el movimiento más allá del punto de destino programado o acortarlo. Esto significa que el punto de destino real no incluirá el punto de destino programado.</p> <p>Unidad: Grados. Ninguna limitación; se puede programar en particular un ángulo circular de > 360°.</p> <ul style="list-style-type: none"> ■ Ángulo circular positivo: La trayectoria circular se desplaza en dirección punto inicial → punto auxiliar → punto de destino. ■ Ángulo circular negativo: La trayectoria circular se desplaza con dirección punto inicial → punto de destino → punto auxiliar.
Posicionamiento aproximado de trayectoria	<p>Este parámetro consigue que se dé un posicionamiento aproximado del punto de destino. Además determina cuando se iniciará como muy pronto el posicionamiento aproximado. Los datos posibles son:</p> <ul style="list-style-type: none"> ■ C_DIS Parámetros de distancia: El posicionamiento aproximado comienza como muy pronto cuando la distancia al punto de destino queda por debajo del valor de \$APO.CDIS. ■ C_ORI Parámetros de orientación: El posicionamiento aproximado comienza como muy pronto cuando el ángulo de orientación dominante queda por debajo del valor de \$APO.CORI. ■ C_VEL Parámetros de velocidad: El posicionamiento aproximado comienza como muy pronto cuando la velocidad durante la fase de frenado queda por debajo del valor de \$APO.CVEL.

- El punto de destino del movimiento circular se determina mediante un ángulo circular de 500°. Se da un posicionamiento aproximado del punto de destino.

```
CIRC_REL {X 100,Y 30,Z -20}, {Y 50},CA 500 C_VEL
```

Procedimiento para programar movimientos mediante KRL

1. Cargar el programa en el editor como experto mediante el pulsador **Abrir**
2. Comprobar los ajustes predefinidos para la programación de movimientos y aceptar o inicializar de nuevo:
 - Herramienta (\$TOOL y \$LOAD)
 - Ajustes base (\$BASE)
 - Herramienta guiada por robot o externa (\$IPO_MODE)
 - Velocidad
 - Aceleración
 - Posible distancia de aproximación
 - Posible control de la orientación
3. Crear la instrucción de movimiento, compuesta de:
 - Tipo de movimiento (PTP_REL, LIN_REL, CIRC_REL)
 - Punto de destino (con CIRC también punto auxiliar)
 - Para LIN elegir el sistema de referencia (#BASE o #TOOL)
 - Para CIRC posible ángulo circular (CA)

- Activar el posicionamiento aproximado (C_PTP, C_DIS, C_ORI, C_VEL)
- 4. En caso de nuevo movimiento de vuelta al punto 3
- 5. Cerrar el editor y guardar

5.3 Calcular o manipular posiciones de robot

Descripción	<p>Posiciones de destino del robot</p> <ul style="list-style-type: none"> ■ se almacenan en las siguientes estructuras: ■ AXIS / E6AXIS - ángulo del eje (A1...A6 y posiblemente E1...E6) ■ POS / E6POS - posición (X, Y, Z), orientación (A, B, C) y Status y Turn (S, T) ■ FRAME - solo posición (X, Y, Z), orientación (A, B, C) <ul style="list-style-type: none"> ■ se pueden manipular posiciones existentes del fichero DAT ■ las unidades individuales de posiciones existentes se pueden modificar de forma precisa mediante el separador de puntos
Principio	<p>⚠ ATENCIÓN En el cálculo es importante tener en cuenta los ajustes TOOL y BASE correctos y activarlos a continuación en la programación de movimientos. En caso de incumplimiento pueden producirse movimientos inesperados y colisiones.</p>
	<p>Variables de sistema importantes</p> <ul style="list-style-type: none"> ■ \$POS_ACT: Posición actual del robot. La variable (E6POS) describe la posición teórica del TCP referida al sistema de coordenadas BASE. ■ \$AXIS_ACT: Posición actual del robot específica del eje (valor teórico). La variable (E6AXIS) contiene el ángulo del eje o la posición del eje actual. <p>Calcular la posición de destino absoluta</p> <ul style="list-style-type: none"> ■ Modificar una vez la posición del fichero DAT <pre>XP1.x = 450 ; Valor X nuevo 450mm XP1.z = 30*distance ; Se calcula el nuevo valor Z PTP XP1</pre> <ul style="list-style-type: none"> ■ Modificar en cada ejecución la posición del fichero DAT <pre>; Valor X desplazado cada vez por 450mm XP2.x = XP2.x + 450 PTP XP2</pre> <ul style="list-style-type: none"> ■ La posición se acepta y se guarda en una variable <pre>myposition = XP3 myposition.x = myposition.x + 100 ; Se añaden 100mm al valor x myposition.z = 10*distance ; Calcular el nuevo valor Z myposition.t = 35 ; Fijar valor Turn PTP XP3 ; Posición no cambiada PTP myposition ; Posición calculada</pre>

Procedimiento

1. Cargar el programa en el editor como experto mediante el pulsador **Abrir**
2. Calcular/manipular la posición. Guardar temporalmente los nuevos valores calculados en una nueva variable
3. Comprobar los ajustes predefinidos para la programación de movimientos y aceptar o inicializar de nuevo:
 - Herramienta (\$TOOL y \$LOAD)
 - Ajustes base (\$BASE)
 - Herramienta guiada por robot o externa (\$IPO_MODE)
 - Velocidad

- Aceleración
 - Posible distancia de aproximación
 - posible control de la orientación
4. Crear la instrucción de movimiento, compuesta de:
 - Tipo de movimiento (PTP, LIN, CIRC)
 - Punto de destino (con CIRC también punto auxiliar)
 - para CIRC posible ángulo circular (CA)
 - Activar el posicionamiento aproximado (C_PTP, C_DIS, C_ORI, C_VEL)
 5. en caso de nuevo movimiento de vuelta al punto 3
 6. Cerrar el editor y guardar

5.4 Modificar de forma adecuada bits de Status y Turn

Descripción

- Los valores de posición (X, Y, Z) y de orientación (A, B, C) del TCP no son suficientes para determinar únicamente la posición del robot, ya que con el mismo TCP son posibles varias posiciones del eje. Status y Turn sirven para determinar una posición única de varias posiciones del eje posibles.

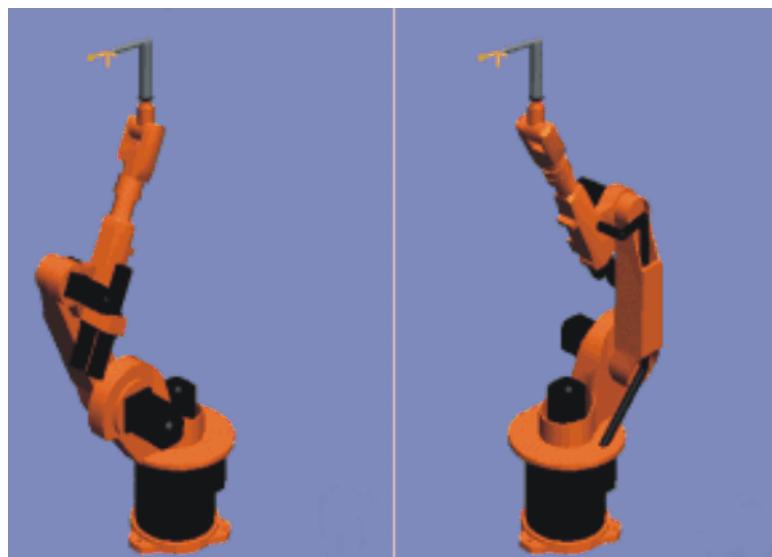


Fig. 5-9: Ejemplo: TCP igual, posición de eje diferente

- Status (S) y Turn (T) forman parte de los tipos de datos POS y E6POS:

```
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
```

- La unidad de control del robot considera los valores de Status y Turn programados **solo en caso de movimientos PTP**. Se ignoran para movimientos CP.
- Por este motivo, la primera instrucción de movimiento en un programa KRL debe ser una de las siguientes instrucciones para que se determine una posición de salida única para el robot:
 - Una instrucción PTP completa del tipo POS o E6POS
 - O bien una instrucción PTP completa del tipo AXIS o E6AXIS
"Completa" significa que se deben indicar todos los componentes del punto de destino. La posición HOME por defecto siempre es una instrucción PTP completa.
- En las demás instrucciones se pueden omitir Status y Turn:

- La unidad de control del robot conserva un valor de Status anterior.
- El valor Turn resulta con el movimiento CP procedente de la trayectoria.
- Para los movimientos PTP, la unidad de control del robot selecciona el valor Turn que ofrezca la carrera más corta posible (es decir, no se infringe ningún interruptor de final de carrera de software y al mismo tiempo el que esté más próximo al ángulo de inicio).

Función**STATUS (Estado)**

- La indicación de Status (estado) evita ambigüedades en la posición de eje.
- **Bit 0:** indica la posición del punto de intersección de los ejes de la muñeca (A4, A5, A6).

Posición	Valor
Área por encima de la cabeza El robot se encuentra en el área por encima de la cabeza, cuando el valor x del punto de intersección de los ejes de la muñeca es negativo referido al sistema de coordenadas A1.	Bit 0 = 1
Área de la base El robot se encuentra en el área de base, cuando el valor x del punto de intersección de los ejes de la muñeca es positivo referido al sistema de coordenadas A1.	Bit 0 = 0

El sistema de coordenadas A1 es idéntico al sistema de coordenadas \$ROBROOT, cuando el eje 1 está en 0°. En los valores distintos de 0° se mueve junto con el eje 1.

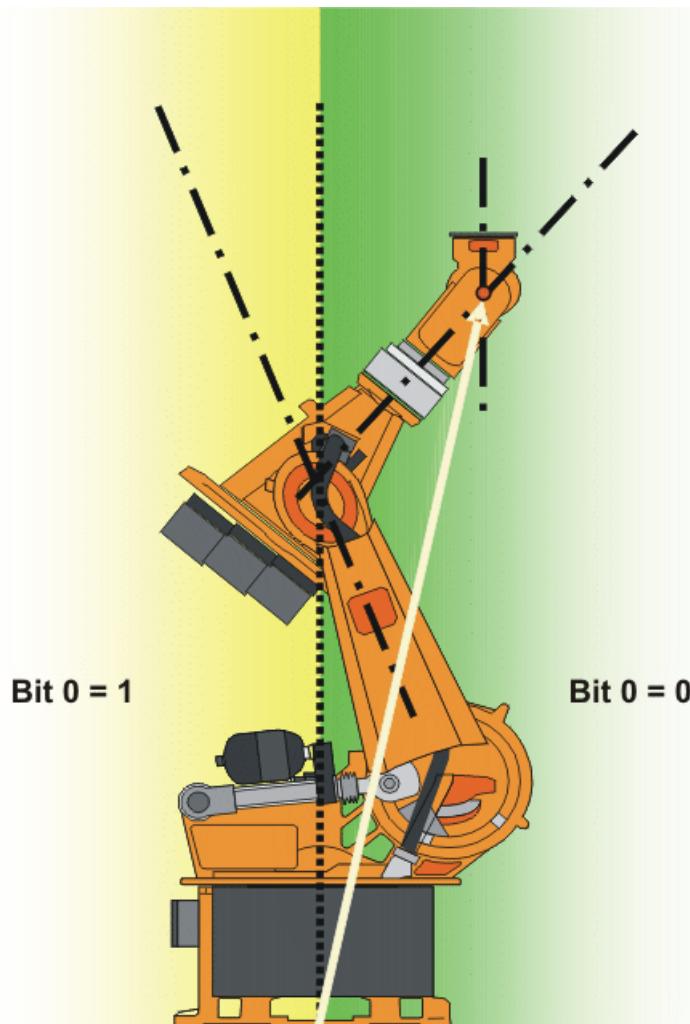


Fig. 5-10: Ejemplo: El punto de intersección de los ejes de la muñeca (punto rojo) se encuentra en el área de base.

- **Bit 1:** indica la posición del eje 3. El ángulo en el que se modifica el valor del bit 1 depende del tipo de robot.

Para los robots cuyos ejes 3 y 4 se interseccionan, se aplica lo siguiente:

Posición	Valor
$A3 \geq 0^\circ$	Bit 1 = 1
$A3 < 0^\circ$	Bit 1 = 0

En los robots con un offset entre el eje 3 y el eje 4, el ángulo en el que se modifica el valor del bit 1, depende de la magnitud de este offset.

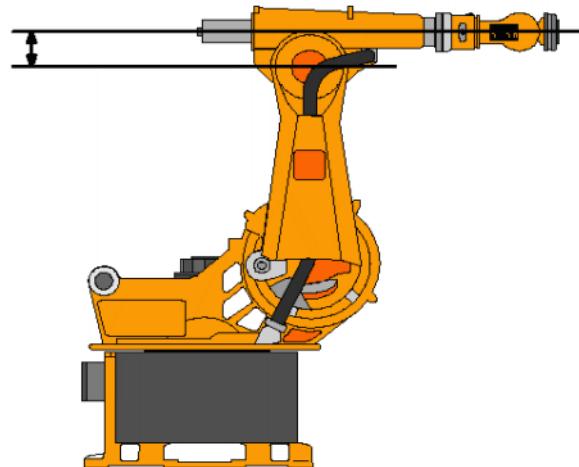


Fig. 5-11: Offset entre A3 y A4: Ejemplo KR 30

- **Bit 2:** indica la posición del eje 5.

Posición	Valor
A5 > 0	Bit 2 = 1
A5 ≤ 0	Bit 2 = 0

- **Bit 3 no se utiliza** y siempre es 0.
- **Bit 4:** indica si el punto ha sido programado o no con un robot de precisión absoluta.

Con independencia del valor del bit, el punto puede ser alcanzado tanto con robots de precisión absoluta como con robots que no sean precisión absoluta. El bit 4 sirve solo de información y no tiene influencia en el modo de cálculo del punto por parte de la unidad de control del robot. Esto también significa que cuando un robot se programa offline, se puede prescindir del bit 4.

Descripción	Valor
El punto no se ha programado con un robot de precisión absoluta.	Bit 4 = 0
El punto se ha programado con un robot de precisión absoluta.	Bit 4 = 1

TURN

- La indicación de Turn (giro) también permite poder alcanzar ángulos de eje superiores a $+180^\circ$ o inferiores a -180° , sin una estrategia de desplazamiento especial (por ejemplo, puntos auxiliares). En los ejes rotatorios, los bits individuales determinan el signo que precede al valor del eje del siguiente modo:

Bit = 0: ángulo $\geq 0^\circ$

Bit = 1: ángulo $< 0^\circ$

- Vista general de todos los ejes

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 $< 0^\circ$

- **Ejemplo**

```
DECL POS XP1 = {X 900, Y 0, Z 800, A 0, B 0, C 0, S 6, T 19}
```

T 19 corresponde a T 'B010011'. Esto significa:

Eje	Angulo	Binario
A 1	negativo	1
A 2	negativo	2
A 3	positivo	4
A 4	positivo	8
A 5	negativo	16
A 6	positivo	32

Procedimiento

1. Cargar el programa en el editor como experto mediante el pulsador **Abrir**
2. Manipular **Status** y **Turn**. Guardar temporalmente los nuevos valores calculados en una nueva variable
3. Comprobar los ajustes predefinidos para la programación de movimientos y aceptar o inicializar de nuevo:
 - Herramienta (\$TOOL y \$LOAD)
 - Ajustes base (\$BASE)
 - Herramienta guiada por robot o externa (\$IPO_MODE)
 - Velocidad
 - Aceleración
 - Posible distancia de aproximación
 - Posible control de la orientación
4. Crear la instrucción de movimiento, compuesta de:
 - Tipo de movimiento (PTP, LIN, CIRC)
 - Punto de destino (con CIRC también punto auxiliar)
 - Para CIRC posible ángulo circular (CA)
 - Activar el posicionamiento aproximado (C_PTP, C_DIS, C_ORI, C_VEL)
5. En caso de nuevo movimiento de vuelta al punto 3
6. Cerrar el editor y guardar

5.5 Ejercicio: Paletizado y despaletizado**Objetivo del ejercicio**

Después de completar correctamente este ejercicio, se dispondrá de la competencia necesaria para efectuar las siguientes tareas:

- Trabajar con campos para el cálculo de coordenadas de posición
- Manejo de estructuras y del separador de puntos
- Aplicación de bucles FOR encajados
- Programación de movimientos sin formularios en línea
- Desplazamiento hasta las coordenadas de destino calculadas.

Requisitos

Los siguientes requisitos son necesarios para completar este ejercicio correctamente:

- Conocimientos sobre campos, estructuras, bucle FOR
- Conocimientos teóricos sobre la manipulación de datos
- Conocimientos teóricos sobre la programación de movimientos sin formulario en línea

Formulación

Se debe crear un programa con el que se recojan 16 cubos del depósito de cubos y se depositen en las posiciones previstas sobre la mesa. A continuación se deberán recoger de nuevo todos los cubos y llevarse de vuelta al depósito de cubos.

Todas las posiciones para el desplazamiento deben calcularse tomando como base **una** posición programada. Los desplazamientos a las posiciones

de descarga también se deberán realizar mediante posiciones previas calculadas. Las distancias entre las distintas posiciones de descarga son en todos los lados de 80 mm respectivamente. La distancia entre la posición de descarga y la posición previa deberá ser de 100 mm.

Tarea parcial 1: Plan de ejecución del programa

- Crear un PEP para la tarea descrita.

Tarea parcial 2: Cálculo de las posiciones de descarga

1. Crear la posición de inicio mediante un formulario en línea.
2. Crear variables adecuadas para el cálculo de las posiciones de descarga.
3. Inicializar las variables con valores iniciales adecuados.
4. Calcular las 16 posiciones de descarga en la mesa de trabajo.

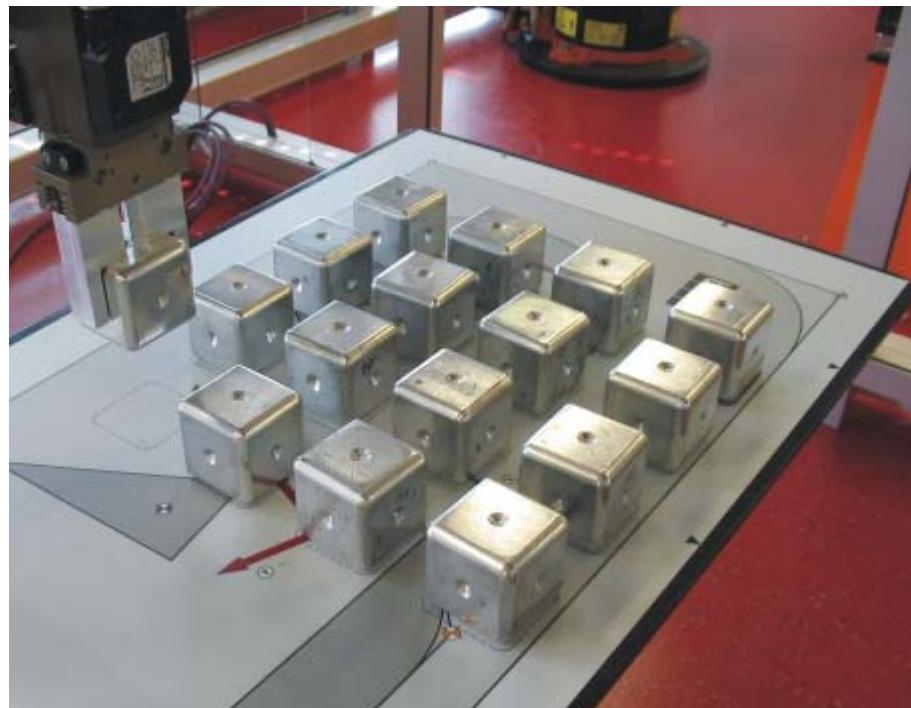


Fig. 5-12

Tarea parcial 3: Cálculo de las posiciones previas

1. Esta posición se encuentra 100 mm por encima la posición real de la mesa del cubo.
2. Crear variables adecuadas para el cálculo de las posiciones previas.
3. Inicializar las variables con valores iniciales adecuados.
4. Calcular las 16 posiciones previas 100 mm por encima de la mesa. Como base para el cálculo se deberán aplicar las posiciones de descarga ya calculadas.



Fig. 5-13

Tarea parcial 4: Paletizado y despaletizado de los cubos

1. Ampliar el programa con las instrucciones de avance necesarias, de forma que todos los cubos se depositen en la mesa y, a continuación, se vuelvan a recoger.
2. Para la recogida y del depósito de los cubos en el depósito se deberán utilizar los módulos ya existentes.
3. Utilizar para ello solo instrucciones KRL.
4. Si es necesario, aplicar los elementos auxiliares aprendidos para la estructuración del programa.

Lo que se debe saber tras el ejercicio:

1. Calcular los valores para A, B, C, X, Y, Z:

INT X, Y, Z

REAL A, B, C

A= 4.5

B= -2

C= 4.1

X= 2.5

Y= 4

Z=0.1

A= A * B + Y ;A=

B= B * Z + X ;B=

C= C + 4/3 ;C=

X= X + 2.5 * 4 ;X=

Y= Y – 10.0/4 ;Y=

Z= 14 – Z * C + A ;Z=

2. ¿Qué se indica con la indicación CA en el movimiento circular?

.....
.....

3. ¿Qué se puede conseguir con la instrucción "CONTINUE"?

.....
.....

4. ¿Qué hay que tener en cuenta en la utilización de la instrucción "CONTINUE"?

.....
.....

6 Trabajar con variables de sistema

6.1 Medición del tiempo ciclo mediante temporizador

Descripción de la medición del tiempo ciclo con el temporizador de sistema de KUKA



Fig. 6-1

- \$TIMER[1]
- \$TIMER[2]
- ...
- \$TIMER[32]

Las variables de sistema \$TIMER[nº] sirven para medir procesos temporales.



La introducción del valor/indicación del temporizador \$TIMER[nº] tiene lugar en milisegundos **ms**.

Inicio y parada del temporizador a través de KRL

- INICIO: \$TIMER_STOP[nº] = FALSE
- PARADA: \$TIMER_STOP[nº] = TRUE



El temporizador también se puede preasignar, iniciar y detener manualmente a través de la ventana de indicación.

Principio de medición del tiempo ciclo

Preasignación del temporizador

- la preasignación del temporizador es de 0 ms en el momento del suministro
- el temporizador mantiene su valor actual
- un temporizador se puede adelantar o restituir al valor deseado

```
;Preasignación de Timer 5 con 0ms
$TIMER[5] = 0

; Timer 12 ajustado a 1,5 segundos
$TIMER[12] = 1500

; Timer 4 reseteado a -8 segundos
$TIMER[4] = -8000
```

- Resetear e iniciar un temporizador

```
; Timer 7 Reset a 0ms
$TIMER[7] = 0
; Arranque Timer 7
$TIMER_STOP[7] = FALSE
```

■ Parada de un temporizador y comparación posterior

```
; Timer 7 en marcha
...
; Stop Timer 7
$TIMER_STOP[7] = TRUE

; en 10 segundos o más acontece ...
IF $TIMER[7] >= 10000 THEN
...
```



El inicio y la parada del temporizador se realiza siempre a través del **puntero de ejecución en avance**.

Procedimiento en la medición del tiempo ciclo

1. Selección de un temporizador "libre" de los 32 temporizadores posibles
2. Preasignación / reset del temporizador
3. Arrancar el temporizador bajo consideración de del puntero de ejecución en avance
4. Detener el temporizador bajo consideración del puntero de ejecución en avance
5. Guardar temporalmente el posible tiempo del ciclo actual o predesignar de nuevo el temporizador

```
DEF MY_TIME( )
...
INI
$TIMER[1] = 0 ; Resetear TIMER 1
PTP HOME Vel=100% DEFAULT

WAIT SEC 0 ; Activar parada del procesamiento en avance
$TIMER_STOP[1]=FALSE ; Arrancar medición tiempo de ciclo

PTP XP1
PTP XP2
LIN XP3
...
PTP X50
PTP HOME Vel=100% DEFAULT

WAIT SEC 0 ; Activar parada del procesamiento en avance
$TIMER_STOP[1]=TRUE ; Parar medición tiempo de ciclo

; Memorización intermedia del tiempo de ciclo actual en el Timer 12
$TIMER[12] = $TIMER[1]
END
```

6.2 Ejercicio: Medición del tiempo ciclo y optimización

Objetivo del ejercicio

Después de terminar con éxito este ejercicio, se dispone de la competencia necesaria para efectuar las siguientes tareas:

- Trabajar con un temporizador (inicializar, iniciar, detener).
- Realizar la optimización del tiempo de ciclo.

Requisitos

Las siguientes condiciones son necesarias para efectuar este ejercicio con éxito:

- Conocimientos sobre variables de sistema \$TIMER[x].

Tarea

Duplicar el programa "Palet" y asignar el nombre "Medición de tiempo". Se deben medir tres tiempos. El temporizador 1 debe medir el tiempo del paletizado, el temporizador 2 el proceso de recogida y el temporizador 3 el tiempo total de ambos procesos. Tener en cuenta que el temporizador **no** se debe iniciar o detener con el avance.

1. Para la medición del tiempo de paletizado, se debe utilizar \$TIMER[1] y guardar adicionalmente el valor final en \$TIMER[4].
2. Nombrar el \$TIMER[1] como "Paletizado actual" y el \$TIMER[4] como "Paletizado ANTERIOR".
3. Para la medición del tiempo de despaletizado, se debe utilizar \$TIMER[2] y guardar adicionalmente el valor final en \$TIMER[5].
4. Nombrar el \$TIMER[2] como "Despaletizado actual" y el \$TIMER[5] como "Despaletizado ANTERIOR".
5. Para la medición del tiempo total se utiliza \$TIMER[3] y se guarda en \$TIMER[6].
6. Nombrar el \$TIMER[3] como "Total actual" y el \$TIMER[5] como "Total ANTERIOR".
7. Intentar optimizar el programa de forma adecuada.
8. Compruebe el programa en los modos de servicio T1, T2 y Automático. Se deben tener en cuenta las prescripciones de seguridad enseñadas.

Lo que deberá saberse ahora:

1. ¿Cuántos temporizadores tiene el controlador de KUKA y cómo se inician?

.....
.....

2. Nombrar un ejemplo para una variable del fichero \$CONFIG.DAT?

.....
.....

7 Utilización de controles de ejecución de programa

7.1 Programar consultas o ramificaciones

Descripción de las consultas y ramificaciones con KRL

- Las ramificaciones se utilizan para dividir un programa en varias rutas.
- La instrucción `IF` comprueba una condición que puede ser verdadera (TRUE) o falsa (FALSE). Dependiendo de ello, se ejecutarán las instrucciones o no.
- Ramificación

```
IF ..... THEN
...
ELSE
...
ENDIF
```

Utilización de ramificaciones

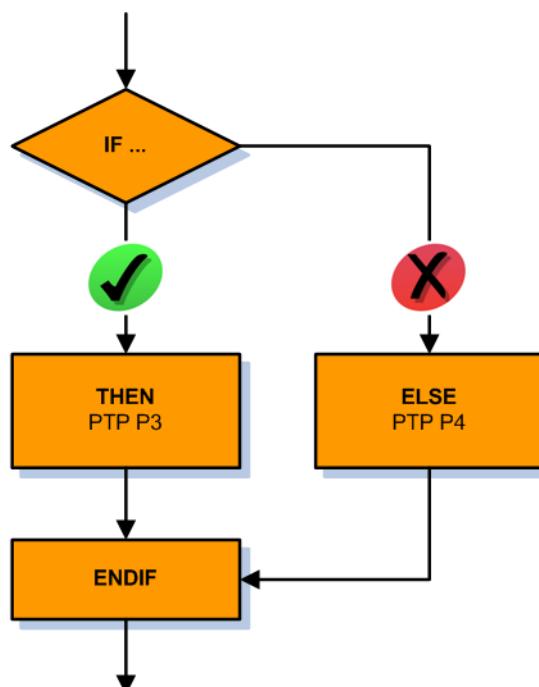


Fig. 7-1: Plano de desarrollo del programa: Ramificación IF

Ramificación

- con ramificación alternativa

```
IF condition THEN
Anweisung
ELSE
;Instrucción
ENDIF
```

- sin ramificación alternativa (consulta)

```
IF condition THEN
;Instrucción
ENDIF
```

Ejemplos de ramificaciones

Ramificación sin ramificación alternativa

```

DEF MY_PROG( )
DECL INT error_nr
...
INI
error_nr = 4
...
; sólo con error_nr 5 se desplaza a P21
IF error_nr == 5 THEN
PTP P21 Vel=100% PDAT21
ENDIF
...
END

```

Ramificación con ramificación alternativa

```

DEF MY_PROG( )
DECL INT error_nr
...
INI
error_nr = 4
...
; sólo con error_nr 5 se desplaza a P21, caso contrario P22
IF error_nr == 5 THEN
PTP P21 Vel=100% PDAT21
ELSE
PTP P22 Vel=100% PDAT22
ENDIF
...
END

```

Ramificación con condiciones de ejecución complejas

```

DEF MY_PROG( )
DECL INT error_nr
...
INI
error_nr = 4
...
; sólo con error_nr 1 o 10 o mayor que 99 se desplaza a P21
IF ((error_nr == 1) OR (error_nr == 10) OR (error_nr > 99)) THEN
PTP P21 Vel=100% PDAT21
ENDIF
...
END

```

Ramificación con expresiones booleanas

```

DEF MY_PROG( )
DECL BOOL no_error
...
INI
no_error = TRUE
...
; sólo en caso de ausencia de error (no_error) se desplaza a P21
IF no_error == TRUE THEN
PTP P21 Vel=100% PDAT21
ENDIF
...
END

```



La expresión IF no_error==TRUE THEN también se puede reducir a IF no_error THEN. Una omisión significa siempre la comparación con TRUE.

7.2 Programar el distribuidor

Descripción del distribuidor con KRL

- Si se desea diferenciar entre varios casos y ejecutar diferentes acciones para cada caso, esto se puede lograr con una instrucción switch case.
- La instrucción switch sirve para la diferenciación de casos.

- Una variable transmitida en la instrucción `switch` se utiliza como interruptor y salta en el bloque de instrucciones a la instrucción `case` predefinida.
- Si la instrucción `switch` no encuentra ninguna instrucción `case` predefinida, se ejecutará la sección `default`.
- Distribuidor

```
SWITCH ...
CASE ...
...
CASE ...
...
CASE
...
...
DEFAULT
...
ENDSWITCH
```

Utilización de distribuidores

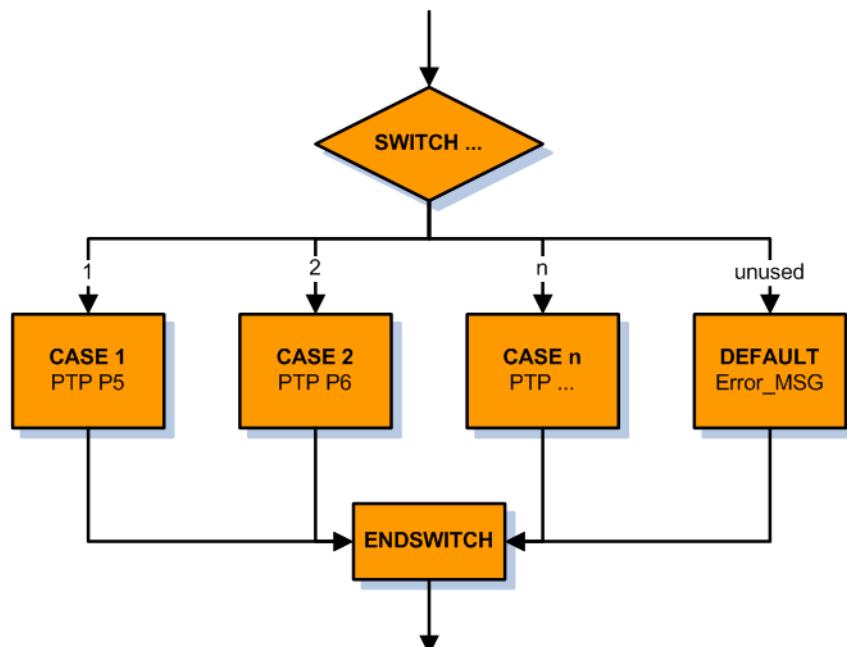


Fig. 7-2: Plano de desarrollo del programa: Distribuidor SWITCH - CASE

Distribuidor

- Un distribuidor se puede utilizar con los siguientes tipos de datos
 - INT (número entero)

```
SWITCH number
CASE 1
...

```

- CHAR (un carácter)

```
SWITCH symbol
CASE "X"
...
```

- ENUM (tipo de datos de enumeración)

```
SWITCH mode_op
CASE #T1
...
```

■ Solo con distribuidores definidos

```
SWITCH number
CASE 1
...
CASE 2
...
CASE 3
...
ENDSWITCH
```



En caso de *número* distinto de 1 o 2 o 3 se salta directamente a END-SWITCH, sin ejecutar una instrucción.

■ Solo con distribuidores definidos y un caso alternativo

```
SWITCH number
CASE 1
...
CASE 2
...
CASE 3
...
DEFAULT
...
ENDSWITCH
```



En caso de *número* distinto de 1 o 2 o 3 se salta al "Caso DEFAULT para ejecutar esta o estas instrucciones.

■ Con varias soluciones en un distribuidor

```
SWITCH number
CASE 1,2
...
CASE 3,4,5
...
CASE 6
...
DEFAULT
...
ENDSWITCH
```

Ejemplo de distribuidores

Distribuidores sin caso alternativo

```
DEF MY_PROG( )
DECL INT error_nr
...
INI
error_nr = 4
...
; sólo se puede desplazar en caso de un caso declarado
SWITCH error_nr
CASE 1
PTP P21 Vel=100% PDAT21
CASE 2
PTP P22 Vel=100% PDAT22
CASE 3
PTP P23 Vel=100% PDAT23
CASE 4
PTP P24 Vel=100% PDAT24
ENDSWITCH
...
```

Distribuidores sin caso alternativo

```

DEF MY_PROG( )
DECL INT error_nr
...
INI
error_nr = 99
...
; en caso de un caso no definido se desplaza a HOME
SWITCH error_nr
CASE 1
PTP P21 Vel=100% PDAT21
CASE 2
PTP P22 Vel=100% PDAT22
CASE 3
PTP P23 Vel=100% PDAT23
CASE 4
PTP P24 Vel=100% PDAT24
DEFAULT
PTP HOME Vel=100% DEFAULT
ENDSWITCH
...

```

Distribuidores con un tipo de datos de enumeración

```

DEF MY_PROG( )
ENUM COLOR_TYPE red, yellow, blue, green
DECL COLOR_TYPE my_color
...
INI
my_color = #red
...
SWITCH my_color
CASE #red
PTP P21 Vel=100% PDAT21
CASE #yellow
PTP P22 Vel=100% PDAT22
CASE #green
PTP P23 Vel=100% PDAT23
CASE #blue
PTP P24 Vel=100% PDAT24
ENDSWITCH
...

```

7.3 Programar bucles

Generalidades sobre bucles

- Los bucles sirven para la repetición de instrucciones de programación
- Un salto desde fuera a un cuerpo de bucle no está permitido
- Los bucles se pueden intercalar entre sí
- Existen diferentes tipos de bucles
 - Bucle sinfín
 - Bucle de conteo
 - Bucle condicionados
 - bucle finito
 - bucle infinito

7.3.1 Programar un bucle sinfín

Descripción de un bucle sinfín

- El bucle sinfín es un bucle que se vuelven a ejecutar después de cada ejecución.
- La ejecución se puede interrumpir por influencias externas.
- Sintaxis

```

LOOP
;   Instrucción
...
;   Instrucción
ENDLOOP

```

Principio de un bucle sinfín

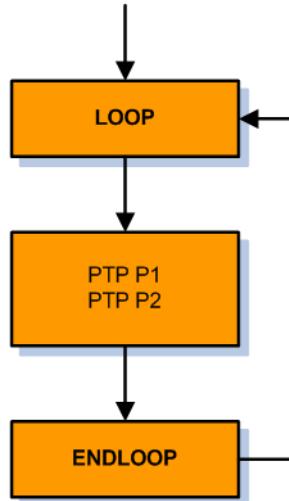


Fig. 7-3: Plano de desarrollo del programa: Bucle sinfín

- El bucle sinfín se puede abandonar mediante EXIT
- Al abandonar un bucle sinfín con EXIT se deberá asegurar la ausencia de colisión
- Si se encuentran dos bucles sinfín encajados, también serán necesarias dos instrucciones EXIT para abandonar ambos bucles

Ejemplo para la programación de un bucle sinfín

```

DEF MY_PROG( )
INI
PTP HOME Vel=100% DEFAULT

LOOP
PTP P1 Vel=90% PDAT1
PTP P2 Vel=100% PDAT2
PTP P3 Vel=50% PDAT3
PTP P4 Vel=100% PDAT4
ENDLOOP

PTP P5 Vel=30% PDAT5
PTP HOME Vel=100% DEFAULT
END

```

El punto P5 nunca se alcanza desde el punto de vista técnico del programa.

Bucle sinfín con interrupción

```

DEF MY_PROG( )
INI
PTP HOME Vel=100% DEFAULT

LOOP
PTP P1 Vel=90% PDAT1
PTP P2 Vel=100% PDAT2
IF $IN[3]==TRUE THEN ; Condición para interrupción
EXIT
ENDIF
PTP P3 Vel=50% PDAT3
PTP P4 Vel=100% PDAT4
ENDLOOP

PTP P5 Vel=30% PDAT5
PTP HOME Vel=100% DEFAULT
END

```



El punto P5 se alcanza en cuanto está activa la entrada 1.
Importante: Se deberá comprobar la ausencia de colisión en el desplazamiento entre P2 y P5.

7.3.2 Programar bucles de conteo

Definición de un bucle de conteo

- El bucle FOR es una estructura de control con la que llevar a cabo una o varias indicaciones con un número fijo de repeticiones.
- Sintaxis con anchura de paso +1

```

FOR counter = start TO last
;Instrucción
ENDFOR

```

- La anchura de paso (increment) también se puede indicar como número entero con la palabra clave STEP.

```

FOR counter = start TO last STEP increment
;Instrucción
ENDFOR

```

Principio de un bucle de conteo

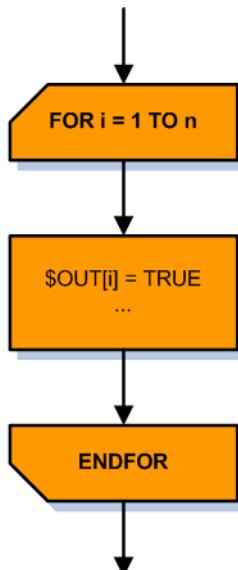


Fig. 7-4: Plano de desarrollo del programa: Bucle de conteo

- Para un bucle de conteo es necesaria una variable entera declarada previamente
- El bucle de conteo comienza en el valor `start` y finaliza como muy tarde en el valor `last`

```
FOR counter = start TO last
;Instrucción
ENDFOR
```

- El bucle de conteo se puede abandonar de forma inmediata mediante `EXIT`

Cómo funciona un bucle de conteo

```
DECL INT counter

FOR counter = 1 TO 3 Step 1
;Instrucción
ENDFOR
```

1. El contador de bucles se inicializa con el valor de inicio: `counter = 1`
2. El contador de bucles se incremente en `ENDFOR` con la anchura de paso `STEP`
3. El bucle vuelve a comenzar en la línea `FOR`
4. Control de la condición de entrada: La variable de conteo debe ser menor o igual que el valor final indicado, en caso contrario se finalizará el bucle
5. Dependiendo de la comprobación, se incrementarán de nuevo el contador de bucles o el bucle se finaliza y el programa continúa después de la línea `ENDFOR`

Conteo descendente con un bucle de conteo

```
DECL INT counter

FOR counter = 15 TO 1 Step -1
;Instrucción
ENDFOR
```



El valor inicial o valor de inicio del bucle debe ser mayor que el valor final para que el bucle pueda recorrerse varias veces.

Programación con un bucle de conteo

Ejemplos para la programación con bucles de conteo

- Bucle de conteo simple sin indicación de la anchura de paso

```
DECL INT counter

FOR counter = 1 TO 50
$OUT[counter] == FALSE
ENDFOR
```



Si no se indica la anchura de paso mediante `STEP` se utiliza automáticamente la anchura de paso +1.

- Bucle de conteo simple con indicación de la anchura de paso

```
DECL INT counter

FOR counter = 1 TO 4 STEP 2
$OUT[counter] == TRUE
ENDFOR
```



Este bucle solo se recorre dos veces. Una vez con el valor de inicio counter=1 y la segunda vez con counter=3. Con el valor de contador 5 el bucle se interrumpe inmediatamente.

- Bucle de conteo doble con indicación de la anchura de paso

```
DECL INT counter1, counter2

FOR counter1 = 1 TO 21 STEP 2
    FOR counter2 = 20 TO 2 STEP -2
        ...
    ENDFOR
ENDFOR
```



Primero se recorre siempre el bucle interior, aquí con counter1 y, a continuación, el exterior (counter2).

7.3.3 Programar bucle finito

Descripción de un bucle finito

- Un bucle finito también se denomina bucle controlado por cabezal.
- Un bucle de este tipo repite procesos mientras que se cumpla una condición determinada (condition).
- Sintaxis

```
WHILE condition
    ; Instrucción
ENDWHILE
```

- El bucle finito se puede abandonar de forma inmediata mediante EXIT.

Principio de un bucle finito

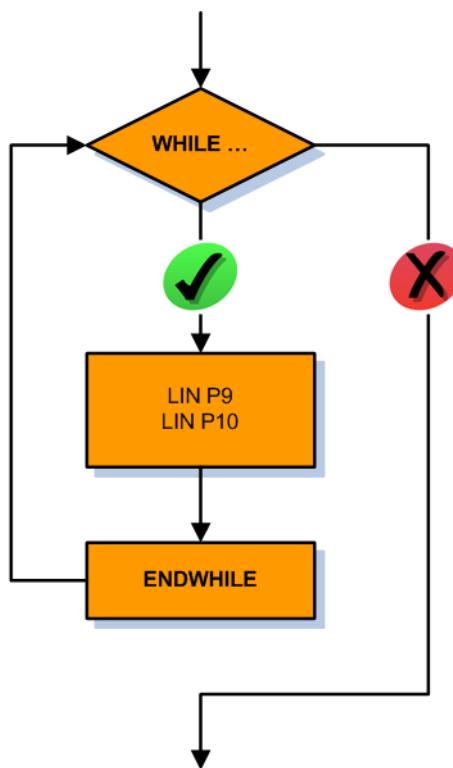


Fig. 7-5: Plan de desarrollo de la programación: Bucle finito

- Los bucles finitos se utilizan para comprobar previamente si se ha iniciado un proceso repetitivo
- Para ejecutar el bucle se deberá cumplir la condición de ejecución
- El incumplimiento de una condición de ejecución tiene como consecuencia que el bucle se finaliza inmediatamente y las instrucciones se ejecutan según ENDWHILE

Programación con un bucle finito

- Bucle finito con condición de ejecución sencilla

```
...
WHILE IN[41]==TRUE ; Pieza preparada en el depósito
PICK_PART( )
ENDWILE
...
```



La expresión WHILE IN[41]==TRUE también se puede reducir a WHILE IN[41]. Una omisión significa siempre la comparación con TRUE.

- Bucle finito con condición de ejecución negada sencilla

```
...
WHILE NOT IN[42]==TRUE ; Entrada 42: Depósito está vacío
PICK_PART( )
ENDWILE...
```

O

```
...
WHILE IN[42]==FALSE ; Entrada 42: Depósito está vacío
PICK_PART( )
ENDWILE...
```

- Bucle finito con condición de ejecución compleja

```
...
WHILE ((IN[40]==TRUE) AND (IN[41]==FALSE) OR (counter>20))
PALETTE( )
ENDWILE
...
```

7.3.4 Programar bucle infinito

Descripción de un bucle infinito

- El bucle infinito también se denomina bucle de control por pedal.
- Este bucle infinito ejecuta primero la indicación y al final comprueba si se ha cumplido la indicación (*condition*) para poder abandonar el bucle.
- Sintaxis

```
REPEAT
; Instrucción
UNTIL condition
```

- El bucle infinito se puede abandonar de forma inmediata mediante EXIT.

Principio de un bucle infinito

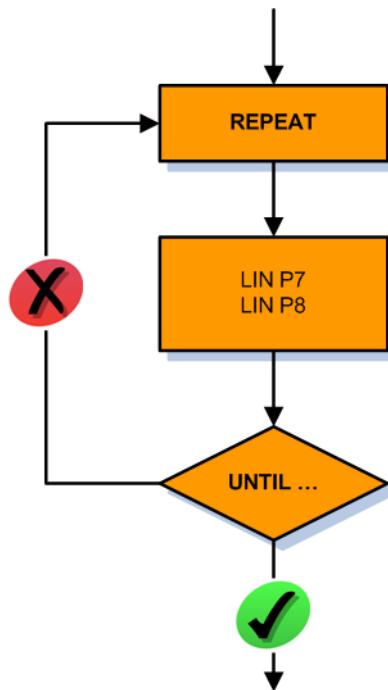


Fig. 7-6: Plan de desarrollo de la programación: Bucle infinito

- En caso de un resultado positivo de la condición, se abandona el bucle y se ejecuta la instrucción UNTIL.
- En caso de un resultado negativo de la condición, el bucle se inicia de nuevo con REPEAT.
- Bucle infinito con condición de ejecución sencilla

```

...
REPEAT
  PICK_PART( )
  UNTIL IN[42]==TRUE ; Entrada 42: Depósito está vacío
...
  
```



La expresión UNTIL IN[42]==TRUE también se puede reducir a UNTIL IN[42]. Una omisión significa siempre la comparación con TRUE.

- Bucle infinito con condición de ejecución compleja

```

...
REPEAT
  PALETTE( )
  UNTIL ((IN[40]==TRUE) AND (IN[41]==FALSE) OR (counter>20))
...
  
```



El bucle se finaliza en caso de un resultado positivo.

7.4 Programar funciones de espera

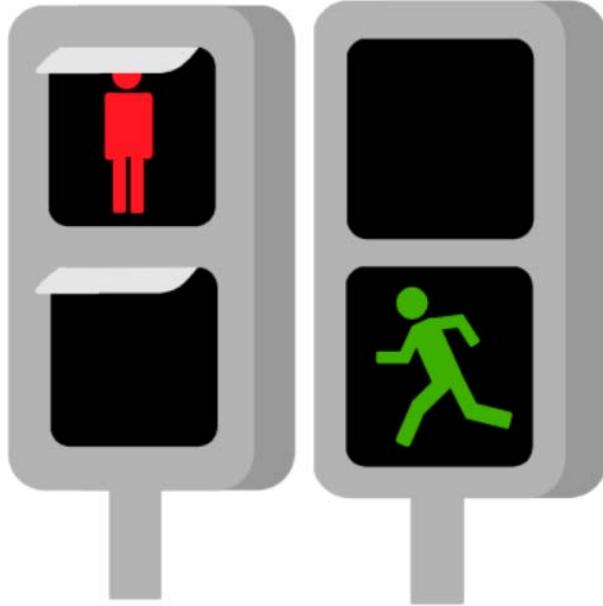


Fig. 7-7

Programación en KRL de

- funciones de espera dependientes del tiempo
- funciones de espera dependientes de una señal

7.4.1 Función de espera dependiente del tiempo

Descripción de una función de espera dependiente del tiempo con KRL

- La función de espera dependiente del tiempo espera el tiempo indicado (*time*) antes de que el proceso pueda continuar
- Sintaxis

`WAIT SEC time`

Principio de la función de espera dependiente del tiempo

- La función de espera dependiente del tiempo tiene la base de tiempo de segundos (s)

 La base de tiempo para un temporizador de KUKA (`$TIMER[n°]`) es en milisegundos (ms)

- El tiempo máximo es de 2147484 segundos, que son más de 24 días

 El formulario en línea para la función de espera dependiente del tiempo puede esperar como máximo 30 segundos

- El valor del tiempo también se puede transmitir con una variable adecuada
- La unidad de tiempo más pequeña adecuada es de 0,012 segundos (ciclo de interpolación)
- Si el tiempo indicado es negativo, no se esperará
- La función de espera dependiente del tiempo activa una parada del procesamiento en avance, por lo que el posicionamiento aproximado no es posible
- Para generar de forma precisa solo una parada del procesamiento en avance, se utilizar la instrucción `WAIT SEC 0`

Programación de una función de espera dependiente del tiempo

- Función de espera dependiente del tiempo con un tiempo fijo

```
PTP P1 Vel=100% PDAT1
PTP P2 Vel=100% PDAT2
WAIT SEC 5.25
PTP P3 Vel=100% PDAT3
```

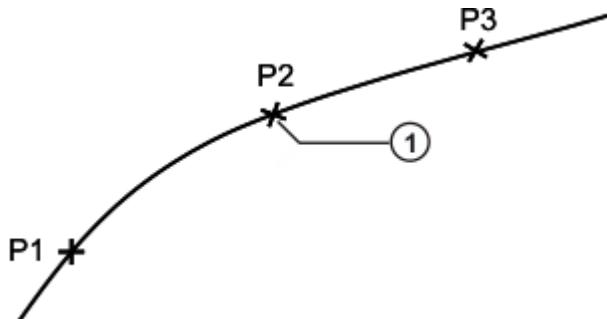


Fig. 7-8: Movimiento ejemplar para lógica

Pos.	Observación
1	El movimiento se interrumpe para 5,25 segundos en el punto P2.

- Función de espera dependiente del tiempo con un tiempo calculado

```
WAIT SEC 3*0.25
```

- Función de espera dependiente del tiempo con una variable

```
DECL REAL time
time = 12.75
WAIT SEC time
```

7.4.2 Función de espera dependiente de una señal

Descripción de una función de espera dependiente de una señal

- La función de espera dependiente de una señal continúa comutando si se cumple la condición (*condition*) y el proceso continúa
- Sintaxis

```
WAIT FOR condition
```

Principio de la función de espera dependiente de una señal

- La función de espera dependiente de una señal activa una parada del procesamiento en avance, por lo que el posicionamiento aproximado no es posible
- A pesar de que se ha cumplido la condición se genera una parada del procesamiento en avance
- Mediante la instrucción **CONTINUE** se evita la parada del procesamiento en avance

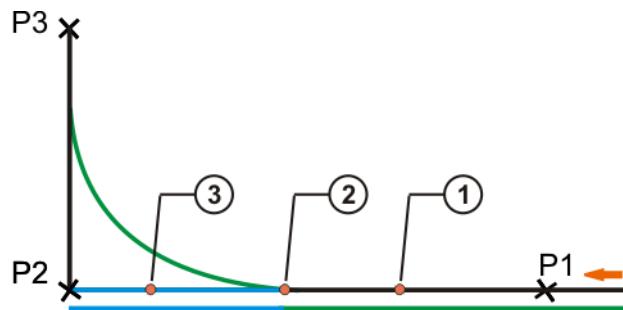


Fig. 7-9: Movimiento ejemplar para lógica con avance

	Posición	Rango de comutación
1	Posición del robot en la que se cumple la condición para que el robot realice un posicionamiento aproximado	Rango de comutación verde para la Activación del contorno de posicionamiento aproximado. Solo se ajusta y no se puede volver a desactivar.
2	Inicio del movimiento de aproximación	Consulta de la Activación <ul style="list-style-type: none"> ■ TRUE: Posicionamiento aproximado ■ FALSE: Acercamiento al punto de destino
3	Posición del robot en la que se cumple la condición para que el robot no realice un posicionamiento aproximado	Rango de comutación azul para el acercamiento y la parada en el punto P2

Programación de una función de espera dependiente de una señal

- WAIT FOR con parada del procesamiento en avance

```

PTP P1 Vel=100% PDAT1
PTP P2 CONT Vel=100% PDAT2
WAIT FOR $IN[20]
PTP P3 Vel=100% PDAT3

```

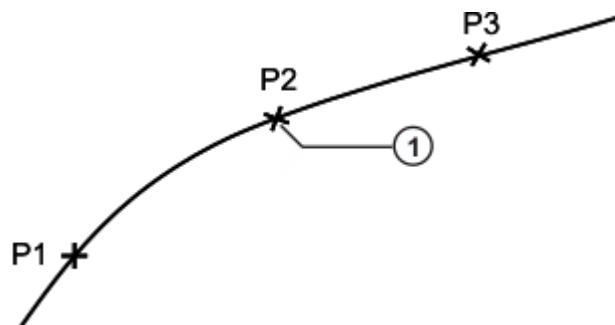


Fig. 7-10: Movimiento ejemplar para lógica

	Observación
1	El movimiento se interrumpe en el punto P2. Tras la parada exacta se verifica la entrada 20. Si el estado de la entrada está correcto se puede continuar directamente, caso contrario se espera al estado.

- WAIT FOR con ejecución en avance (aplicación de CONTINUE)

```

PTP P1 Vel=100% PDAT1
PTP P2 CONT Vel=100% PDAT2
CONTINUE
WAIT FOR ($IN[10] OR $IN[20])
PTP P3 Vel=100% PDAT3

```

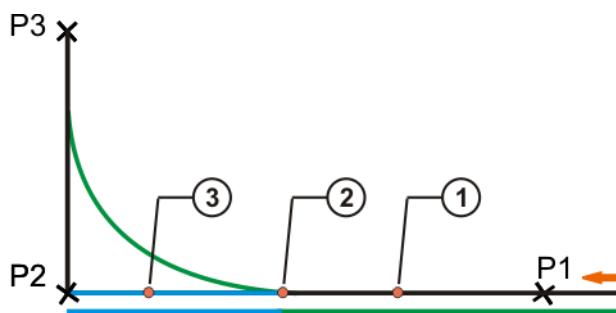


Fig. 7-11: Movimiento ejemplar para lógica con avance

	Acción
1	La entrada 10 o 20 son o han sido TRUE desde el puntero de ejecución en avance, de este modo se realiza un posicionamiento aproximado
2	Condición cumplida poco antes, de este modo se realiza el posicionamiento aproximado del robot
3	Condición cumplida demasiado tarde, de este modo no se puede realizar el posicionamiento aproximado del robot y debe desplazarse al punto P2. En el punto P2 pueden continuar el desplazamiento de forma inmediata

7.5 Ejercicio: Técnicas de bucles

Objetivo del ejercicio

Después de terminar con éxito este ejercicio, se dispone de la competencia necesaria para efectuar las siguientes tareas:

- Sustituir bucles FOR por otros bucles
- Utilización de bucles WHILE o REPEAT/UNTIL

Requisitos

Las siguientes condiciones son necesarias para efectuar este ejercicio con éxito:

- Conocimientos teóricos sobre la funcionalidad de las distintas técnicas de bucles para la programación

Tarea

1. Transformar en la sección del programa Paletizado el bucle doble FOR en un bucle WHILE.
2. En el despaletizado se deben transformar los bucles existentes en bucles REPEAT/UNTIL.
3. Duplicar el programa "Palet" con el nombre "Pal_bucle".
4. Actualizar el PEP disponible para los nuevos bucles.
5. Crear nuevas variables de conteo para los bucles. Tener en cuenta la inicialización correcta o la restauración adecuada de las variables.
6. Modificar la retirada de los cubos de forma que en primer lugar se recoja y se coloque de vuelta el último cubo depositado.
7. Compruebe el programa en los modos de servicio T1, T2 y Automático. Se deben tener en cuenta las prescripciones de seguridad enseñadas.

Lo que deberá saberse ahora:

1. En el distribuidor SWITCH/CASE existe la instrucción "DEFAULT". ¿Qué función tiene esta instrucción "DEFAULT"?
-
.....

2. ¿Con qué instrucción se puede ajustar la anchura de paso en el bucle FOR?

.....
.....
.....
.....
.....
3. ¿Qué bucles se pueden abandonar con la instrucción "EXIT"?

.....
.....
.....
.....
.....
4. ¿Qué parte se puede omitir en una ramificación ? a. IF b. THEN c. ELSE d. ENDIF

.....
.....
.....
.....
.....
5. ¿Qué es incorrecto en este ejemplo de programa?

```
IF $IN[14]==FALSE THEN  
$OUT[12]=TRUE  
GOTO MARCA1  
ELSE  
$OUT[12]=FALSE  
GOTO MARCA2  
ENDIF  
WHILE $IN[17]==TRUE  
PTP P1  
MARC1: ...  
ENDWHILE  
MARC2: ...  
PTP HOME
```

.....
.....

8 Funciones de conmutación con KRL

8.1 Programación de funciones de conmutación sencillas

Descripción de funciones de conmutación sencillas

Generalidades

- La unidad de control del robot puede administrar, como máximo, 4.096 entradas digitales y 4.096 salidas digitales
- Las entradas/salidas se realizan a través de sistemas de buses de campo opcionales
- La configuración es específica del cliente
- La proyección se realiza a través de WorkVisual

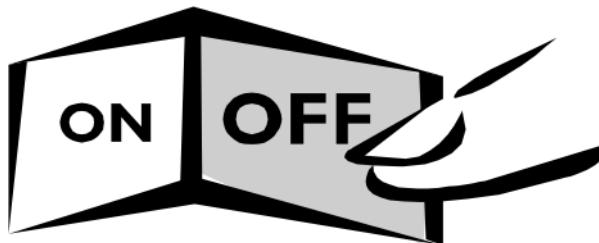


Fig. 8-1

Posibilidades de funciones de conmutación sencillas

- Conexión / desconexión sencilla de una salida (con avance / parada del procesamiento en avance)
- Pulsación de una salida
- Comutación de una salida con el puntero de ejecución principal (sin parada del procesamiento en avance)

Función de las funciones de conmutación sencillas

Conexión / desconexión sencilla de una salida

- Conexión de una salida

```
$OUT[10]=TRUE
```

- Desconexión de una salida

```
$OUT[10]=FALSE
```

- Mediante la comutación de una salida se genera una parada del procesamiento en avance, de este modo el movimiento no podrá programarse por aproximación

```
...
PTP P20 CONT Vel=100% PDAT20
$OUT[30]=TRUE
PTP P21 CONT Vel=100%PDAT21
```

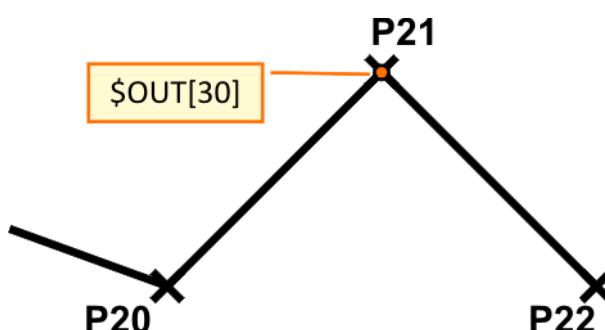


Fig. 8-2: Comutación con parada del procesamiento en avance



Se generan dos mensajes de error:
 Aproximación no posible (módulo *nombre*, paso *número*)
 Secuencia de instrucciones sin aptitud de aproximación (módulo *nombre*, paso *número*)

- Mediante la utilización de la instrucción CONTINUE se anula la parada del procesamiento en avance
- Mediante la utilización de la instrucción CONTINUE se conmuta con el avance
- El posicionamiento aproximado es posible con CONTINUE
- CONTINUE se refiere **solo** a la línea siguiente (incluyendo líneas en blanco)

```
...
PTP P20 CONT Vel=100% PDAT20
CONTINUE
$OUT[30]=TRUE
PTP P21 CONT Vel=100%PDAT21
```

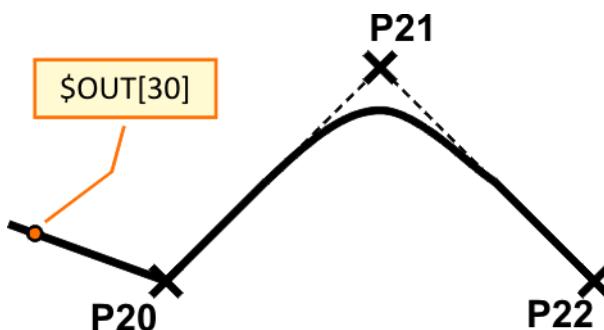


Fig. 8-3: Conmutación en el avance

Conmutación de una salida con el puntero de ejecución principal

- Se pueden conectar hasta 8 salidas referidas a la ejecución principal y sin parada del procesamiento en avance
- Se encuentra programada una parada exacta que se conecta al alcanzar el punto de destino
- Si se encuentra programado posicionamiento aproximado, se conectará en el centro del movimiento de aproximación del punto de destino

```
...
LIN P20 CONT Vel=100% PDAT20
$OUT_C[30]=TRUE
LIN P21 CONT Vel=100%PDAT21
```

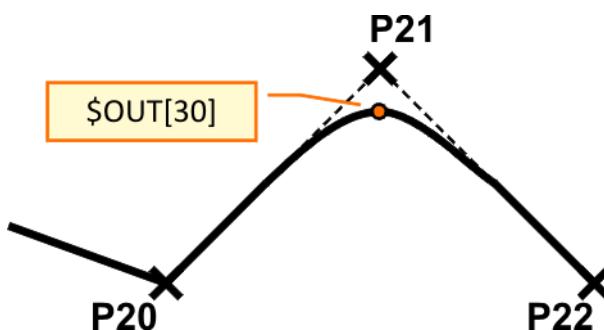


Fig. 8-4: Conmutación con la ejecución principal

Pulsación de una salida

- Activa un impulso
- Con ello la salida se pone a un nivel definido durante un tiempo determinado

- A continuación, el sistema devuelve automáticamente la salida a su estado inicial
- La instrucción PULSE provoca una parada del procesamiento en avance

AVISO

El impulso no se interrumpe si se da una PARADA DE EMERGENCIA, una parada de operación o una parada por error.

- Sintaxis

PULSE (*señal*, *nivel*, *duración del impulso*)

Elemento	Descripción
<i>Señal</i>	Tipo: BOOL Salida en la que se genera el impulso. Son admisibles: <ul style="list-style-type: none"> ■ OUT[Nr] ■ Variable de señal
<i>Nivel</i>	Tipo: BOOL Expresión lógica: <ul style="list-style-type: none"> ■ TRUE hace referencia a un impulso positivo (alto). ■ FALSE hace referencia a un impulso negativo (bajo).
<i>Longitud del impulso</i>	Tipo: REAL p. ej. 1000000 segundos

```
PULSE ($OUT[30], TRUE, 20); Impulso positivo
```

```
PULSE ($OUT[31], FALSE, 20); Impulso negativo
```

- Si se programa un impulso antes de la instrucción END el tiempo de ejecución del programa se prolongará correspondientemente

```
...
PULSE ($OUT[50], TRUE, 2)
END
```

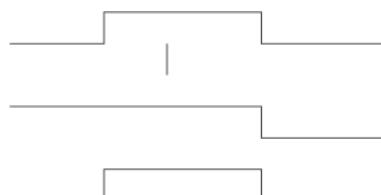


Fig. 8-5: PULSE+END ejemplo

- Si la ejecución del programa se devuelve a su estado inicial (RESET) o se interrumpe (CANCEL) mientras está activo un impulso, el impulso se restablecerá de inmediato

```
...
PULSE ($OUT[50], TRUE, 2)
; Programa ahora reseteado o deseleccionado
```



Fig. 8-6: PULSE+RESET ejemplo

Programación de funciones de conmutación sencillas

■ Comutación de salidas con parada del procesamiento en avance

```
...
LIN P20 CONT Vel=100% PDAT20
$OUT[50]=TRUE ; Conectar
LIN P21 CONT Vel=100%PDAT21
$OUT[50]=FALSE ; desconectar
LIN P22 CONT Vel=100%PDAT22
```

■ Comutación de salidas mediante función de pulso con parada del procesamiento en avance

```
...
LIN P20 CONT Vel=100% PDAT20
PULSE ($OUT[50], TRUE, 1.5) ; Impulso positivo
PULSE ($OUT[51], FALSE, 1.5) ; Impulso negativo
LIN P21 CONT Vel=100%PDAT21
```

■ Comutación de salidas en el avance

```
...
LIN P20 CONT Vel=100% PDAT20
CONTINUE
$OUT[50]=TRUE ; Conectar
LIN P21 CONT Vel=100%PDAT21
CONTINUE
$OUT[50]=FALSE ; desconectar
LIN P22 CONT Vel=100%PDAT22
```

■ Comutación de salidas mediante función de pulso en el avance

```
...
LIN P20 CONT Vel=100% PDAT20
CONTINUE
PULSE ($OUT[50], TRUE, 1.5) ; Impulso positivo
CONTINUE
PULSE ($OUT[51], FALSE, 1.5) ; Impulso negativo
LIN P21 CONT Vel=100%PDAT21
```

■ Comutación de salidas con la ejecución principal

```
...
LIN P20 CONT Vel=100% PDAT20
$OUT_C[50]=TRUE
LIN P21 CONT Vel=100%PDAT21
```

8.2 Programación de funciones de conmutación referidas a la trayectoria

TRIGGER WHEN DISTANCE

Descripción de funciones de conmutación referidas a la trayectoria con TRIGGER WHEN DISTANCE



Fig. 8-7: Aplicación de bucles

- La instrucción de conmutación de trayectoria TRIGGER lanza una instrucción definida
- La instrucción se refiere al punto de inicio y de destino del paso de movimiento

- La instrucción se ejecuta de forma paralela al movimiento del robot
- Es posible un corrimiento temporal del punto de conmutación

Funcionamiento de las funciones de conmutación referidas a la trayectoria con

TRIGGER WHEN DISTANCE

Sintaxis

- TRIGGER WHEN DISTANCE=*Posición* DELAY=*Tiempo* DO *Instrucción* <PRIO=*Prioridad*>
- **Posición:** Determina en qué punto se lanza la instrucción. Valores posibles:
 - **0:** La instrucción se lanza en el punto de arranque del paso de movimiento.
 - **1:** La instrucción se lanza en el punto de destino. Si el punto de destino no se ha posicionado de forma aproximada, la instrucción se lanza en la mitad de la curva de aproximación.
- **Tiempo:** De este modo se define un tiempo de desplazamiento desde la posición seleccionada
 - se pueden utilizar valores positivos y negativos
 - la base de tiempo es en milisegundos (ms)
 - se pueden utilizar sin problemas tiempos de hasta 10.000.000 ms
 - en caso de un tiempo demasiado extenso o reducido, se conectarán como máximo o como mínimo en los límites de conmutación
- **Instrucción:** Posibilidades para ello:
 - La asignación de valor a una variable.



La asignación de valor se puede realizar a variables de tiempo de ejecución reducidas.

- Instrucción OUT
- Instrucción PULSE
- Llamada de un subprograma. En este caso **se debe** indicar la *Prioridad*.
- **Prioridad** (solo para la llamada de un subprograma):
 - Están disponibles las prioridades 1, 2, 4 - 39 y 81 -128.
 - Las prioridades 40 - 80 están reservadas para casos en los que la prioridad se asigna automáticamente por el sistema. Si la prioridad debe adjudicarla de forma automática el sistema, se programa: PRIO = -1.



Antes de un movimiento pueden existir como máximo 8 TRIGGER.

Programación de funciones de conmutación referidas a la trayectoria

TRIGGER WHEN DISTANCE

Posibilidades de conmutación con TRIGGER WHEN DISTANCE

- El punto de arranque y el punto de destino son puntos de parada exacta

```
LIN XP1
LIN XP2
TRIGGER WHEN DISTANCE = 0 DELAY = 20 DO dues = TRUE
TRIGGER WHEN DISTANCE = 1 DELAY = -25 DO UP1() PRIO=75
LIN XP3
LIN XP4
```

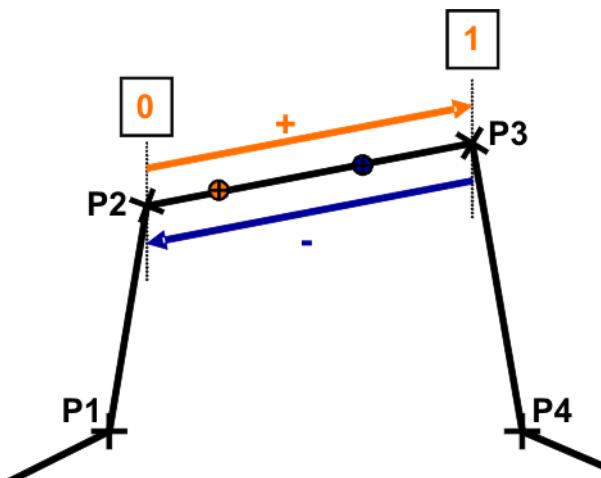


Fig. 8-8: TRIGGER WHEN DISTANCE Ejemplo con parada exacta/parada exacta

DISTANCE	Rango de comutación	DELAY
0 (naranja)	entre 1 y 0	+
1 (azul)	entre 1 y 0	-

- El punto de inicio es punto de posicionamiento aproximado y el punto de destino es punto de parada exacta

```
LIN XP1
LIN XP2 C_DIS
TRIGGER WHEN DISTANCE = 0 DELAY = 20 DO duese = TRUE
WHEN DISTANCE = 1 DELAY = -25 DO UP1() PRIO=75
LIN XP3
LIN XP4
```

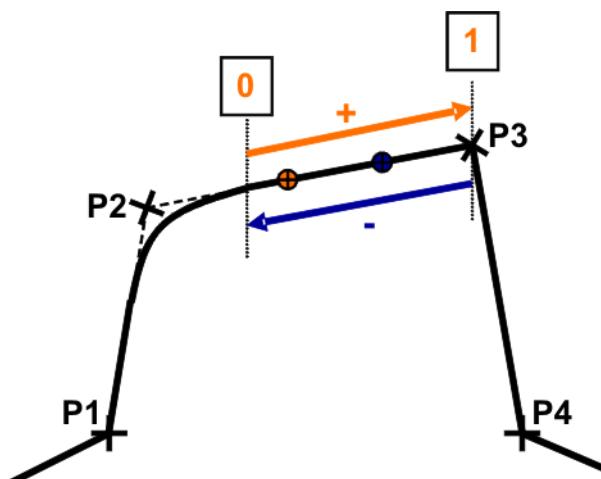


Fig. 8-9: TRIGGER WHEN DISTANCE Ejemplo con posicionamiento aproximado/parada exacta

DISTANCE	Rango de comutación	DELAY
0 (naranja)	entre 1 y 0	+
1 (azul)	entre 1 y 0	-

- Punto de inicio es punto de parada exacta y el punto de destino es punto de posicionamiento aproximado

```

LIN XP1
LIN XP2
TRIGGER WHEN DISTANCE = 0 DELAY = 20 DO duese = TRUE
TRIGGER WHEN DISTANCE = 1 DELAY = -25 DO UP1() PRIO=75
LIN XP3 C_DIS
LIN XP4

```

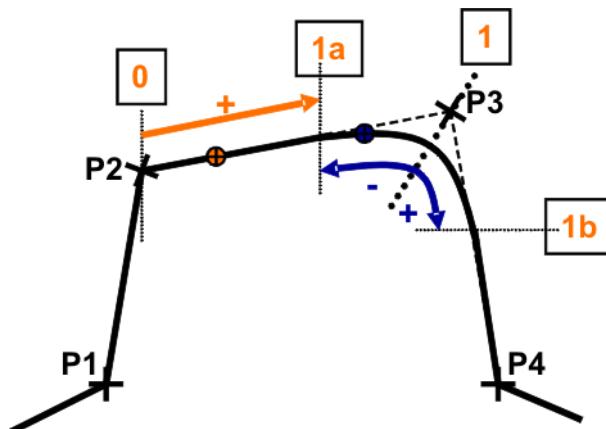


Fig. 8-10: TRIGGER WHEN DISTANCE Ejemplo con parada exacta/posicionamiento aproximado

DISTANCE	Rango de conmutación	DELAY
0 (naranja)	entre 0 y 1a	+
1 (azul)	entre 1a y 1b	-/+

- El punto de inicio y el punto de destino son puntos de posicionamiento aproximado

```

LIN XP1
LIN XP2 C_DIS
TRIGGER WHEN DISTANCE = 0 DELAY = 20 DO duese = TRUE
TRIGGER WHEN DISTANCE = 1 DELAY = -25 DO UP1() PRIO=75
LIN XP3 C_DIS
LIN XP4

```

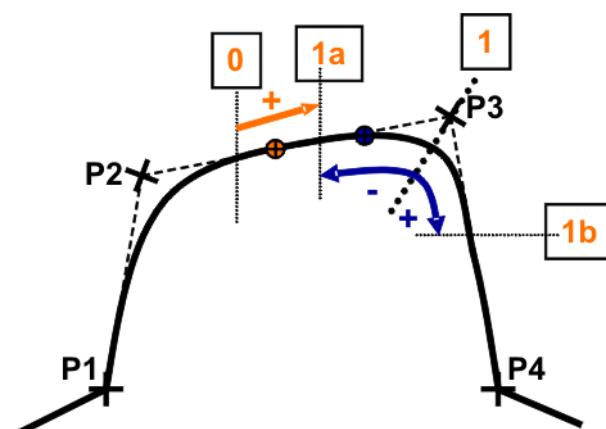


Fig. 8-11: TRIGGER WHEN DISTANCE Ejemplo con posicionamiento aproximado/posicionamiento aproximado

DISTANCE	Rango de conmutación	DELAY
0 (naranja)	entre 0 y 1a	+
1 (azul)	entre 1a y 1b	-/+

8.3 Programación de funciones de conmutación referidas a la trayectoria

TRIGGER WHEN PATH

Descripción de funciones de conmutación referidas a la trayectoria con TRIGGER WHEN PATH

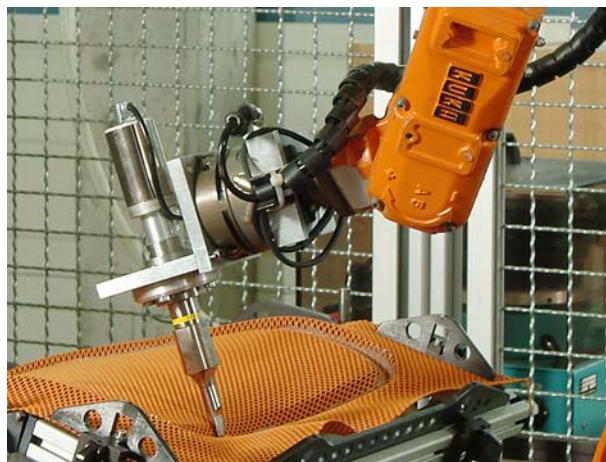


Fig. 8-12: Aplicación de pegamento

- La instrucción de conmutación de trayectoria TRIGGER lanza una instrucción definida
- La instrucción PATH se refiere al punto de destino del paso de movimiento
- La instrucción se ejecuta de forma paralela al movimiento del robot
- Es posible un corrimiento local y/o temporal del punto de conmutación



El punto de destino se debe alcanzar con un movimiento de trayectoria (LIN o CIRC). El movimiento no puede ser PTP.

Funcionamiento de las funciones de conmutación referidas a la trayectoria con TRIGGER WHEN PATH

Sintaxis

- TRIGGER WHEN PATH=*Trayecto* DELAY=*Tiempo* DO *Instrucción* <PRIO=*Prioridad*>
- **Trayecto:** Determina el desplazamiento en el espacio desde el punto de destino.
 - Valor positivo: Desplaza la instrucción hacia el final del movimiento
 - Valor negativo: Desplaza la instrucción hacia el principio del movimiento
 - El trayecto se indica en milímetros (mm)
 - Se pueden realizar indicaciones de trayecto de +/- 10.000.000 mm
 - En caso de indicaciones de valores demasiado extensas o reducidas, se conectarán como máximo o como mínimo en los límites de conmutación
- **Tiempo:** De este modo se define un tiempo de desplazamiento desde la posición seleccionada mediante la indicación PATH.
 - se pueden utilizar valores positivos y negativos
 - la base de tiempo es en milisegundos (ms)
 - se pueden utilizar sin problemas tiempos de hasta 10.000.000 ms
 - en caso de un tiempo demasiado extenso o reducido, se conectarán como máximo o como mínimo en los límites de conmutación
- **Instrucción:**
 - La asignación de valor a una variable.



La asignación de valor se puede realizar a variables de tiempo de ejecución reducidas.

- Instrucción OUT
- Instrucción PULSE
- Llamada de un subprograma. En este caso **se debe** indicar la *Prioridad*.
- **Prioridad** (solo para la llamada de un subprograma):
 - Están disponibles las prioridades 1, 2, 4 - 39 y 81 -128.
 - Las prioridades 40 - 80 están reservadas para casos en los que la prioridad se asigna automáticamente por el sistema. Si la prioridad debe adjudicarla de forma automática el sistema, se programa: PRIO = -1.

Rango de conmutación

- Desplazamiento en dirección al final del movimiento:

Una instrucción puede desplazarse **como máximo hasta el siguiente punto de parada exacta** tras TRIGGER WHEN PATH (a través de todos los puntos de posicionamiento aproximado).

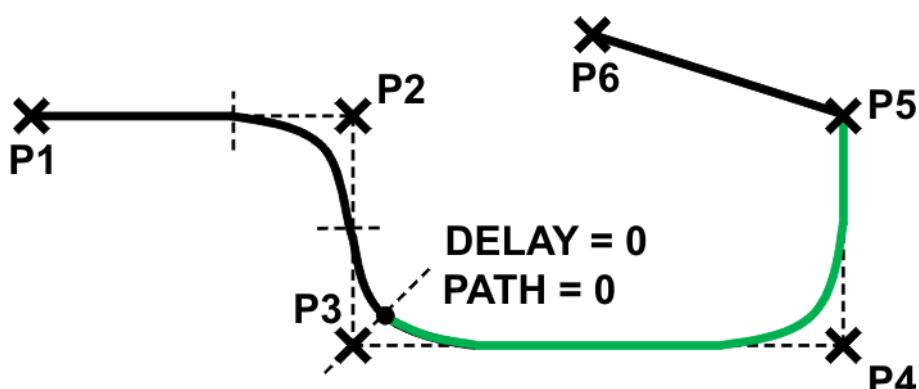


Fig. 8-13: TRIGGER WHEN PATH Límite de conmutación final del movimiento



Esto significa: Si el punto de destino es un punto de parada exacta, la instrucción no se puede desplazar más allá del punto de destino.

- Desplazamiento en dirección al principio del movimiento:

Una instrucción se puede desplazar **como máximo hasta el punto de inicio del paso de movimiento** (es decir, hasta el último punto antes de TRIGGER WHEN PATH).

 - Si el punto de inicio es una parada exacta, la instrucción puede aplazarse como máximo hasta el punto de inicio.

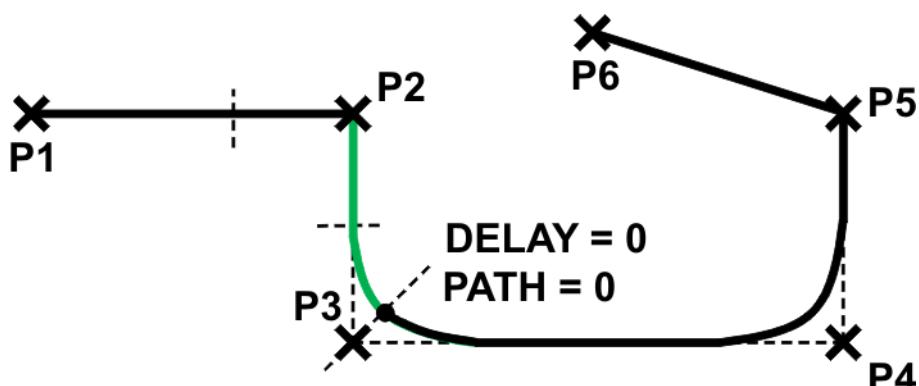


Fig. 8-14: TRIGGER WHEN PATH Límite de conmutación punto de inicio (parada exacta)

- Si el punto de inicio es un punto PTP programado por aproximación, la instrucción puede ser desplazada como máximo hasta el final de su arco de aproximación.

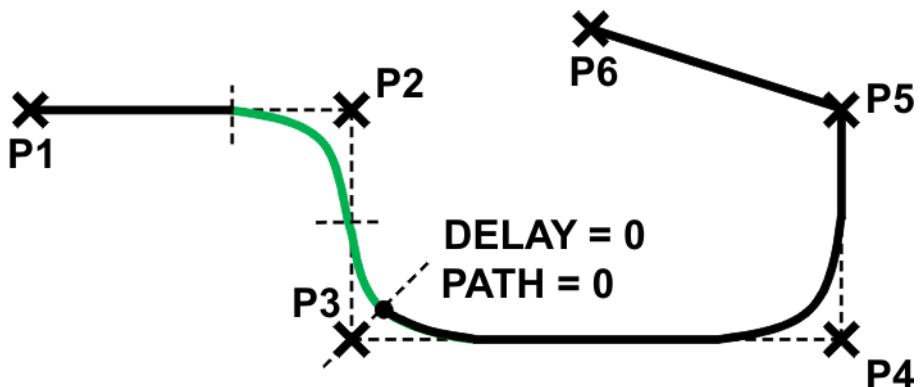


Fig. 8-15: TRIGGER WHEN PATH Límite de comutación punto de inicio (programado por aproximación)

Programación de funciones de comutación referidas a la trayectoria
TRIGGER WHEN PATH

- Comutación en dirección al final del movimiento

```
LIN XP2 C_DIS
TRIGGER WHEN PATH = Y DELAY = X DO $OUT[2] = TRUE
LIN XP3 C_DIS
LIN XP4 C_DIS
LIN XP5
LIN XP6
```

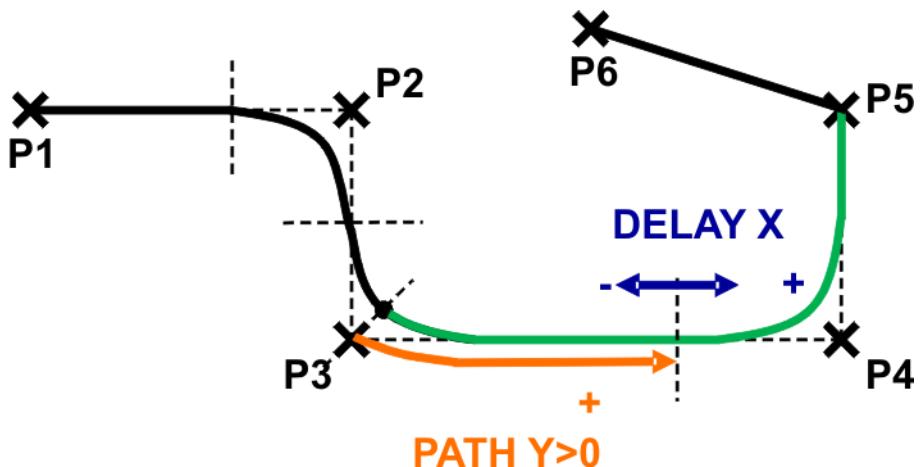


Fig. 8-16: TRIGGER WHEN PATH Comutación en dirección al final del movimiento

- Comutación en dirección al inicio del movimiento

```
LIN XP2 C_DIS
TRIGGER WHEN PATH = Y DELAY = X DO $OUT[2] = TRUE
LIN XP3 C_DIS
LIN XP4 C_DIS
LIN XP5
LIN XP6
```

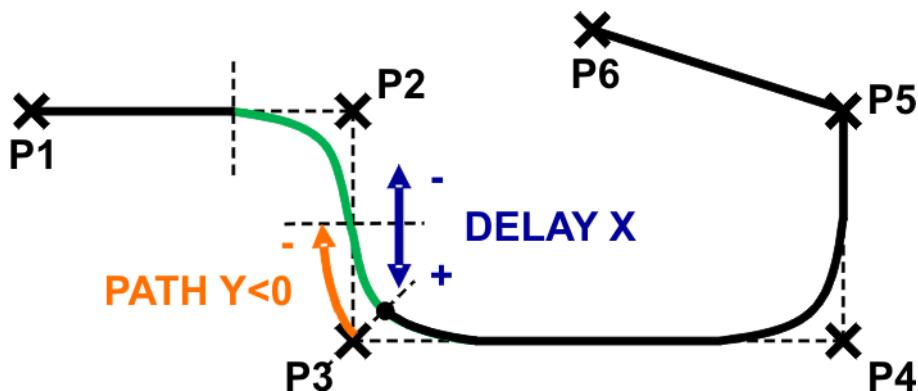


Fig. 8-17: TRIGGER WHEN PATH Comutación en dirección al principio del movimiento

8.4 Ejercicio: Funciones tiempo-distancia en KRL

Objetivo del ejercicio Despues de terminar con éxito este ejercicio, se dispone de la competencia necesaria para efectuar las siguientes tareas:

- Trabajar con la instrucción TRIGGER WHEN DISTANCE ...

Requisitos Las siguientes condiciones son necesarias para efectuar este ejercicio con éxito:

- Conocimientos sobre la instrucción TRIGGER

Tarea Ampliar el programa para el paletizado y el despaletizado con las siguientes funciones:

- Ajustar la salida \$OUT[1] con la garra abierta en el estado "TRUE"; con la garra cerrada el estado debe ser "FALSE".
- Ajustar la salida \$OUT[2] durante el proceso de recogida en el estado "TRUE". Durante el proceso de depósito el estado debe ser "FALSE".
- Ajustar la salida \$OUT[3] durante el paletizado en el estado "TRUE". Durante el despaletizado el estado debe ser "FALSE".
- Tarea adicional: Conectar los temporizadores también a través de la instrucción Trigger.

Lo que deberá saberse ahora:

1. ¿A qué se refiere la instrucción "TRIGGER WHEN PATH"?

.....
.....

2. ¿Qué sector está reservado por el sistema para la asignación automática?

.....
.....

3. ¿Existen limitaciones para la utilización de la instrucción "TRIGGER WHEN PATH"?

.....
.....

4. ¿A qué se refiere la instrucción "TRIGGER WHEN DISTANCE=0"?

.....
.....

9 Programar con WorkVisual

9.1 Gestionar el proyecto con WorkVisual

Proceso del proyecto

1. Cargar el proyecto en WorkVisual desde la unidad de control del robot
(>>> 9.1.1 "Abrir proyecto con WorkVisual" Página 115)
2. Modificar el proyecto, p. ej. programas KRL
(>>> 9.2 "Editar programas KRL con WorkVisual" Página 130)
3. Comparar el proyecto (combinar)
(>>> 9.1.2 "Comparar proyectos con WorkVisual" Página 119)
4. Cargar el proyecto desde WorkVisual a la unidad de control del robot (implantación)
(>>> 9.1.3 "Transmitir el proyecto a la unidad de control del robot (instalar)" Página 123)
5. Activar el proyecto (confirmación)
(>>> 9.1.4 "Activar proyecto en la unidad de control del robot" Página 127)

9.1.1 Abrir proyecto con WorkVisual

Breve descripción de WorkVisual

El paquete de software **WorkVisual** es un entorno de ingeniería para celdas robotizadas controladas por KR C4. Dispone de las siguientes funcionalidades:

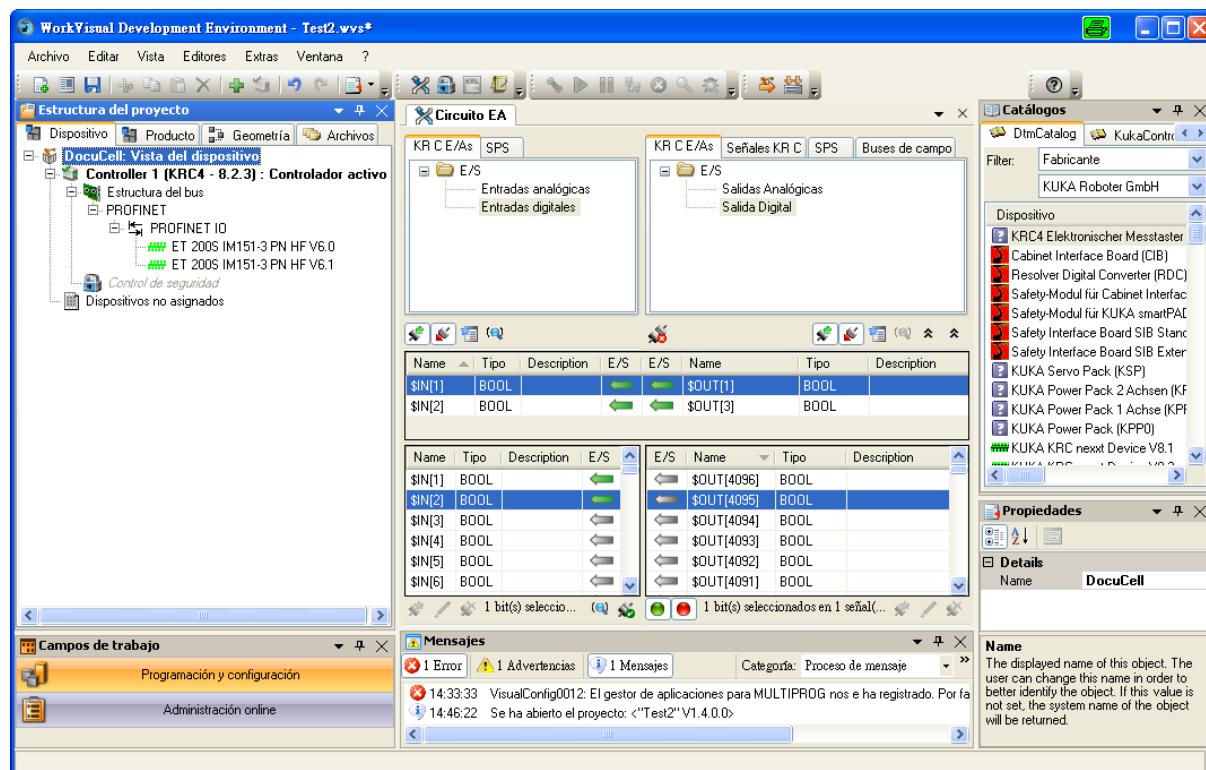


Fig. 9-1: Interfaz de usuario de WorkVisual

- Transmitir proyectos de la unidad de control del robot a WorkVisual
En cada unidad de control del robot hacia la que exista una conexión de red se puede seleccionar cualquier proyecto y transmitirlo a WorkVisual. Esto también es posible aunque el proyecto no esté en el PC.

- Comparar el proyecto con otro proyecto y si es necesario, aceptar las diferencias
- Un proyecto puede compararse con otro proyecto. Este puede ser un proyecto de una unidad de control de robot o un proyecto almacenado localmente. El usuario podrá decidir individualmente para cada diferencia si quiere dejar el estado como en el proyecto actual o si quiere aceptar el estado del otro proyecto.
- Transmitir proyectos a la unidad de control del robot
- Construir y conectar buses de campo
- Editar la configuración de seguridad
- **Programar robot offline**
- Administrar textos largos
- Funcionalidad de diagnóstico
- Visualización online de información del sistema relativa a la unidad de control del robot
- Configurar Traces, iniciar registros, valorar Traces (con osciloscopio)

Estructura y funcionamiento de la interfaz de usuario de WorkVisual

En la interfaz de usuario no están visibles todos los elementos por defecto, sino que se pueden mostrar u ocultar según sea necesario.

Aparte de las ventanas y editores mostrados, se dispone de otros. Pueden activarse de diferentes maneras o a través de las opciones de menú **Ventanas** y **Editores**.

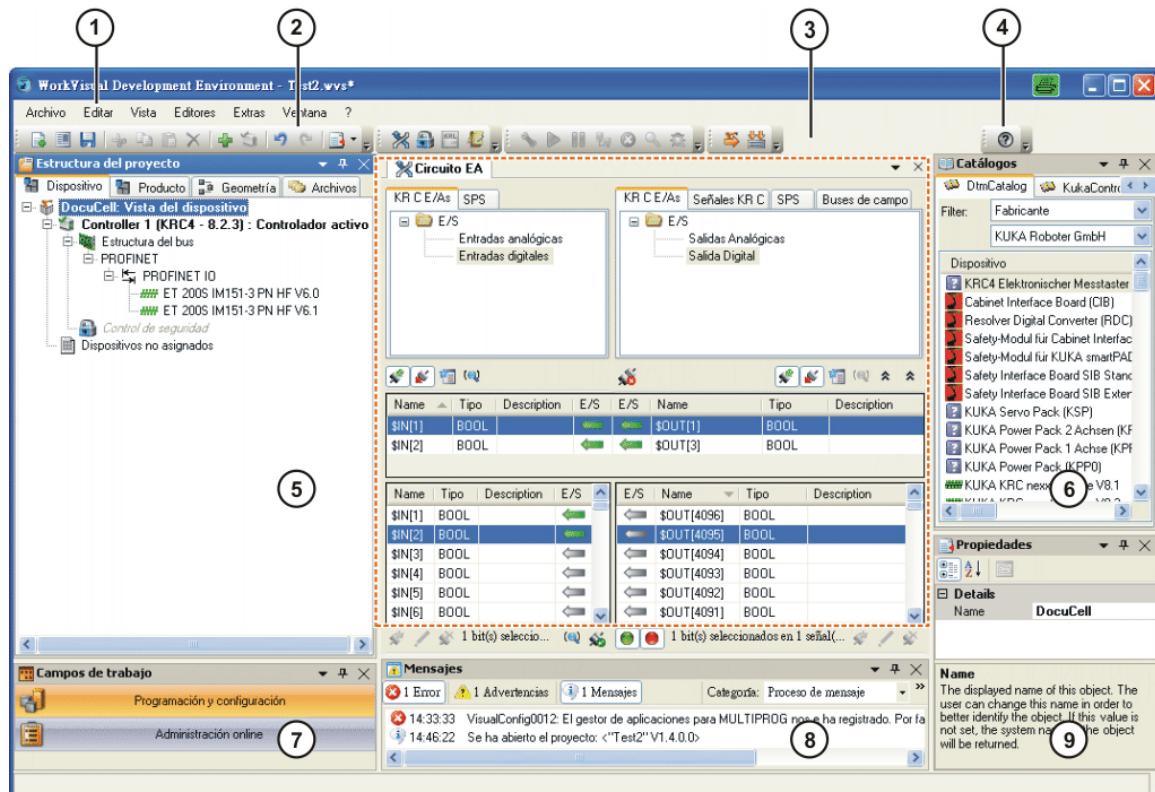


Fig. 9-2: Resumen de la superficie de operación

Pos.	Descripción
1	Barra de menús
2	Barras de botones

Pos.	Descripción
3	Zona de los editores Cuando se abre el editor, se visualiza aquí. Tal y como se muestra en el ejemplo, pueden estar abiertos simultáneamente varios editores. Aparecerán uno encima del otro y se podrán seleccionar a través de las pestañas.
4	Botón de ayuda
5	Ventana "Estructura del proyecto"
6	Ventana Catálogos En esta ventana se muestran todos los catálogos añadidos. Los elementos de los catálogos se pueden añadir a la ventana arrastrando y soltando sobre las pestañas Dispositivos o Geometría .
7	Ventana Campos de trabajo
8	Ventana Mensajes
9	Ventana Propiedades Si se selecciona un objeto se mostrarán sus propiedades en esta ventana. Las propiedades se pueden modificar. No es posible modificar las propiedades de los campos grises.

Ventana Estructura del proyecto

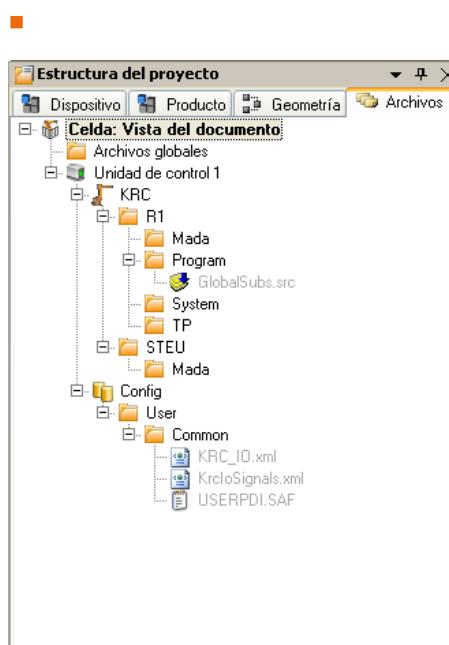


Fig. 9-3: Ejemplo: archivos generados automáticamente en gris

■ **Dispositivos:**

En la pestaña **Dispositivos** se muestra de manera clara la correspondencia de los dispositivos. Aquí se puede asignar una unidad de control del robot a cada dispositivo.

■ **Producto:**

La pestaña **Producto** se utiliza principalmente en WorkVisual Process y no tanto en WorkVisual. Aquí se muestran en estructura de árbol todas las tareas necesarias para un producto.

■ **Geometría:**

La pestaña **Geometría** se utiliza principalmente en WorkVisual Process y no tanto en WorkVisual. Aquí se muestran en estructura de árbol todos los objetos 3D disponibles en el proyecto.

■ **Archivos:**

La pestaña **Archivos** contiene los archivos de programa y configuración del proyecto.

Visualización con colores de los nombres de los archivos:

- Archivos generados automáticamente (con la función **Crear código**): gris
- Archivos añadidos manualmente a WorkVisual: azul
- Archivos transmitidos por una unidad de control del robot a WorkVisual: negro

Explorador de proyecto

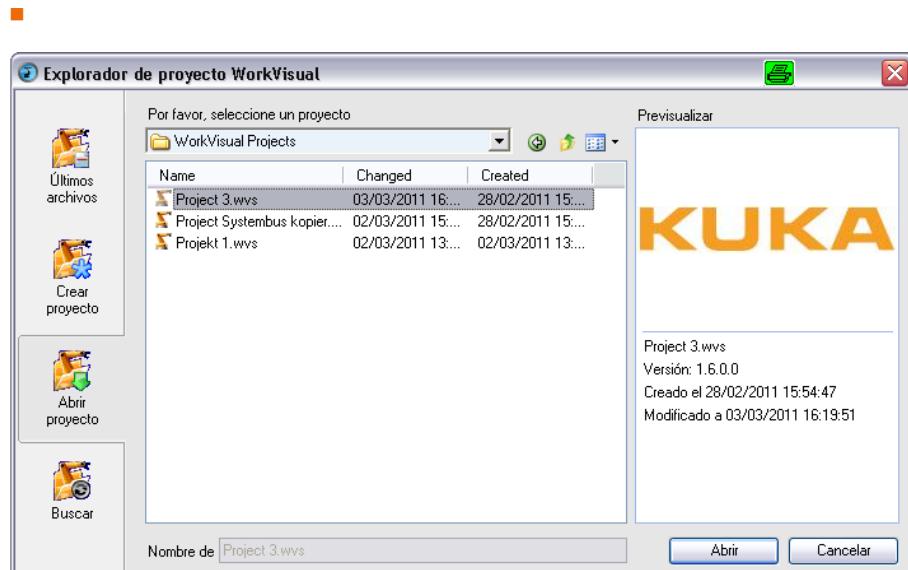


Fig. 9-4: Explorador de proyecto

- En **Últimos archivos** se muestran los últimos utilizados
- Con **Crear proyecto** se genera:
 - un proyecto nuevo y vacío
 - un proyecto nuevo a partir de un modelo
 - un proyecto nuevo basado en un proyecto ya existente
- Con **Abrir proyecto** se abren los proyectos ya existentes
- Con **Buscar** se puede escoger un proyecto de la unidad de control del robot para descargarlo

Procedimiento de carga de un proyecto con WorkVisual

En cada unidad de control del robot **en la que exista una conexión de red** se puede seleccionar un proyecto y transmitirlo a WorkVisual.

Esto también es posible aunque el proyecto no esté en el PC.

El proyecto se guarda en el directorio Mis documentos\WorkVisual Projects\Downloaded Projects.

1. Seleccionar la secuencia de menú **Archivo > Buscar proyecto**. Se abre el **Explorador de proyecto**. A la izquierda está seleccionada la pestaña **Buscar**.
2. En el sector **Celdas disponibles**, desplegar el nodo de la celda deseada. Se muestran todas las unidades de control del robot de dicha celda.
3. Desplegar los nodos de la unidad de control del robot deseada. Se muestran todos los proyectos.
4. Marcar el proyecto deseado y hacer clic en **Abrir**. El proyecto se abre en WorkVisual.

9.1.2 Comparar proyectos con WorkVisual

Descripción

- Es posible comparar un proyecto en WorkVisual con otro proyecto
- Este puede ser un proyecto de una unidad de control de robot o un proyecto almacenado localmente.
- Las diferencias aparecen de forma clara en una lista y puede mostrarse información detallada.
- El usuario puede decidir individualmente para cada diferencia
 - si quiere dejar el estado como en el proyecto actual
 - o si quiere aceptar el estado del otro proyecto

Principio de comparación de proyectos

Unir proyectos

-

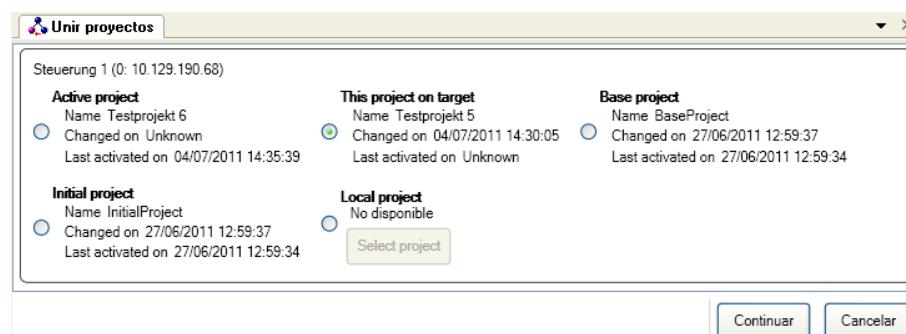


Fig. 9-5: Seleccionar el proyecto para "Unir"

- Proyecto activo
- Proyecto con el mismo nombre en el controlador (únicamente posible si se dispone de conexión de red)
- Proyecto base
- Proyecto inicial
- Proyecto local (del ordenador portátil)

Comparación

Las diferencias entre los proyectos se muestran en un resumen. Para cada una de las diferencias puede seleccionarse el estado que debe asumirse.

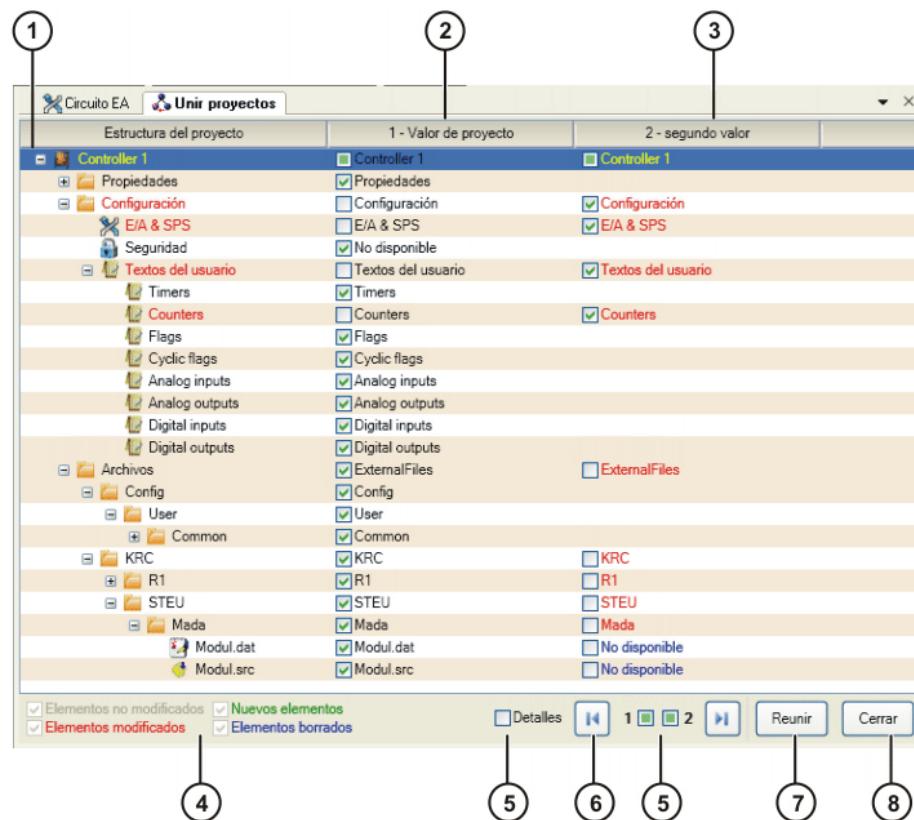


Fig. 9-6: Ejemplo: resumen de las diferencias

Pos.	Descripción
1	<p>El nodo de la unidad de control del robot. Las diferentes áreas del proyecto se visualizan en subnodos. Abrir los nodos para mostrar las comparaciones.</p> <p>Si hay disponibles varias unidades de control del robot estas se listan una debajo de la otra.</p> <ul style="list-style-type: none"> ■ En una línea, colocar siempre la marca de verificación en el valor que se quiere tomar. (Alternativa: emplear las casillas al pie de página). ■ Un símbolo de confirmación activado en No disponible significa que el elemento no se aceptará o que, en caso de estar ya disponible, se borrará del proyecto. ■ Si se coloca un símbolo de confirmación en un nodo, se marcarán automáticamente también todos los elementos subordinados. <p>Si se quita un símbolo de confirmación en un nodo, se quitarán automáticamente también en todos los elementos subordinados.</p> <p>Los elementos subordinados también se pueden editar individualmente.</p> <ul style="list-style-type: none"> ■ Una casilla con contenido significa que, de los elementos subordinados, al menos uno está seleccionado.
2	Estado en el proyecto abierto en WorkVisual
3	Estado en el proyecto de comparación
4	Filtros para mostrar u ocultar los diversos tipos de diferencias
5	TRUE: En la vista general se muestran informaciones detalladas de las líneas marcadas.
6	<p>Flecha de retroceso: el indicador pasa a la diferencia anterior.</p> <p>Flecha de avance: el indicador pasa a la diferencia siguiente.</p> <p>Los nodos cerrados se abren automáticamente.</p>
7	Las casillas muestran el estado de la línea marcada. En lugar de hacerse directamente en la línea, los símbolos de confirmación también se pueden colocar aquí.

Pos.	Descripción
8	Confirma las modificaciones seleccionadas en el proyecto abierto.
9	Cierra la ventana Unir proyectos .

Descripción de los colores:

Columna	Descripción
Estructura del proyecto	Cada elemento se muestra con el color que tiene en la columna desde la que se ha seleccionado.
Valor actual del proyecto (1)	Todos los elementos se muestran en negro.
Valor de comparación (2)	<ul style="list-style-type: none"> ■ Verde: elementos que no están disponibles en el proyecto abierto, pero sí en el proyecto de comparación. ■ Azul: elementos que están disponibles en el proyecto abierto, pero no en el proyecto de comparación. ■ Rojo: el resto de elementos. También se incluyen los elementos superiores, los cuales contienen otros elementos en diferentes colores.

Procedimiento para la comparación de proyectos

1. En WorkVisual, seleccionar la secuencia de menú **Extras > Comparar proyectos**. Se abre la ventana **Comparar proyectos**.

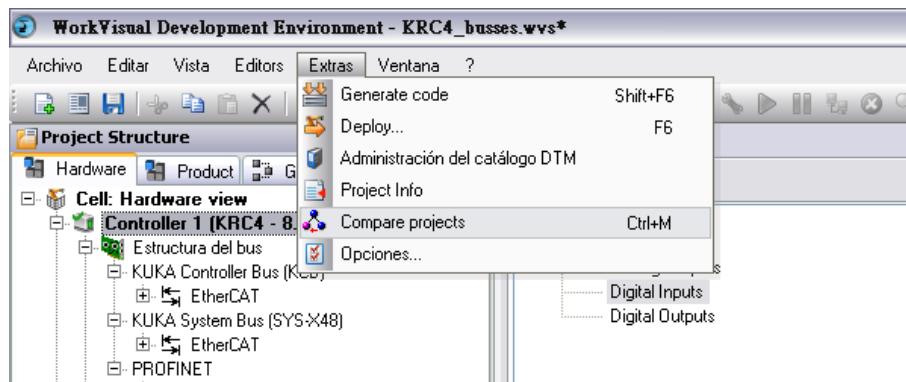


Fig. 9-7

2. Seleccionar el proyecto con el que debe compararse el proyecto WorkVisual actual, p. ej. el proyecto con el mismo nombre de la unidad de control real del robot.

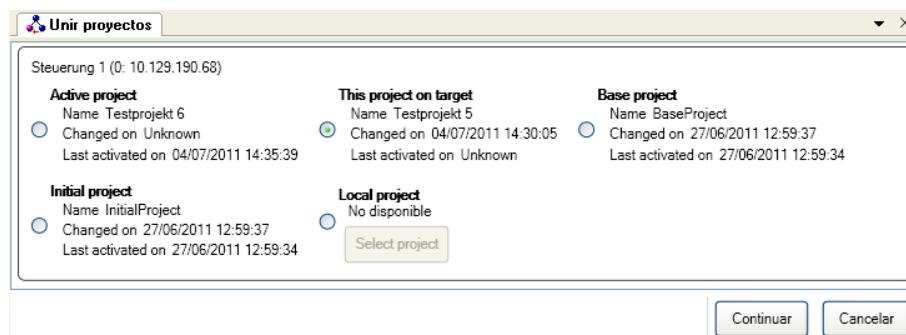


Fig. 9-8: Seleccionar el proyecto para "Unir"

3. Hacer clic en **Continuar**. Se muestra una barra de progreso. (Si el proyecto contiene varias unidades de control del robot, se muestra una barra para cada una.)
4. Cuando la barra de progreso se completa y además se muestra el estado: **Preparado para la conexión**: Hacer clic en **Mostrar diferencias**. Las diferencias entre los proyectos se muestran en un resumen.

Si no se detectan diferencias se indicará en la ventana de mensajes. Se seguir con el paso 8. Después no serán necesarios más pasos.

5. Para cada una de las diferencias puede seleccionarse el estado que debe adoptarse. Esto no tiene que hacerse para todas las diferencias de un paso.
Si es adecuado, puede dejarse también la selección por defecto.
6. Pulsar **Reunir** para aceptar los cambios en WorkVisual.
7. Repetir los pasos 5 y 6 las veces que se desee. Esto permite la edición progresiva de las diferentes zonas.
Si no hay más diferencias se mostrará el siguiente mensaje: **No existen más diferencias**.
8. Cerrar la ventana **Comparar proyectos**.
9. Si se han modificado parámetros de los ejes adicionales en el proyecto de la unidad de control del robot, estos deberán ser actualizados en WorkVisual:
 - Abrir la ventana **Configuración de los datos de máquina** para este eje adicional.
 - En el área **Datos de máquina generales específicos del eje** pulsar sobre el botón para la importación de los datos de la máquina.
 Se actualizan los datos.
10. Guardar el proyecto.

Ejemplo de una comparación de proyectos

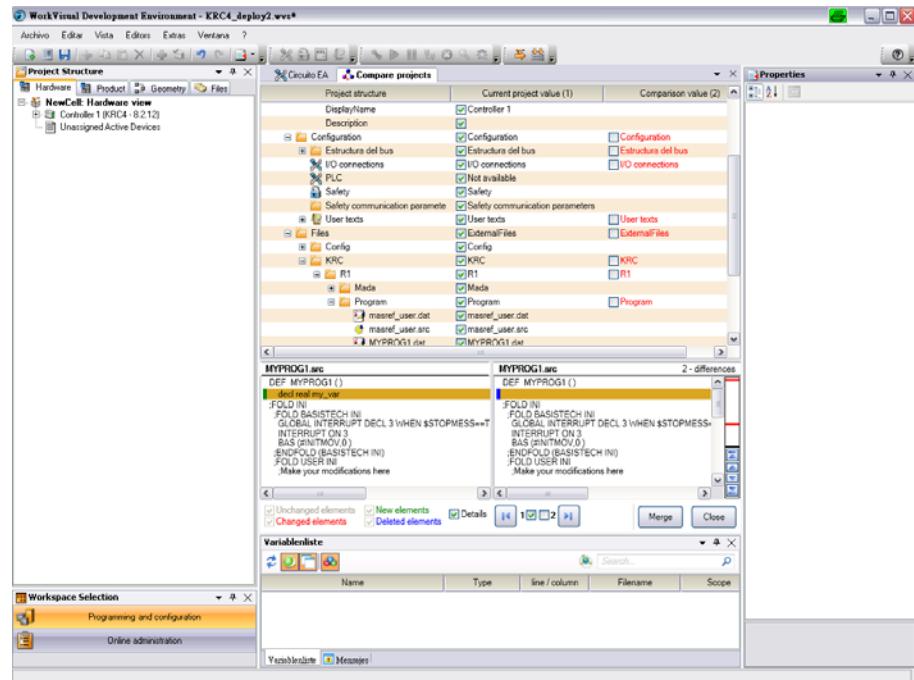


Fig. 9-9: Ejemplo: Comparación de proyectos, vista general

Decisión sobre qué estado del archivo(s) debe aceptarse

Project structure	Current project value (1)	Comparison value (2)
Controller 1	Controller 1	<input checked="" type="checkbox"/> Controller 1 - same identifier
Properties	Properties	
Identifier	<input checked="" type="checkbox"/> S-1-5-21-468942068-2137545632-	
Serial number:	<input checked="" type="checkbox"/>	
BaseAddress	<input checked="" type="checkbox"/> 10.129.190.34	
FirmwareVersion	<input checked="" type="checkbox"/> 8.2.12	
Language	<input checked="" type="checkbox"/> KRL	
ControllerType	<input checked="" type="checkbox"/> KRC4	
DisplayName	<input checked="" type="checkbox"/> Controller 1	
Description	<input checked="" type="checkbox"/>	
Configuration	Configuration	<input checked="" type="checkbox"/> Configuration
Estructura del bus	<input checked="" type="checkbox"/> Estructura del bus	<input type="checkbox"/> Estructura del bus
I/O connections	<input checked="" type="checkbox"/> I/O connections	<input type="checkbox"/> I/O connections
PLC	<input checked="" type="checkbox"/> Not available	
Safety	<input checked="" type="checkbox"/> Safety	
Safety communication parameters	<input checked="" type="checkbox"/> Safety communication parameters	
User texts	<input checked="" type="checkbox"/> User texts	<input type="checkbox"/> User texts
Files	ExternalFiles	<input type="checkbox"/> ExternalFiles
Config	<input checked="" type="checkbox"/> Config	
KRC	<input checked="" type="checkbox"/> KRC	<input type="checkbox"/> KRC
R1	<input checked="" type="checkbox"/> R1	<input type="checkbox"/> R1
Mada	<input checked="" type="checkbox"/> Mada	

Fig. 9-10: Ejemplo: Unir proyectosAl activar los **Details** pueden mostrarse las diferencias de los archivos

```

MYPROG1.src
DEF MYPROG1 ()
;Decl real my_var
;FOLDINI
;FOLD BASISTECHINI
GLOBAL INTERRUPT DECL 3 WHEN $STOPMESS==TRUE DO I
INTERRUPT ON 3
BAS (#INITMOV,0)
;ENDFOLD (BASISTECHINI)
;FOLD USERINI
;Make your modifications here

;ENDFOLD (USERINI)
;ENDFOLD (INI)

;FOLD PTP HOME Vel= 100 % DEFAULT;%(PE)%MKUKATPBASIS
$BWDSTART = FALSE
PDAT_ACT=PDEFAULT
FDAT_ACT=FHOME
BAS (#PTP_PARAMS,100)
SH_POS=XHOME
PTP_XHOME
;ENDFOLD
;wait sec 2
;FOLD PTP HOME Vel= 100 % DEFAULT;%(PE)%MKUKATPBASIS
$BWDSTART = FALSE
PDAT_ACT=PDEFAULT
FDAT_ACT=FHOME
BAS (#PTP_PARAMS,100)
SH_POS=XHOME
PTP_XHOME
;ENDFOLD

MYPROG1.src
2 - differences
DEF MYPROG1 ()
;FOLDINI
;FOLD BASISTECHINI
GLOBAL INTERRUPT DECL 3 WHEN $STOPMESS==TRUE D
INTERRUPT ON 3
BAS (#INITMOV,0)
;ENDFOLD (BASISTECHINI)
;FOLD USERINI
;Make your modifications here

;ENDFOLD (USERINI)
;ENDFOLD (INI)

;FOLD PTP HOME Vel= 100 % DEFAULT;%(PE)%MKUKATPBASIS
$BWDSTART = FALSE
PDAT_ACT=PDEFAULT
FDAT_ACT=FHOME
BAS (#PTP_PARAMS,100)
SH_POS=XHOME
PTP_XHOME
;ENDFOLD
;wait sec 2
;FOLD PTP HOME Vel= 100 % DEFAULT;%(PE)%MKUKATPBASIS
$BWDSTART = FALSE
PDAT_ACT=PDEFAULT
FDAT_ACT=FHOME
BAS (#PTP_PARAMS,100)
SH_POS=XHOME
PTP_XHOME
;ENDFOLD

```

Unchanged elements New elements Details 1 2 Merge Close

Fig. 9-11: Ejemplo: Detalles activados

9.1.3 Transmitir el proyecto a la unidad de control del robot (instalar)

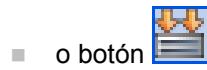
Descripción

- Tras realizar modificaciones en el proyecto, este se envía desde WorkVisual a la unidad de control.
- Para KUKA, este proceso se denomina "**Deployment**".
- El código se genera previamente a la transmisión de proyecto a la unidad de control del robot.
- La conexión de red con la unidad de control del robot real es un requisito para esta "**Implantación**".

i Si en la unidad de control del robot real existe un proyecto que ha sido transmitido en un momento anterior y no se ha activado nunca, este es sobreescrito por causa de la transmisión de otro proyecto. Mediante la transmisión y activación de un proyecto, se sobrescribe un proyecto con mismo nombre existente en la unidad de control del robot (tras una consulta de seguridad).

Funciones**Crear código**

- Con este procedimiento se puede crear por separado el código y comprobar por adelantado si está libre de errores.
- Se accede a través de
 - Secuencia de menú **Extras > Crear código**.



- o botón
- El código aparece en la ventana **Estructura del proyecto** en la pestaña **Archivos**.

El código generado automáticamente se muestra en color gris claro.

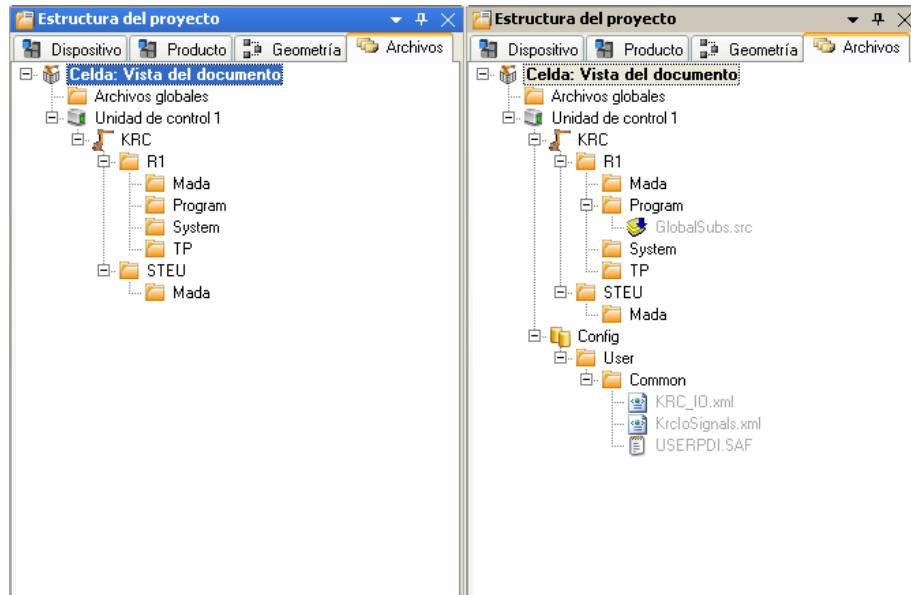


Fig. 9-12: Ejemplo de creación de código: antes y después

- Se crea el código. Cuando ha finalizado el proceso, en la ventana de mensajes se muestran los siguientes mensajes: **Se ha compilado el proyecto <"{0}" V{1}>. Los resultados están disponibles en el árbol de archivos.**

Instrucciones de actuación

1. En la barra de menús hacer clic en el botón **Instalar ...** Se abre la ventana **Transmisión del proyecto**.

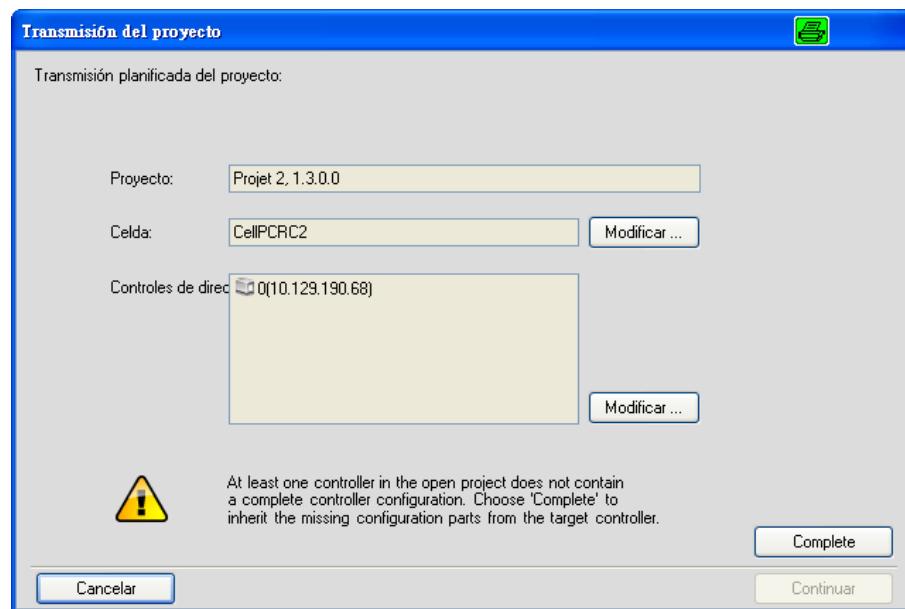


Fig. 9-13: Resumen indicando la configuración incompleta

2. Cuando se trata de un proyecto, que nunca fue retransferido a WorkVisual por parte de una unidad de control de robot, todavía no contiene todos los ficheros de configuración. Se muestra una indicación correspondiente. (Entre los archivos de configuración se incluyen archivos de datos de la máquina, archivos de la configuración de seguridad y muchos otros).
 - Cuando esa indicación no se muestre: Continuar en el paso 13.
 - Cuando esa indicación se muestre: Continuar en el paso 3.
3. Hacer clic en **Completar**. Aparece la siguiente pregunta de seguridad: **Configurar el proyecto y restablecer el controlador activo. ¿Desea continuar?**
4. Responder **Sí** a la pregunta. Se abre la ventana **Unir proyectos**.

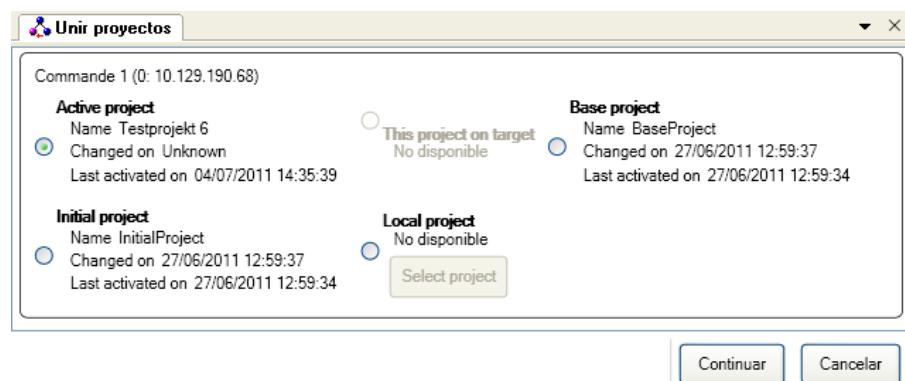


Fig. 9-14: Seleccionar el proyecto para "Completar"

5. Seleccionar un proyecto cuyos datos de configuración deben adoptarse, p. ej.: el proyecto activo en la unidad de control real del robot.
6. Hacer clic en **Continuar**. Se muestra una barra de progreso. (Si el proyecto contiene varias unidades de control del robot, se muestra una barra para cada una.)
7. Cuando la barra de progreso se completa y además se muestra el estado: **Preparado para la conexión**: Hacer clic en **Mostrar diferencias**. Las diferencias entre los proyectos se muestran en un resumen.

8. Para cada una de las diferencias puede seleccionarse el estado que debe adoptarse. Esto no tiene que hacerse para todas las diferencias de un paso.
Si es adecuado, puede dejarse también la selección por defecto.
9. Pulsar **Reunir** para aceptar los cambios.
10. Repetir los pasos 8 y 9 las veces que se desee. Esto permite la edición progresiva de las diferentes zonas.
Si no hay más diferencias se mostrará el siguiente mensaje: **No existen más diferencias**.
11. Cerrar la ventana **Comparar proyectos**.
12. En la barra de menús hacer clic en el botón **Instalar...**. Aparece de nuevo el resumen sobre la asignación de celdas. No aparecerá más la indicación relativa a una configuración incompleta.

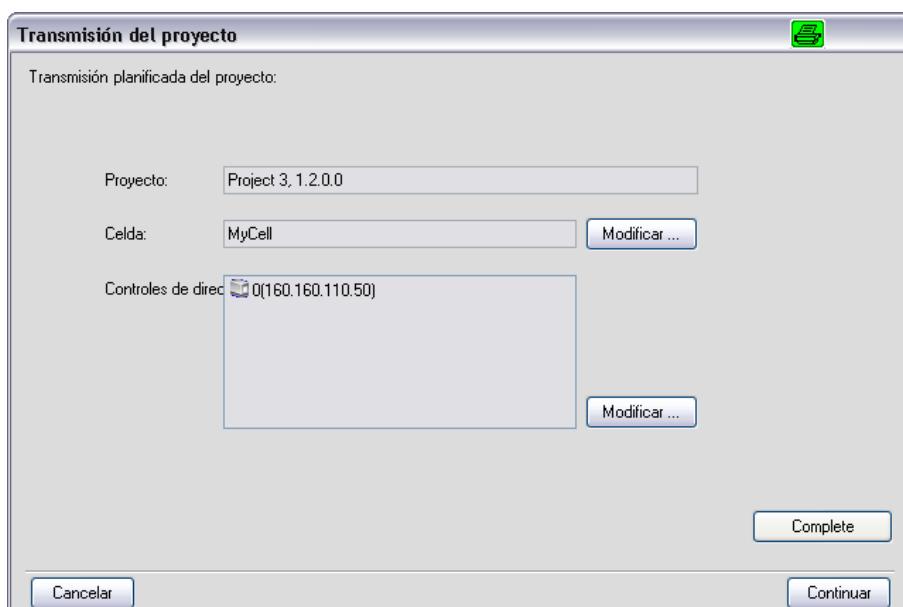


Fig. 9-15: Vista general

13. Hacer clic en **Continuar**. Inicia la creación del programa. Cuando el indicador del proceso llegue al 100% se habrá creado el programa y se habrá transmitido el proyecto.
14. Hacer clic en **Activar**.

ADVERTENCIA En los modos de servicio AUT y AUT EXT, el proyecto se activa sin pregunta de seguridad si solo se trata de modificaciones del programa.

15. Exclusivo para los modos de servicio T1 y T2: KUKA smartHMI muestra la pregunta de seguridad *¿Quiere permitir la activación del proyecto [...]?*. Además se indica si con la activación se ha sobrescrito un proyecto, y en caso afirmativo, cuál.
Si no se ha sobrescrito ningún proyecto relevante: Confirmar la pregunta en el plazo de 30 minutos con **Sí**.
16. Se muestra una vista general de los cambios que se han realizado en la unidad de control del robot en comparación con el proyecto aún activo. Mediante la casilla **Detalles** se pueden mostrar detalles de las modificaciones.
17. La vista general muestra la pregunta de seguridad *¿Desea continuar?*. Responder con **Sí**. El proyecto se activa en la unidad de control del robot. En WorkVisual se visualiza una confirmación.

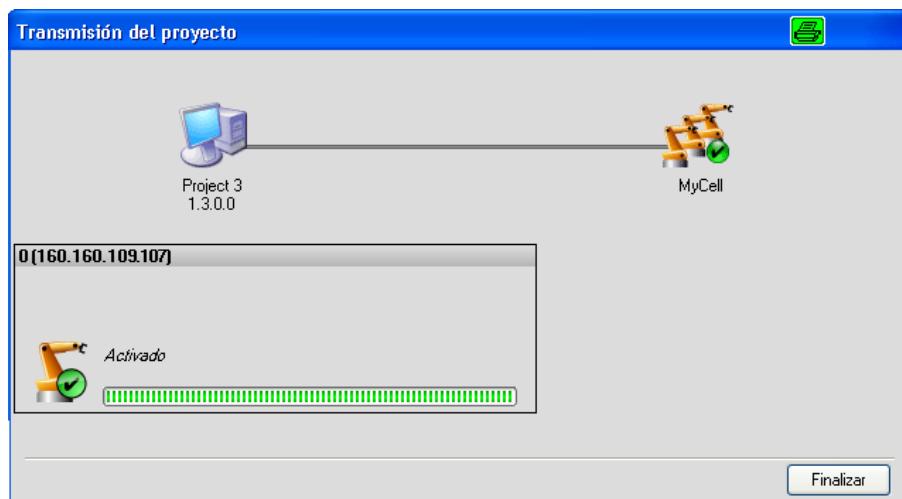


Fig. 9-16: Confirmación en WorkVisual

18. Cerrar la ventana con **Transmisión del proyecto** con **Finalizar**.
19. Si la pregunta no se responde en la unidad de control del robot en 30 minutos, el proyecto se transmitirá de todas formas, aunque no estará activo en la unidad de control del robot. Se puede activar el proyecto por separado.

⚠️ ADVERTENCIA

Tras la activación de un proyecto en la unidad de control del robot se debe comprobar la configuración de seguridad. De lo contrario, es posible que el robot reciba datos erróneos. Esto puede provocar la muerte, lesiones graves o daños materiales importantes.

⚠️ ADVERTENCIA

Si la activación de un proyecto falla aparece un mensaje de error en WorkVisual. En tal caso deben tomarse las siguientes medidas:

- Activar nuevamente un proyecto. (El mismo u otro distinto).
- O arrancar la unidad de control del robot con un arranque en frío.

9.1.4 Activar proyecto en la unidad de control del robot

Descripción	<ul style="list-style-type: none"> ■ Un proyecto puede activarse directamente en la unidad de control del robot. ■ Un proyecto también puede activarse desde WorkVisual en la unidad de control del robot. (no se explica aquí, para ello utilizar la documentación en línea de WorkVisual)
Función de la gestión del proyecto	<p>Generalidades</p> <ul style="list-style-type: none"> ■ La unidad de control del robot tiene la posibilidad de gestionar varios proyectos en la unidad de control ■ Todas las funciones solo están disponibles en el grupo de usuario Experto ■ Puede accederse mediante: <ul style="list-style-type: none"> ■ Secuencia de menú Archivo > Gestión del proyecto ■ En la interfaz de usuario, a través de la tecla WorkVisual-Symbol y a continuación el botón Abrir

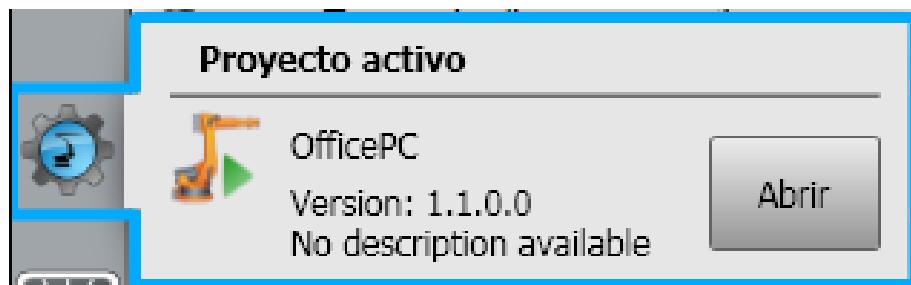


Fig. 9-17: Visualización del proyecto en la interfaz de usuario

Manejo / servicio

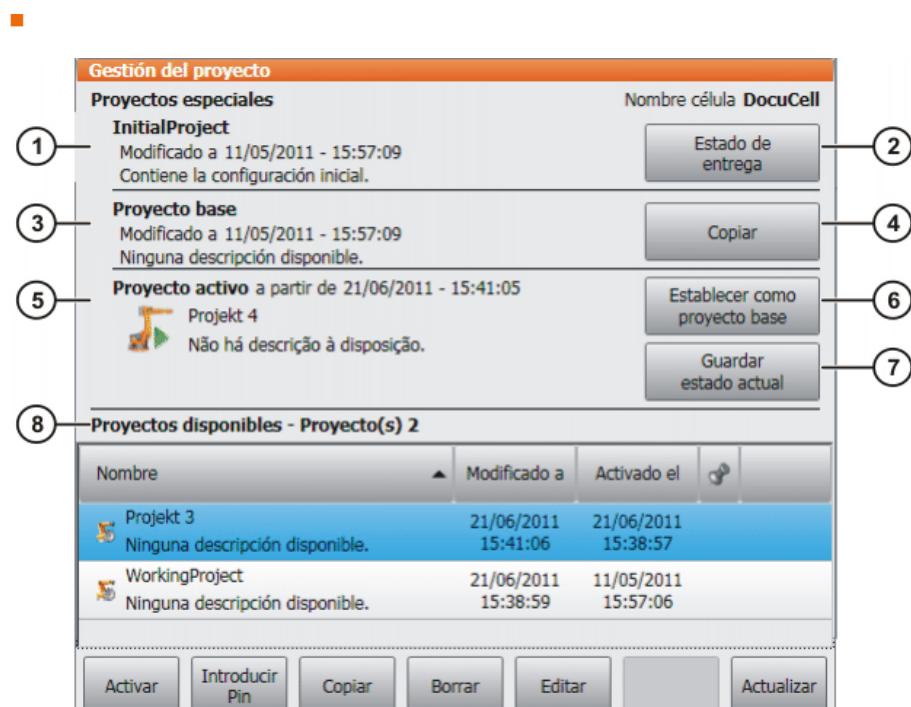


Fig. 9-18: Ventana Gestión del proyecto

Pos.	Descripción
1	Se muestra el proyecto inicial.
2	Restaura el estado de suministro de la unidad de control del robot. Está disponible desde el grupo de usuario Experto.
3	Se muestra el proyecto base.
4	Crea una copia del proyecto base. Está disponible desde el grupo de usuario Experto.
5	Se muestra el proyecto activo.
6	Guarda el proyecto activo como proyecto base. El proyecto activo permanece activo. Está disponible desde el grupo de usuario Experto.
7	Crea una copia bloqueada del proyecto activo. Está disponible desde el grupo de usuario Experto.
8	Lista de los proyectos. Aquí no se muestra el proyecto activo.

- Además de los proyectos regulares, la ventana **Gestión del proyecto** contiene los siguientes proyectos especiales:

Proyecto	Descripción
Proyecto inicial	El proyecto inicial está siempre disponible. No puede ser modificado por el usuario. Contiene el estado de la unidad de control del robot en el momento del suministro.
Proyecto base	<p>El usuario puede guardar el proyecto activo como proyecto base. Esta funcionalidad se utiliza por regla general para asegurar un proyecto protegido y en condiciones de funcionamiento.</p> <p>El proyecto base no puede activarse pero sí copiarse. El proyecto base ya no puede ser modificado por el usuario. Sin embargo, puede sobrescribirse grabando un nuevo proyecto base (tras una pregunta de seguridad).</p> <p>Cuando se activa un proyecto que no contiene todos los archivos de configuración, la información que falta se toma del proyecto base. Puede ser el caso p. ej. cuando se activa un proyecto de una versión anterior de WorkVisual. (Entre los archivos de configuración se incluyen archivos de datos de la máquina, archivos de la configuración de seguridad y muchos otros).</p>

■ Descripción de los botones

Botón	Descripción
Activar	<p>Activa el proyecto marcado.</p> <p>Si el proyecto marcado está bloqueado: Crea una copia del proyecto marcado. (Un proyecto bloqueado no puede activarse, solo una copia del mismo.) El usuario puede decidir si la copia debe activarse ya o el proyecto actual debe quedar activo.</p> <p>Está disponible desde el grupo de usuario Experto.</p>
Introducir Pin	<p>Los proyectos bloqueados no pueden ser modificados, activados o borrados. Sin embargo, pueden copiarse o desbloquearse. Así, se puede bloquear un proyecto p. ej. para evitar borrarlo por error.</p> <p>Sólo está disponible si está marcado un proyecto bloqueado. Está disponible desde el grupo de usuario Experto.</p>
Desbloquear	<p>Desbloquea el proyecto.</p> <p>Sólo está disponible si está marcado un proyecto bloqueado. Está disponible desde el grupo de usuario Experto.</p>
Copiar	<p>Copia el proyecto marcado.</p> <p>Está disponible desde el grupo de usuario Experto.</p>
Borrar	<p>Borra el proyecto marcado.</p> <p>Sólo está disponible si está marcado un proyecto inactivo y desbloqueado. Está disponible desde el grupo de usuario Experto.</p>

Botón	Descripción
Editar	Abre una ventana en la que pueden modificarse el nombre y/o la descripción del proyecto marcado. Sólo está disponible si está marcado un proyecto desbloqueado. Está disponible desde el grupo de usuario Experto.
Actualizar	Actualiza la lista de proyectos. De esta forma se muestran, p. ej., los proyectos que se han transmitido a la unidad de control del robot desde la apertura del indicador.

Procedimiento



Restricción: Cuando la activación causa modificaciones en el sector **Parámetros de comunicación** de la configuración de seguridad, debe seleccionarse el grupo de usuarios Técnico de mantenimiento de seguridad o superior.
Si se ha seleccionado el modo de servicio AUT o AUT EXT: El proyecto sólo se puede activar si únicamente se modifican programas KRL. Si el proyecto contiene ajustes que provocan otros cambios no se puede activar.

1. Seleccionar la secuencia de menú **Archivo > Gestión del proyecto**. Se abre la ventana **Gestión del proyecto**.
2. Marcar el proyecto deseado y activar pulsando el botón **Activar**.
3. KUKA smartHMI muestra la pregunta de seguridad *¿Quiere permitir la activación del proyecto [...]?*. Además se indica si con la activación se ha sobrescrito un proyecto, y en caso afirmativo, cuál.
Si no se ha sobrescrito ningún proyecto relevante: Confirmar la pregunta en el plazo de 30 s con **Sí**.
4. Se muestra una vista general de los cambios que se han realizado en la unidad de control del robot en comparación con el proyecto aún activo. Mediante la casilla de control **Detalles** se pueden mostrar detalles de las modificaciones.
5. La vista general muestra la pregunta de seguridad *¿Desea continuar?*. Responder con **Sí**. El proyecto se activa en la unidad de control del robot.



ADVERTENCIA

Tras la activación de un proyecto en la unidad de control del robot se debe comprobar la configuración de seguridad. De lo contrario, es posible que el robot reciba datos erróneos. Esto puede provocar la muerte, lesiones graves o daños materiales importantes.

9.2 Editar programas KRL con WorkVisual

- Manipulación de archivos
(>>> 9.2.1 "Manipulación de ficheros" Página 130)
- Manejo del editor KRL
(>>> 9.2.2 "Manejo del editor KRL" Página 136)

9.2.1 Manipulación de ficheros

Descripción

- Cargar ficheros disponibles en el editor
- Añadir un fichero nuevo del catálogo
- Añadir ficheros externos

Principio de las plantillas procedentes del catálogo

- Antes de la utilización de las plantillas se deberá cargar el catálogo correspondiente
- Cargar a través de **Archivo > Añadir catálogo** y activar la plantilla correspondiente
 - Plantillas KRL: *KRL Templates.afc*
 - Plantillas VW: *VW Templates.afc*
- Catálogo de plantillas KRL

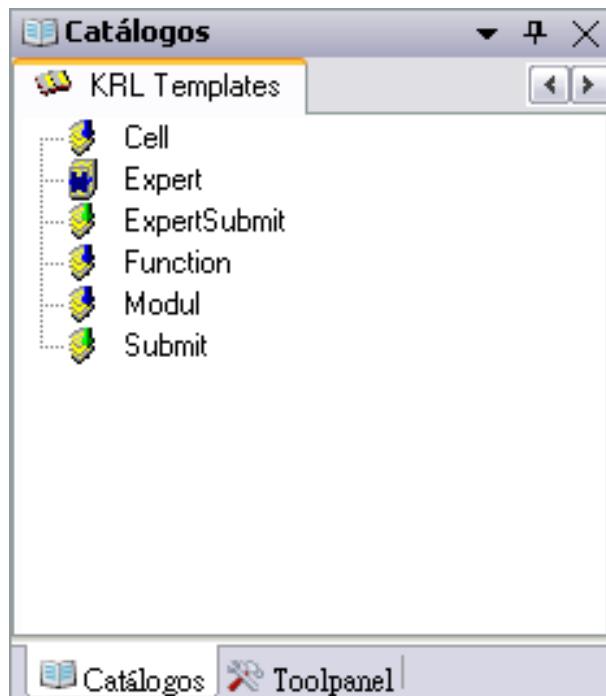


Fig. 9-19: Catálogo de plantillas KRL

Instrucciones de actuación

Procedimiento para abrir un fichero (SRC/DAT) en el editor KRL

1. Cambiar el árbol de proyecto del fichero

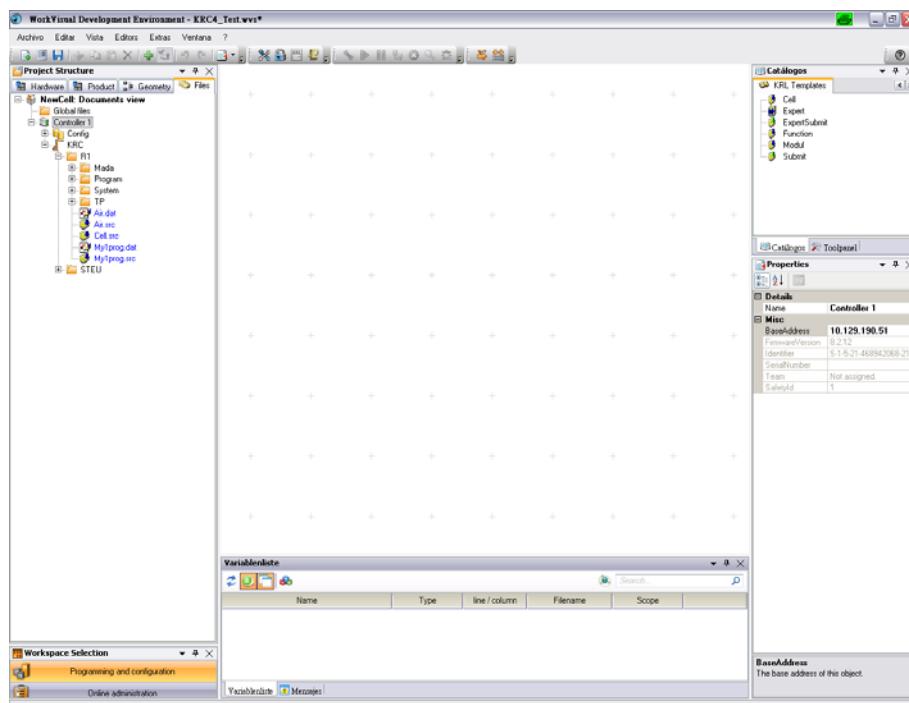


Fig. 9-20: Árbol de proyecto de WorkVisual

2. Desplegar los directorios hasta el directorio R1

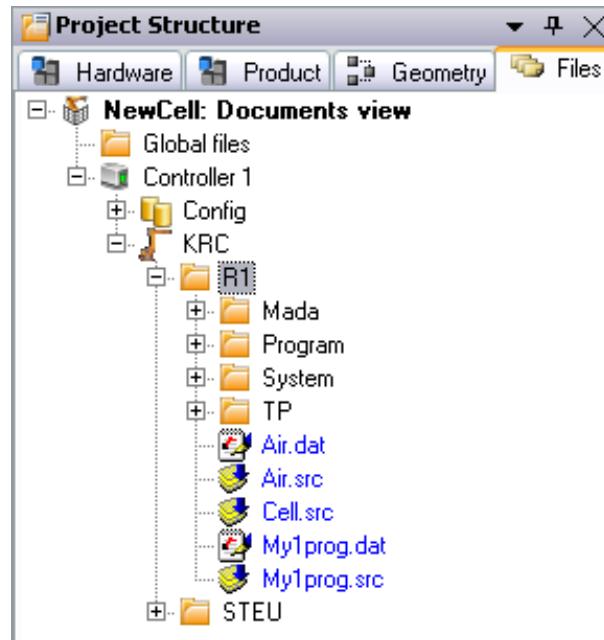
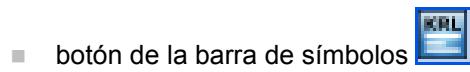


Fig. 9-21: Estructura de ficheros de árbol de proyecto de WorkVisual (R1)

3. Seleccionar el fichero o

- mediante doble clic



- botón de la barra de símbolos



- hacer clic con el botón derecho y seleccionar editor Krl en el menú contextual

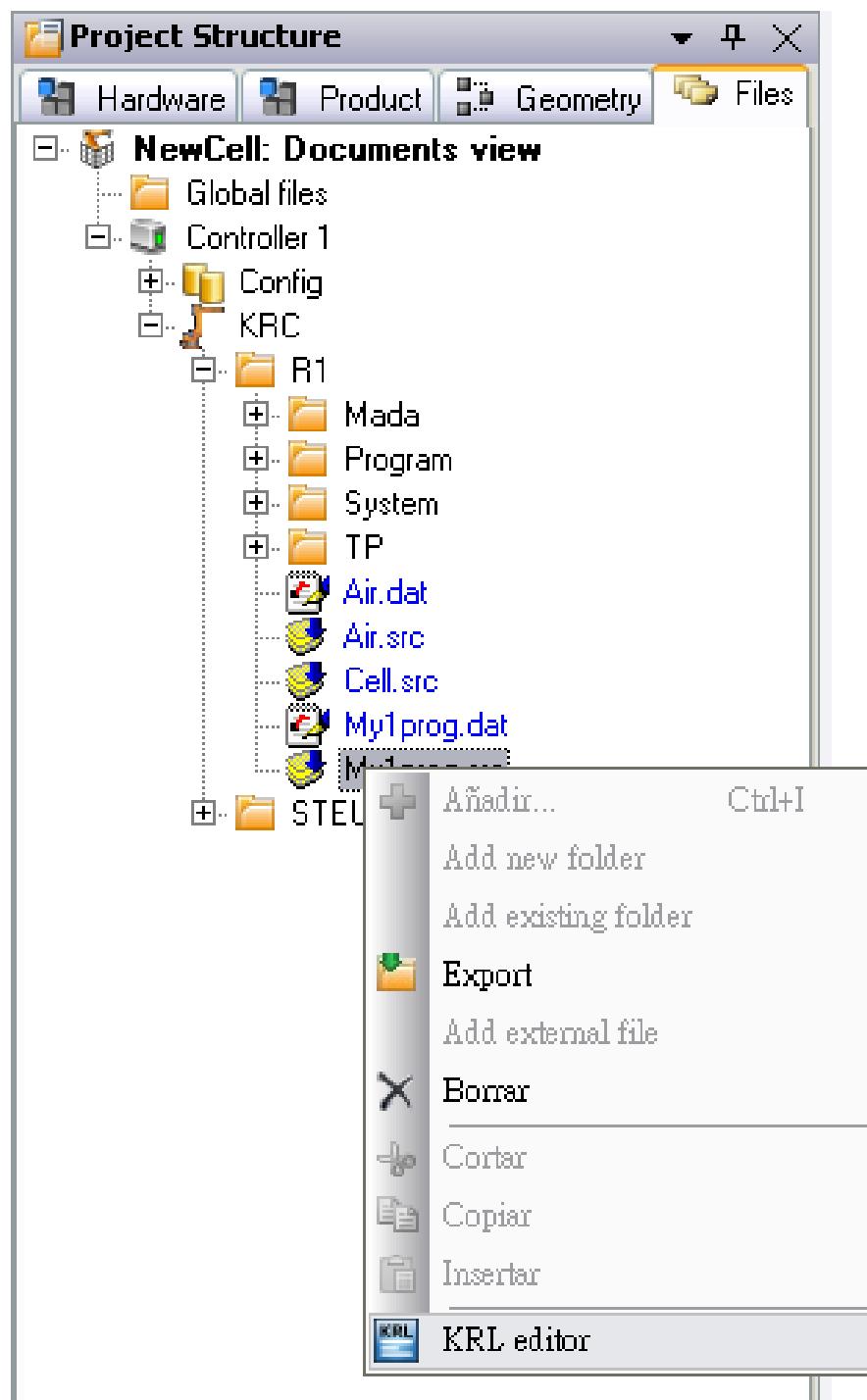


Fig. 9-22: Menú contextual de ratón de WorkVisual (editor KRL)

Procedimiento para añadir un fichero mediante plantillas KRL

1. Cambiar el árbol de proyecto del fichero
2. Desplegar los directorios hasta el directorio R1
3. Seleccionar la carpeta en la que se va a crear el fichero nuevo
4. Hacer clic con el botón derecho y añadir en el menú contextual

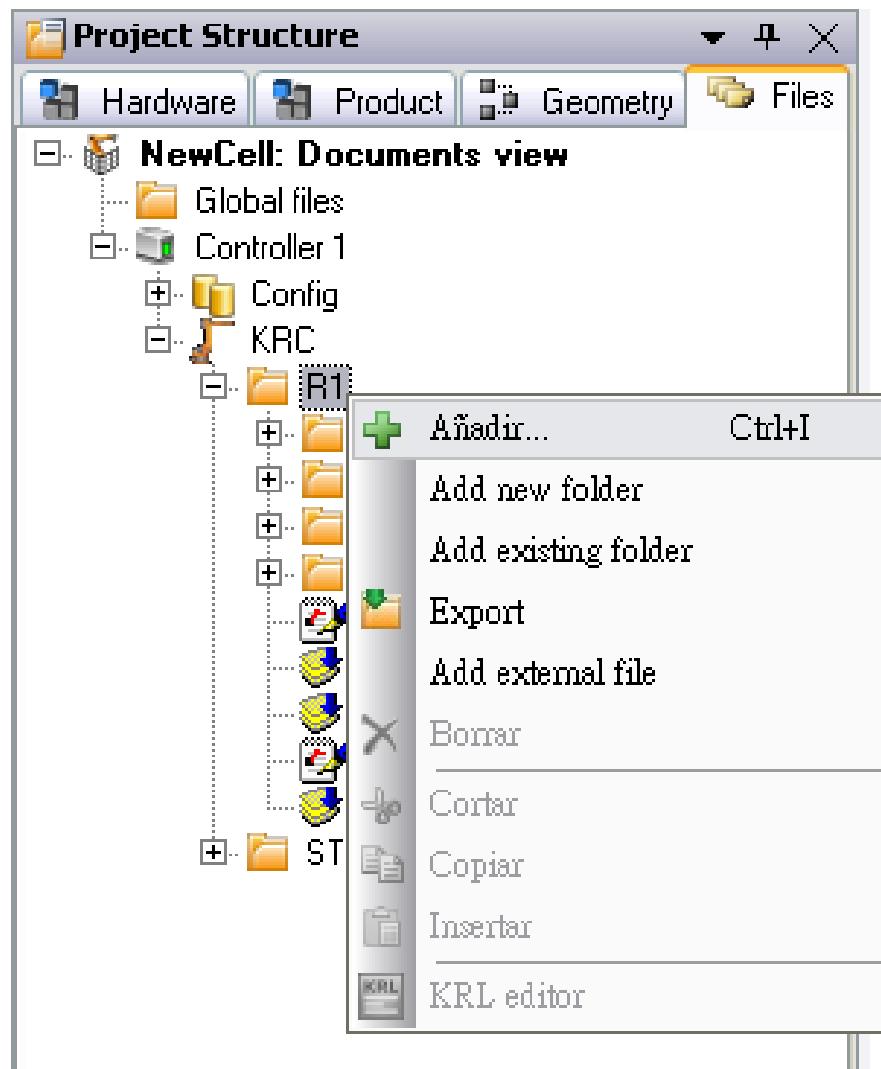


Fig. 9-23: Menú contextual de WorkVisual (añadir)

5. Seleccionar plantilla

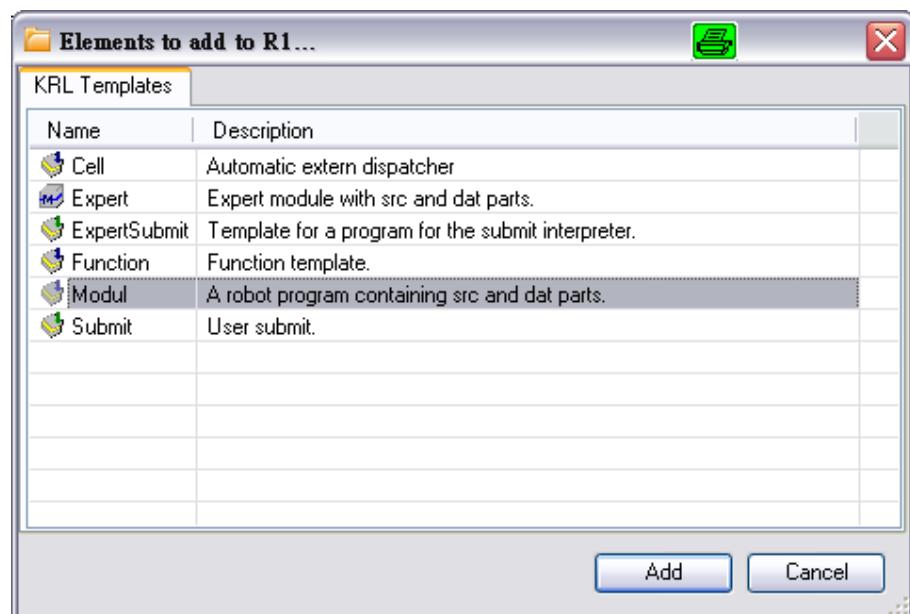
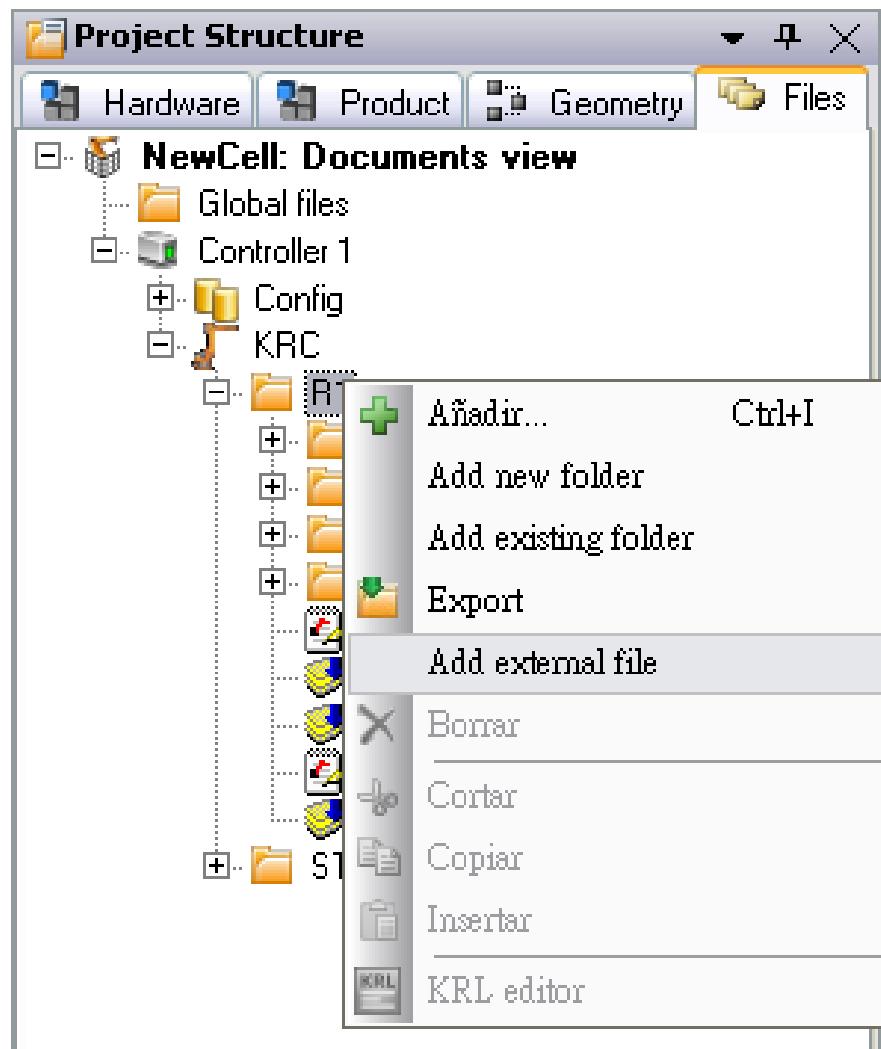


Fig. 9-24: Plantillas KRL de WorkVisual

6. Asignar el nombre del programa

Procedimiento para añadir ficheros externos

1. Cambiar el árbol de proyecto del fichero
2. Desplegar los directorios hasta el directorio R1
3. Seleccionar la carpeta en la que se va a crear el fichero nuevo
4. Hacer clic con el botón derecho y añadir Fichero externo en el menú contextual

**Fig. 9-25: Menú contextual de WorkVisual (añadir fichero externo)**

5. Seleccionar el fichero(s) y pulsar Abrir

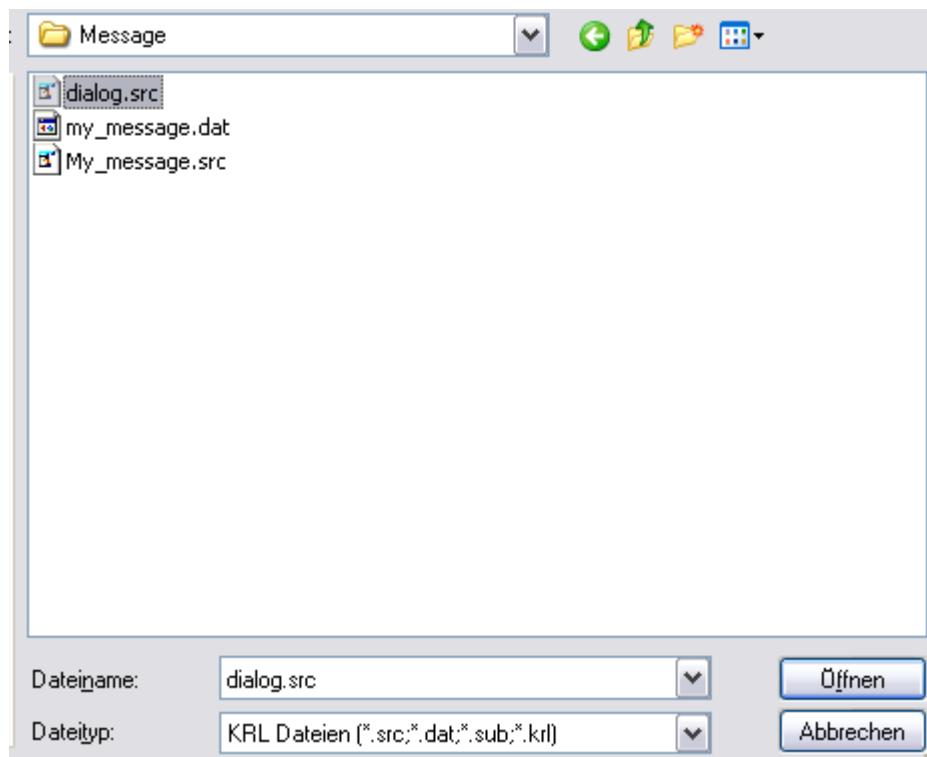


Fig. 9-26: Añadir ficheros externos en WorkVisual

9.2.2 Manejo del editor KRL

- | | |
|-----------------------------------|--|
| Descripción del editor KRL | Ejecución de programa (SRC/DAT) <ul style="list-style-type: none">■ A través de introducción de KRL directa■ Mediante entrada rápida para instrucciones KRL (fragmentos KRL (snippets))■ Mediante formularios en línea de la caja de herramientas |
| Configurar el editor KRL | Propiedades del editor KRL <ul style="list-style-type: none">■ Configuración del editor■ Descripción de colores en el editor KRL■ Detección de errores (anificador KRL)■ Lista de variables■ Función de procesamiento adicional en el editor KRL
<ul style="list-style-type: none">■ Seleccionar la secuencia de menú Extras > Opciones. Se abre la ventana Opciones.■ La carpeta Editor de texto contiene las opciones secundarias Appearance y Behavior (Visualización y Comportamiento).■ Appearance (Visualización) |

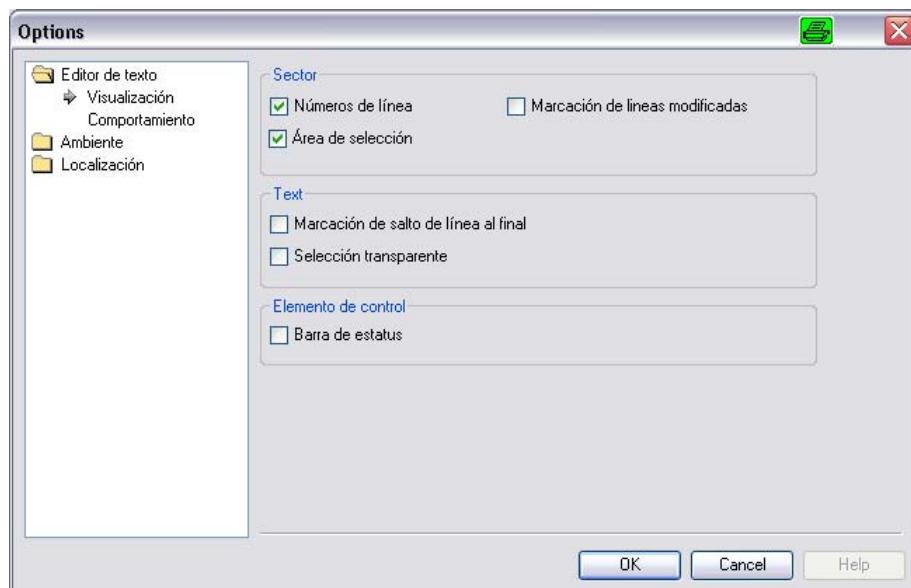


Fig. 9-27: Appearance (Visualización)

Appearance:

Campo	Descripción
Números de línea	Activo: muestra los números de línea.
Área de selección	Activo: el código marcado es resaltado adicionalmente por una barra roja vertical situada a la izquierda.
Marcación de líneas modificadas	Activo: marcar de amarillo las líneas modificadas al inicio de la línea.
Marcación de salto de línea al final	Sólo es relevante si en Behavior (Comportamiento) está activa la casilla Ajuste de línea . <ul style="list-style-type: none"> ■ Activo: los saltos de línea están identificados por una pequeña flecha verde. ■ Inactivo: no se marcan los saltos de línea.
Selección transparente	Visualización del código marcado: <ul style="list-style-type: none"> ■ Activo: letra en color original sobre fondo brillante ■ Inactivo: letra blanca sobre fondo oscuro
Barra de estado	Activo: en la parte inferior del editor KRL se muestra una barra de estado. Muestra, por ejemplo, el nombre del programa y el número de la línea en la que se encuentra el cursor en ese momento.

■ Behavior

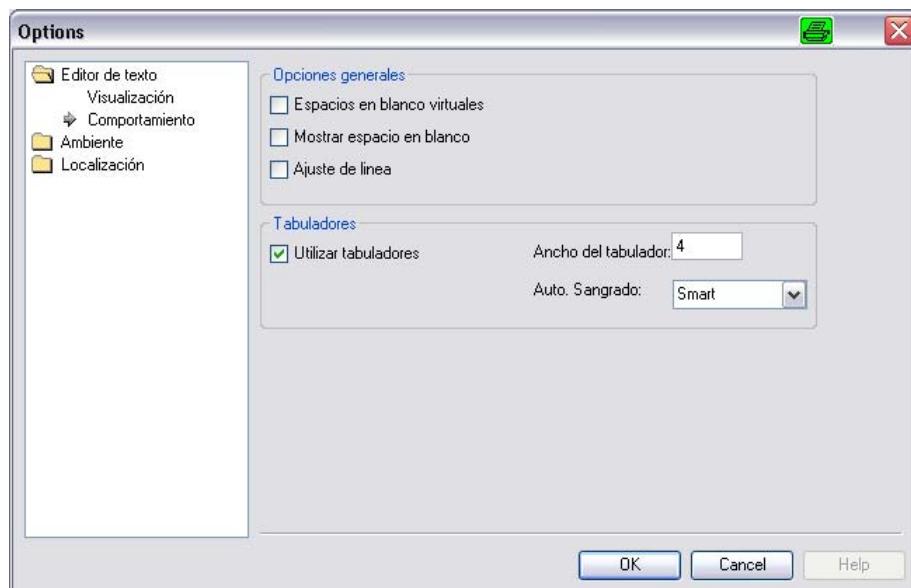


Fig. 9-28: Behavior

Behavior:

Campo	Descripción
Espacios en blanco virtuales	<ul style="list-style-type: none"> ■ Activo: el cursor se puede colocar en cualquier posición de líneas vacías. ■ Inactivo: el cursor se puede colocar sólo en el inicio de los espacios.
Mostrar espacios en blanco	Activo: se muestran los caracteres de control. (Espacios, tabulaciones, etc.)
Ajuste de línea	<ul style="list-style-type: none"> ■ Activo: cuando las líneas llegan al ancho de pantalla se produce un salto de línea. ■ Inactivo: no se produce un salto de línea. Si hay líneas más anchas que la ventana se muestra automáticamente una barra de desplazamiento.
Utilizar tabuladores	<ul style="list-style-type: none"> ■ Activo: la tecla del tabulador introduce una tabulación. ■ Inactivo: la tecla del tabulador introduce tantos espacios como se haya definido en Ancho del tabulador.
Ancho del tabulador	Una tabulación debe ser tan ancha como x espacios.
Auto. sangrado	<p>Comportamiento cuando se crea una nueva línea con la tecla Enter:</p> <ul style="list-style-type: none"> ■ None: la nueva línea no está sangrada. ■ Block: la nueva línea tiene el mismo sangrado que la línea anterior. ■ Smart: comportamiento dentro de folds Si la línea anterior está sangrada se aplicará este sangrado en la nueva línea. Si la línea anterior no está sangrada se sangrará la nueva línea.

Descripción de colores del editor KRL

■ Descripción de los colores

El editor KRL reconoce las partes del código introducido y las representa automáticamente en diferentes colores.

Parte del código	Color
Palabras clave KRL (a excepción de ;FOLD y ;ENDFOLD)	azul medio
; FOLD y ;ENDFOLD	gris
Números	azul oscuro
Cadenas (texto entre comillas "...")	rojo
Comentarios	verde
Caracteres especiales	verde azulado
Otro código	negro

- Ejemplo para la utilización de los colores

```

15
16 ;FOLD: PTP HOME Vel= 100 % DEFAULT;{PE}%MKUKATPBASIS
17 $BWDSTART = FALSE
18 PDAT_ACT=PDEFAULT
19 FDAT_ACT=FHOME
20 BAS (#PTP_PARAMS,100 )
21 $H_POS=XHOME
22 PTP XHOME
23 ;ENDFOLD
24

```

Fig. 9-29: Ejemplo de colores en el editor KRL

1	Palabras clave KRL: azul
2	Comentario: verde
3	FOLD: gris
4	Otro código: negro

Detección de errores del editor KRL

- El editor KRL dispone de una detección automática de errores
- Los errores detectados en el código de programación se subrayan en rojo
- Los errores sólo son visibles en la ventana de mensajes cuando la categoría **Analizador KRL** está seleccionada
- Se detectan errores KRL y parcialmente errores estructurales (declaración en el lugar incorrecto en el programa)
- Los errores de escritura en las variables no se detectan



La detección de errores no detecta todos los errores. Si no hay nada subrayado no quiere decir que el programa esté libre de errores.

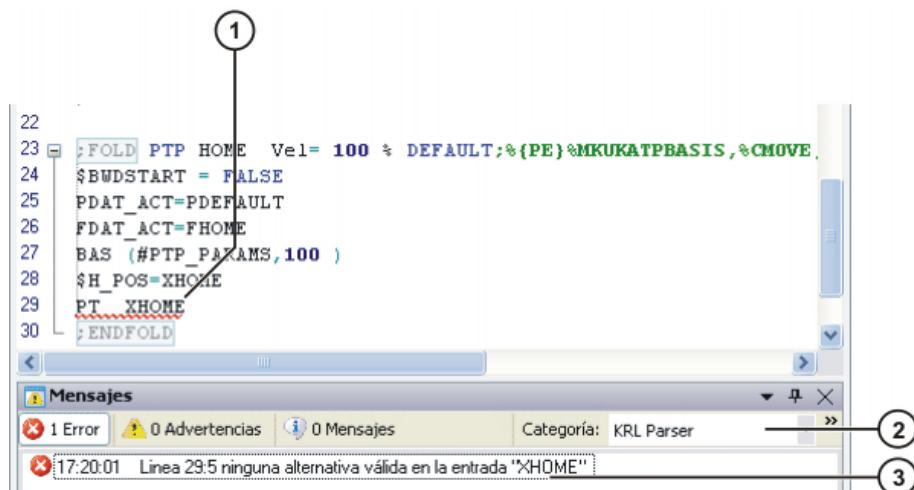


Fig. 9-30: Ejemplo de detección de errores

1	Editor KRL: El error está subrayado en rojo.
2	Ventana de mensajes: La categoría Analizador KRL está seleccionada.
3	Ventana de mensajes: Descripción de errores con número de líneas y columnas

Función de la lista de variables

- Todas las variables KRL declaradas en un fichero determinado pueden visualizarse de forma clara en una lista

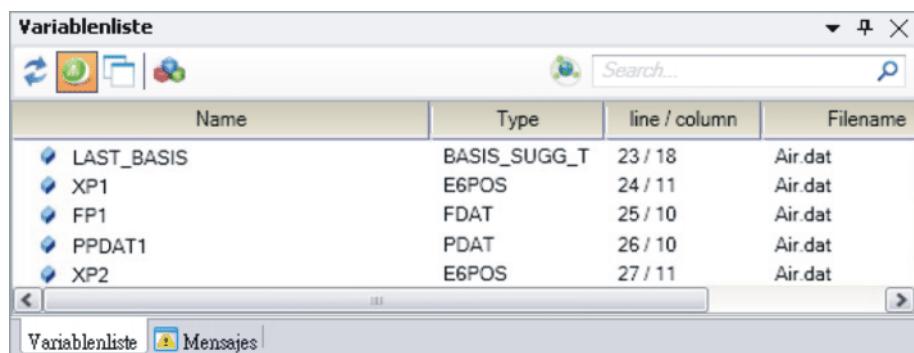


Fig. 9-31: Ejemplo de lista de variables

- En los ficheros SRC también se visualizan siempre las variables de los ficheros DAT correspondientes y viceversa

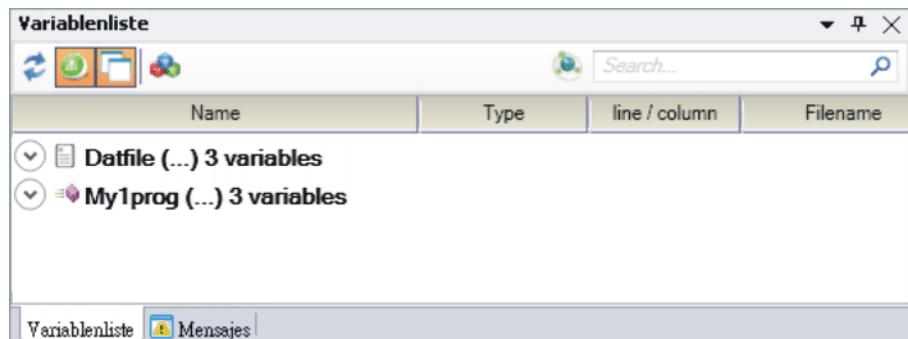


Fig. 9-32: Ejemplo de lista de variables (variables de SRC y DAT)

- Introducir el nombre de variables o una parte del nombre en el campo de búsqueda. Se muestra inmediatamente el resultado de la búsqueda.

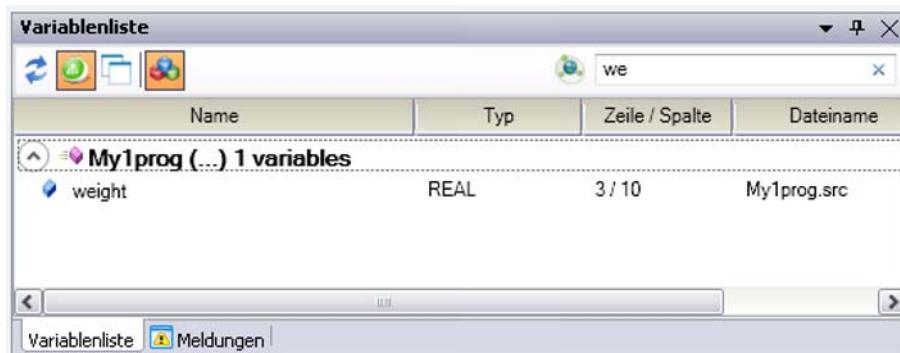


Fig. 9-33: Ejemplo de lista de variables (función de búsqueda)



Si se ha introducido algo en el campo de búsqueda, esto aún tendrá validez aunque se enfoque otro fichero. Solamente es posible mostrar todas las variables de un fichero si el campo de búsqueda está vacío.

- Opciones de ajuste

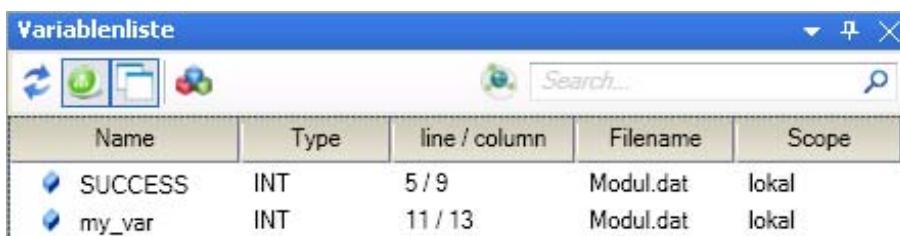


Fig. 9-34: Ventana lista de variables

Haciendo clic sobre una columna es posible ordenar la lista según esa columna.

Botón	Descripción
	Leer de nuevo
	Actualiza la lista siempre que se modifica el fichero seleccionado en el árbol
	Actualiza la lista cuando se modifica el editor actual
	Agrupa las variables según las funciones locales del nivel secundario (SRC/DAT)
	El botón está pulsado: La búsqueda incluye todas las variables globales.

Funciones de procesamiento adicionales

Se puede acceder a las funciones de procesamiento convencionales a través de **Editar** en el menú contextual. Son:

- **Cortar, Insertar, Copiar, Borrar**
- **Anular, Restaurar**
- **Buscar ..., Reemplazar ...**
- **Ir a...**
- **Seleccionar todo**

En **Ampliado** del menú contextual se dispone de más funciones de procesamiento:

Ampliado > ...	Descripción
Utilizar pestañas	Sustituye los espacios por tabuladores en el sector marcado. Condición: el ajuste de configuración Utilizar tabuladores está activo.
Borrar pestañas	Sustituye las tabulaciones por espacios en el sector marcado.
Aumentar sangrado	Añade espacios o una tabulación en la posición del cursor.
Reducir sangrado	Borra la tabulación o los espacios a la izquierda del cursor.
Insertar punto y coma	Coloca al inicio de la línea marcada un punto y coma.
Insertar almohadilla	Borra el punto y coma del inicio de la línea marcada.
Cerrar todos los folds	Cierra todos los folds del archivo que se muestra actualmente.
Abrir todos los folds	Abre todos los folds del archivo que se muestra actualmente.

Instrucciones de actuación

Entrada rápida para instrucciones KRL (fragmentos KRL (snippets))

Se debe programar una declaración de interrupción. Para no tener que introducir la sintaxis completa INTERRUPT DECL ... WHEN ... DO, se utilizan fragmentos KRL (snippets). Solo será necesario rellenar manualmente las posiciones variables de la sintaxis.

Procedimiento para la programación con fragmentos de código (snippets)

1. Colocar el cursor en el lugar deseado

- Hacer clic con el botón derecho y seleccionar la entrada **Insertar Co-desnippet** en el menú contextual. Se muestra un campo de listas. Hacer doble clic en la instrucción deseada.

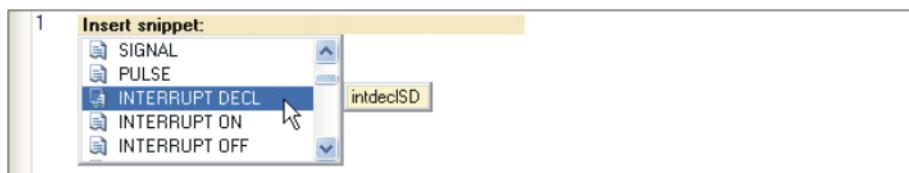


Fig. 9-35: Hacer doble clic en la instrucción

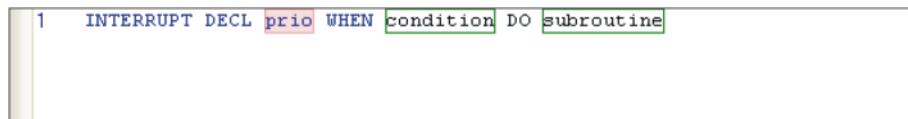
- O bien: teclear la abreviatura y pulsar la tecla TAB.



Fig. 9-36: Fragmentos KRL (snippets) con función de búsqueda

(Se pueden determinar las abreviaturas accediendo al campo de listas con el Codesnippets. Al pasar con el puntero del ratón sobre la instrucción se visualizará la abreviatura al lado de la ventana. Las últimas dos letras son siempre SD y no deben escribirse.)

2. La sintaxis KRL se añadirá automáticamente. La primera posición variable está marcada de color rojo. Introducir el valor deseado.



```
1 INTERRUPT DECL prio WHEN condition DO subroutine
```

Fig. 9-37: La primera posición variable está marcada de color rojo

3. Con la tecla Enter saltar hasta la siguiente posición variable. Introducir el valor deseado.
4. Repetir el paso 3 para todas las posiciones variables.

Editor KRL: Programación con la caja de herramientas

Activar la caja de herramientas

1. Menú Ventana>Caja de herramientas
2. Posicionar o acoplar la caja de herramientas
3. Cargar el programa en el editor KRL y la caja de herramientas se llena con las funciones

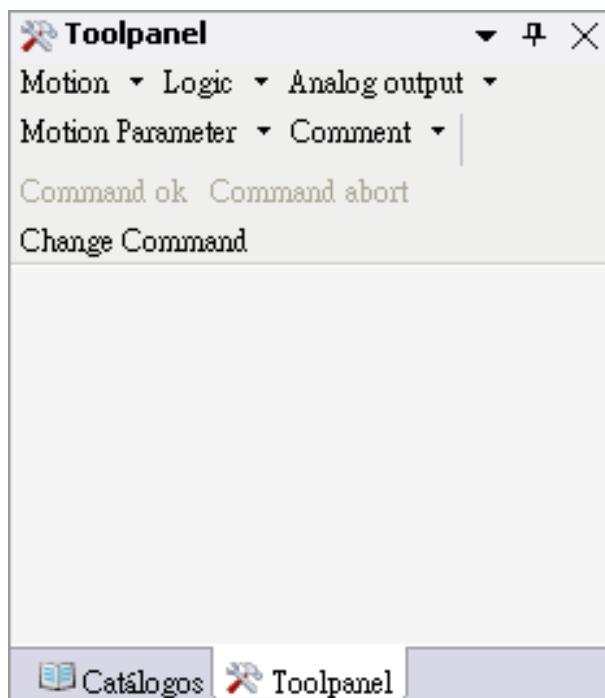


Fig. 9-38: Caja de herramientas llena

Procedimiento para la programación con la caja de herramientas

1. Colocar el cursor en el lugar deseado.
2. Seleccionar el formulario en línea deseado en la caja de herramientas

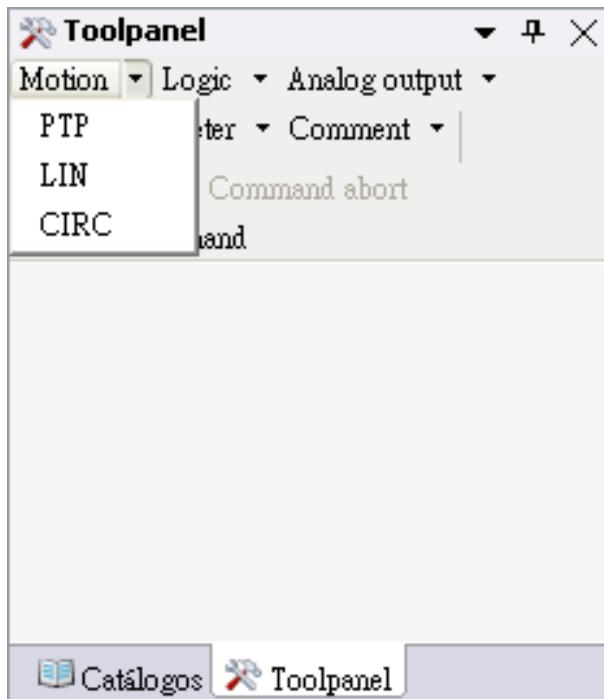


Fig. 9-39: Movimiento caja de herramientas

3. Editar formulario en línea / ajustar parámetros

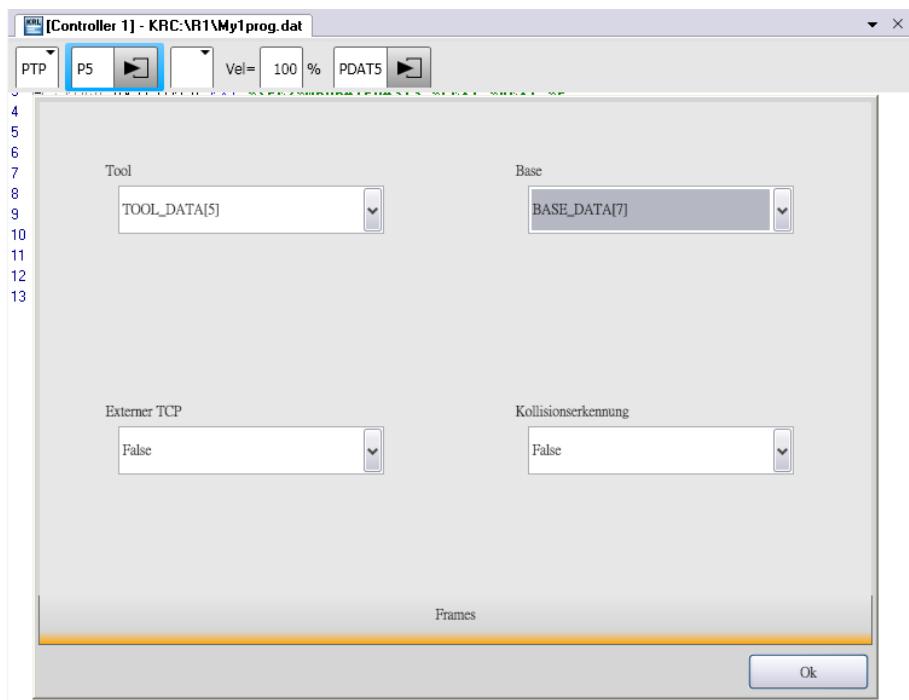


Fig. 9-40: Formulario en línea del editor KRL

4. Finalizar el formulario en línea con la tecla **Instrucción OK** en la caja de herramientas

Indice

Símbolos

\$TIMER
° 83

A

Abrir proyecto 115
Activar proyecto 127
Administrador 15
Ayuda 117

Á

Ángulo circular 62, 73

B

Barra de menús 116
Barras de botones 116
Bloquear 129
Bucle de conteo 93
Bucle sinfín 91
Bucle, finito 95
Bucle, infinito 96
Bucles 91

C

Campos 32
Campos de trabajo (ventana) 117
Cargar proyecto 118
CASE 89
Catálogos (ventana) 117
Comentario 5
Comparar proyectos 119
Consultas 87
Control de ejecución de programa 87

D

DECL 24
Declaraciones 21
Declaración 24, 25
DEFFCT 55
DISTANCE 106
Distribuidor 88, 89

E

Editor KRL 136
Ejemplo PEP 11
ENUM 42
Estructura 38
Estructura del proyecto (ventana) 117
Explorador de proyecto 118

F

Fold 7
Funciones 45, 55, 57
Funciones de conmutación 103
Funciones estándar 29, 57
Funciones para la emisión de mensaje 57
Funciones para variables de cadena 57
Funciones, matemáticas 57

función de conmutación referida a la trayectoria 106, 110
Función de espera, dependiente de una señal 99
Función de espera, dependiente del tiempo 98

G

Gestión de datos 21
global 21
Grupo 32
Grupo de usuario, por defecto 15

I

IF ... THEN 87
importar, proyecto de WorkVisual 130
Impulso 104
Inicialización 27
Instalar 123
Interfaz de usuario de WorkVisual 116
Introducción al nivel del experto 15

L

local 21

M

Manipulación 28, 29
Mensajes (ventana) 117
Metodología de programación, ejemplo PEP 11

N

Nombres de datos 9

O

Operaciones de bit 29
Operaciones de comparación 29
Operaciones lógicas 29
Operario 15

P

Palabra clave 24
PATH 110
PEP 10
Plan de ejecución del programa, PEP 10
Prioridad 31, 107, 111
Programación de movimientos en KRL 59
Programación estructurada 5
Programador 15
Propiedades (ventana) 117
Puntero de ejecución en avance 24

R

Ramificación 87
Return 46, 55

S

Subprogramas 8, 45
Subprogramas, globales 47
Subprogramas, locales 45
SWITCH 89

Símbolos PEP 10

T

Template 130
Temporizador 83
Tiempo ciclo 83
Tipo de datos de enumeración 42
Tipos de cálculo básicos 28
Transferencia de parámetros 49
Transmitir 123
TRIGGER 106, 110

V

Variables 21, 24
Variables de sistema 83
Vida útil 21

W

WHILE 95
WorkVisual 115

