

LIBRO DE

# PROGRAMACIÓN EXPERTO DE ROBOTS

Para Software KUKA V5.x

PROGRAMADOR



# **KUKA Sistemas de Automatización, S.A.**

**Pol. Ind. Torrent de la Pastera**

**08800 Vilanova i la Geltrú**

**Tel +34 93 814 23 53**

**Fax +34 93 814 29 50**

---

**© Copyright 2006**

**KUKA Roboter GmbH  
Zugspitzstraße 140  
D-86165 Augsburg**

**KUKA Sistemas de automatización S.A.  
Pol. Ind. Torrent de la Pastera  
08800 Vilanova i la Geltrú**

This documentation or excerpts therefrom may not be reproduced or disclosed to third parties without the express permission of the publishers.

Other functions not described in this documentation may be operable in the controller. The user has no claims to these functions, however, in the case of a replacement or service work.

We have checked the content of this documentation for conformity with the hardware and software described.

Nevertheless, discrepancies cannot be precluded, for which reason we are not able to guarantee total conformity. The information in this documentation is checked on a regular basis, however, and necessary corrections will be incorporated in subsequent editions.

Subject to technical alterations without an effect on the function.

KUKA Roboter GmbH accepts no liability whatsoever for any errors in technical information communicated orally or in writing in the training courses or contained in the documentation. Nor will liability be accepted for any resultant damage or consequential damage.

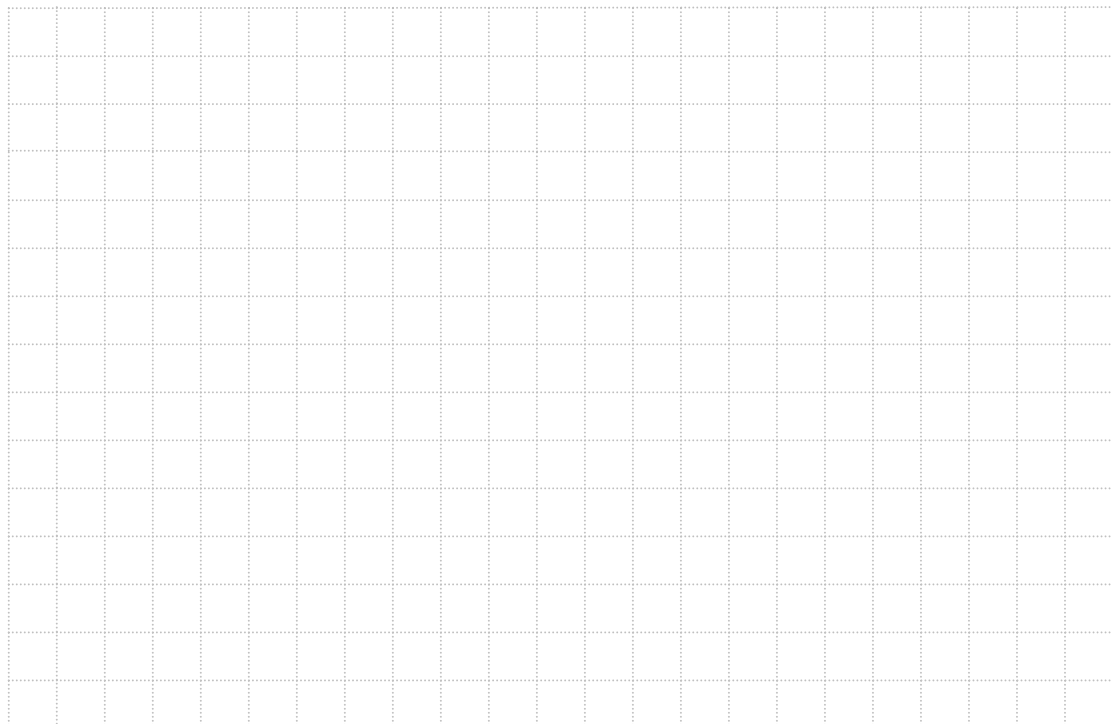
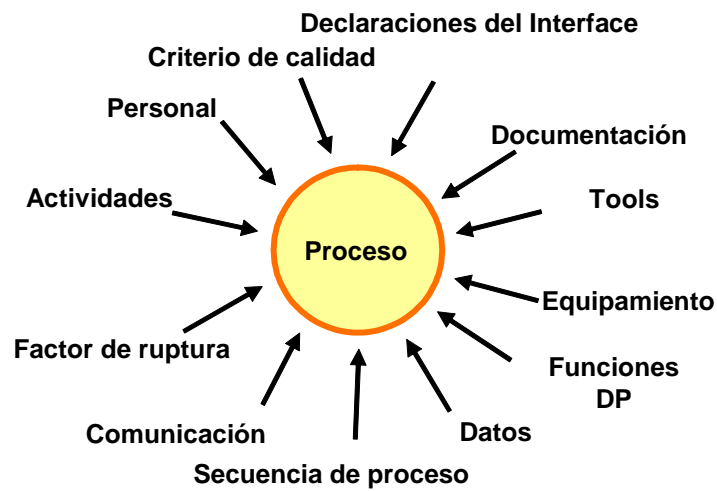
Responsible for this training documentation: College Development (WSC-IC)

## **ÍNDICE**

<b>1. EJECUCIÓN DE UN PROYECTO .....</b>	<b>4</b>
<b>2. MEDIDAS ANTICOLISIÓN PARA ROBOT (DISPOSITIVO DE SEGURIDAD).....</b>	<b>18</b>
2.1 Dispositivo de seguridad del robot.....	18
2.2 Monitorización de los Workspace .....	41
<b>3. PROGRAMACIÓN DE MENSAJES .....</b>	<b>57</b>
3.1 KRL-Sintaxis para programación de mensajes.....	57
3.2 Mensajes de aviso .....	65
3.3 Mensajes de confirmación .....	67
3.4 Mensajes de estado .....	69
3.5 Mensajes de dialogo.....	72
3.6 Mensajes con variables.....	75
<b>4. OPERADOR GEOMÉTRICO .....</b>	<b>85</b>
<b>5. PROGRAMACIÓN DE INTERRUPCIONES .....</b>	<b>89</b>
5.1 Teoría y KRL-sintaxis para programación de interrupciones.....	89
5.2 Cancelar movimiento con interrupciones .....	104
<b>6. FUNCIONES DE CONMUTACIÓN RELATIVAS A LA TRAYECTORIA (TRIGGER).....</b>	<b>109</b>
<b>7. TRATAMIENTO AUTOMÁTICO DE DEFECTOS .....</b>	<b>121</b>
7.1 Estrategia ante defectos en general .....	121
7.2 Estrategia del retorno automático.....	129
<b>8. TRABAJANDO CON ENTRADAS / SALIDAS.....</b>	<b>140</b>
8.1 Entradas / salidas digitales.....	140
8.2 Entradas predefinidas .....	152
8.3 Entradas / salidas analógicas.....	157

## **1. Ejecución de un proyecto**

## Influencias en una aplicación



## Metodología de programación – Diagrama de trabajo



Todas las etapas para crear un aplicación.

## Metodología de programación – Diagrama de trabajo



Investigar cliente: Es el primer contacto entre el cliente y el programador. El puede ser un departamento de la empresa.

## Metodología de programación – Diagrama de trabajo



El informe se utiliza para recopilar la información de fondo, los objetivos, el contenido, los horario y otra información preliminar.



## Metodología de programación – Diagrama de trabajo



La definición de metas se elabora en base del informe y sirve para definir metas y soluciones fijas.

Las metas se pueden modificar como resultado del análisis de situación. Esto da lugar a un lazo de regeneración.

## Metodología de programación – Diagrama de trabajo



El análisis de situación describe la situación actual e influencia en la aplicación para soluciones y requerimientos. Un lazo de regeneración se puede establecer entre la definición de metas y el análisis de situación.

## Metodología de programación – Diagrama de trabajo



La especificación de requisitos documenta detalladamente los requisitos, la puesta en marcha y la conclusión del proyecto. Los apartados sin resolver también se registran en la especificación de requisitos.

## Metodología de programación – Diagrama de trabajo



El comienzo formal de la programación, de la configuración y de la consecución ("kick-off").

## Metodología de programación – Diagrama de trabajo



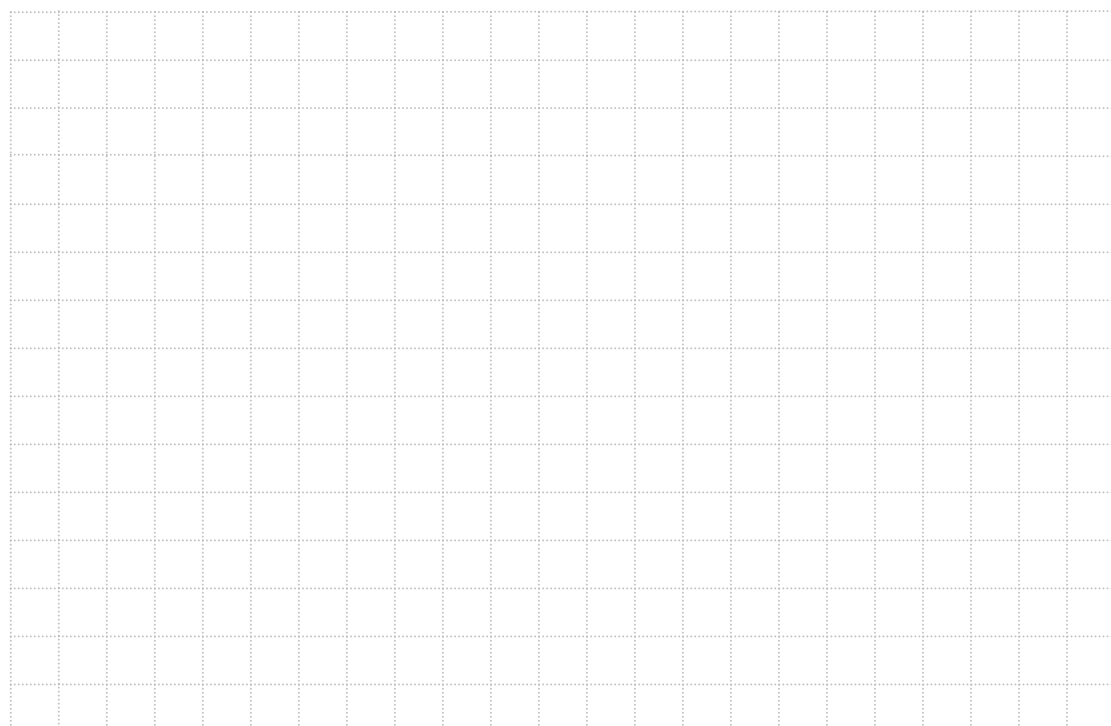
Las partes y los accesorios que se procurarán (grippers, tarjetas de interfaz, etc.) se han definido exactamente en el análisis de situación y la definición de metas.

## Metodología de programación – Diagrama de trabajo



Todas las interfaces se deben definir::

- Robot – Máquina
- Robot – Persona
- Robot – PLC



## Metodología de programación – Diagrama de trabajo

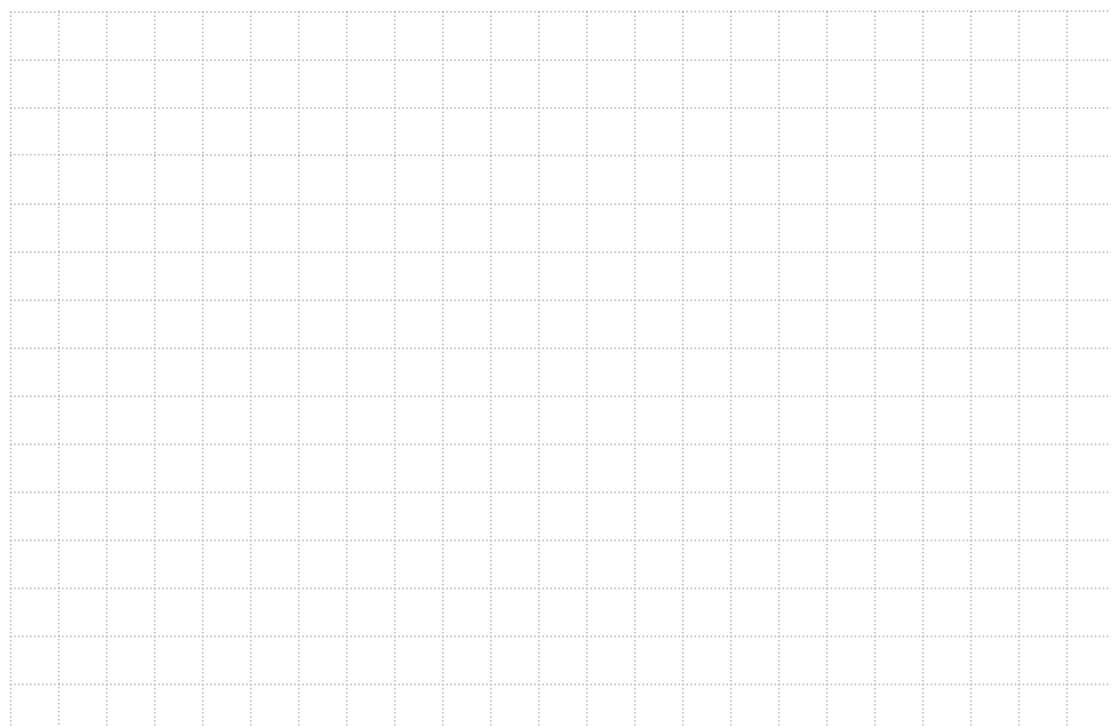


La programación es el actual foco de la aplicación. Esta es la fase en la cual se realiza la programación, soluciones, etc..,

## Metodología de programación – Diagrama de trabajo



Una vez programado y chequeado es momento de realizar la documentación la programación y la documentación de uso.





## Metodología de programación – Diagrama de trabajo



La completación del proyecto de programación se finaliza con la aceptación de la instalación por parte del cliente, en este apartado se incluye la documentación.

## **2. Medidas anticolisión para robot (dispositivo de seguridad)**

### **2.1 Dispositivo de seguridad del robot**

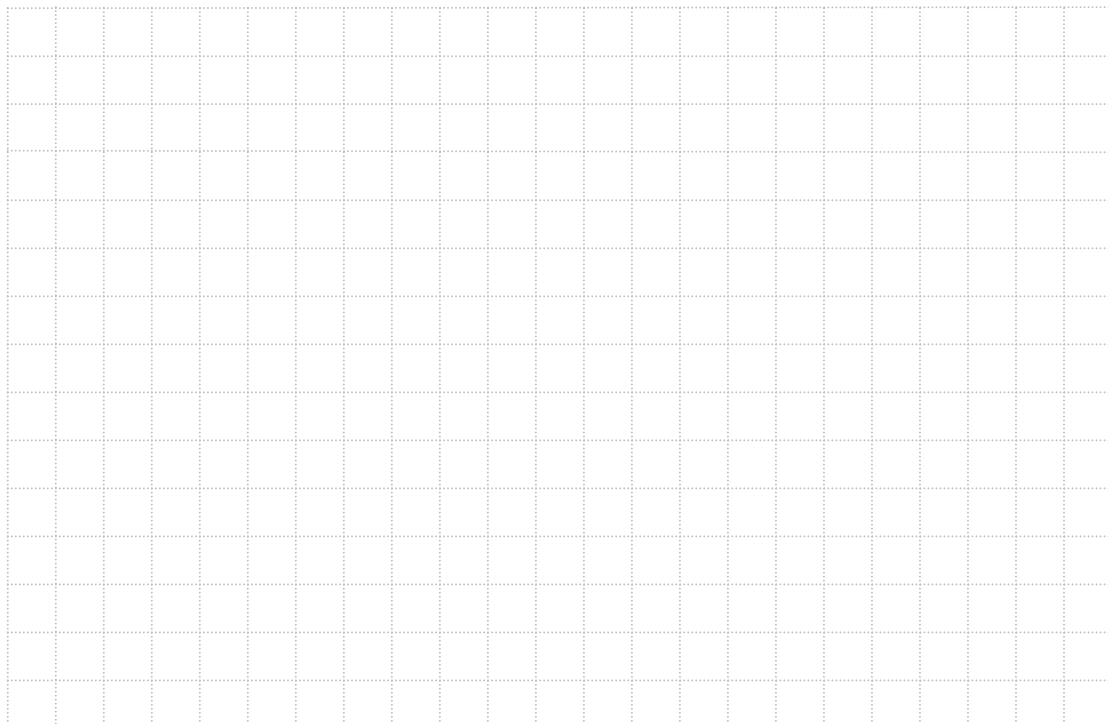
## Anti-colisión con más de un robot

Opciones de bloqueo entre robots que comparten un área de trabajo:

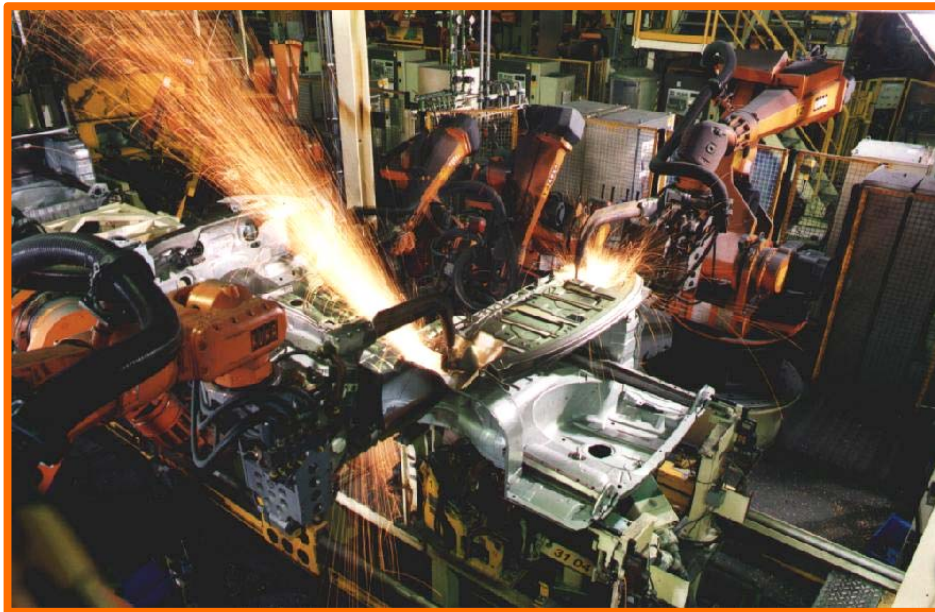
***1.) Comunicación directa entre los controles***

***2.) Comunicación indirecta entre los controles***

***3.) Control de prioridades de anticolidión via PLC***



## Soldadura por resistencia en la industria de automoción

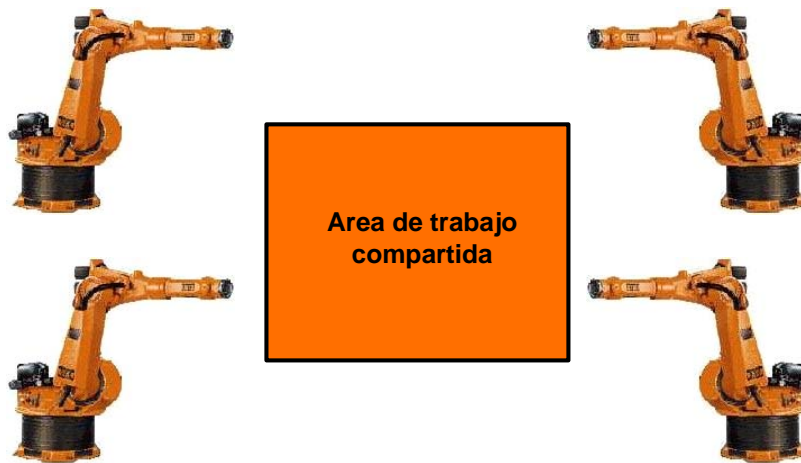


01/2006 2  
robot\_interlock\_en.ppt  
© Copyright by KUKA Roboter GmbH College



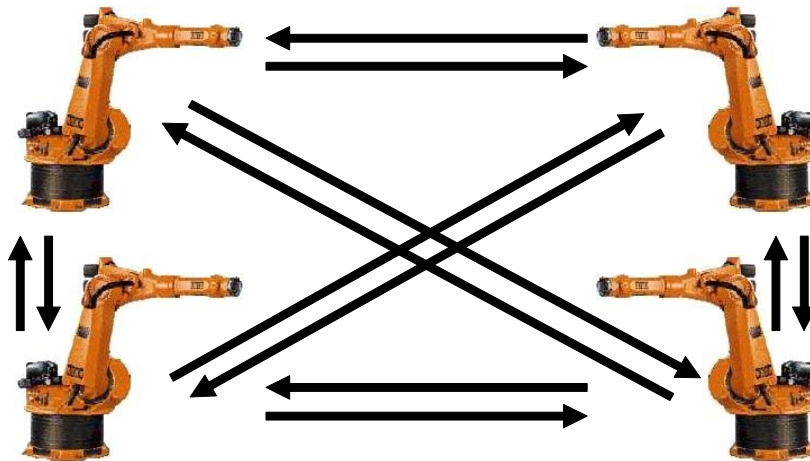
## Comunicación directa entre los controles

La comunicación se lleva a cabo via un enlace directo de I/O entre los controles

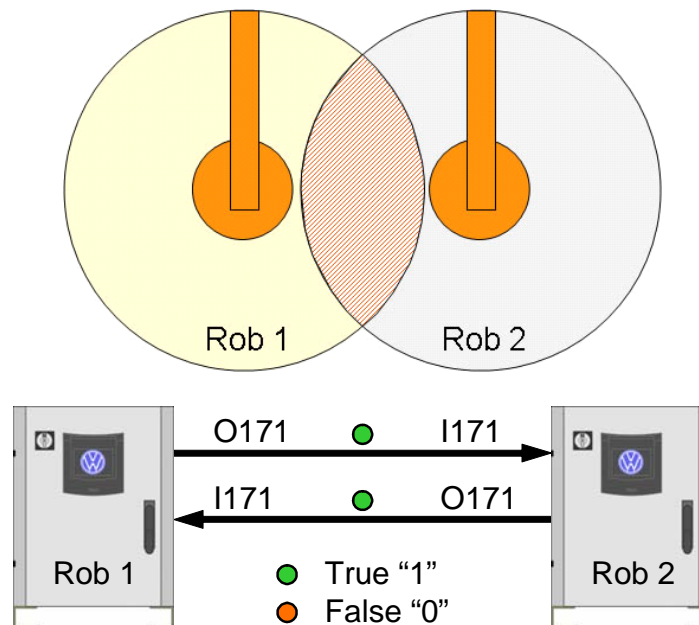


## Comunicación directa entre los controles

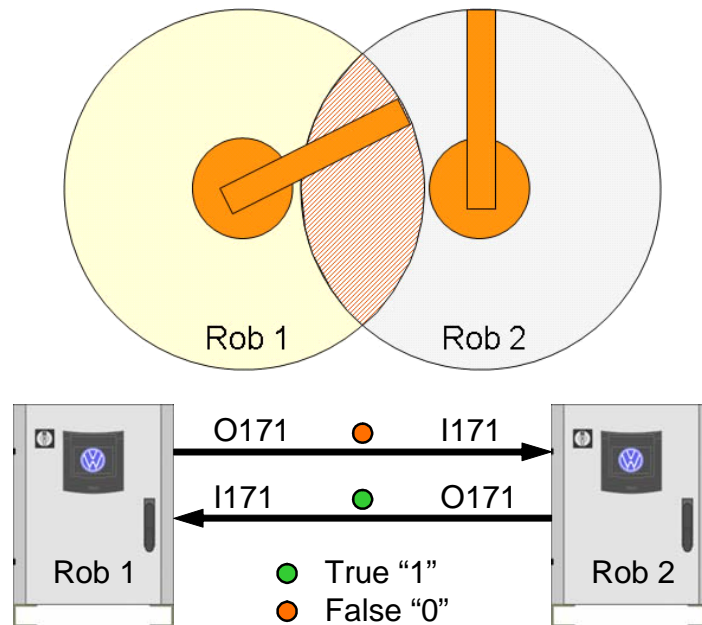
La comunicación se lleva a cabo via un enlace directo de I/O entre los controles



Ejemplo: Bloqueo sin PLC (1)

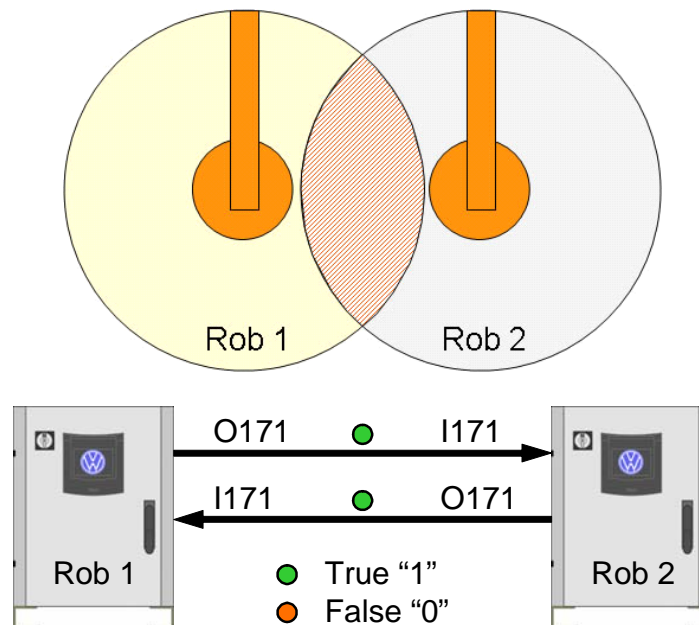


Ejemplo: Bloqueo sin PLC (2)

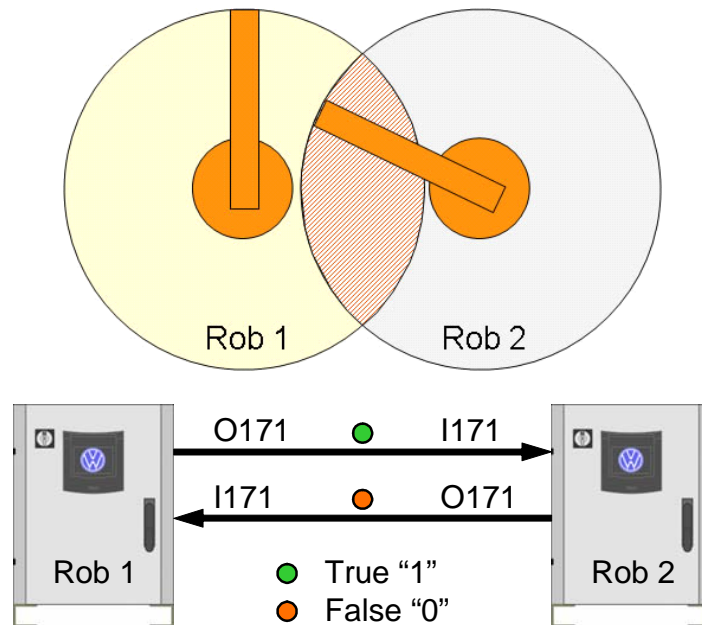




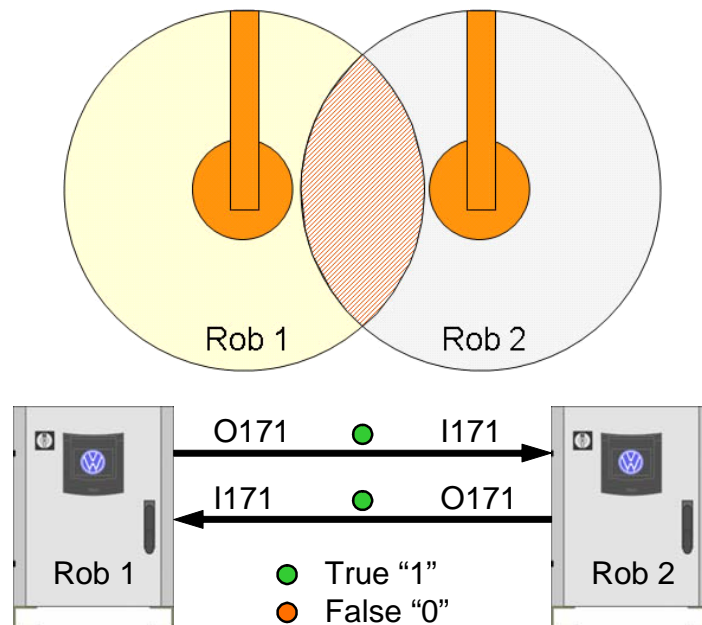
Ejemplo: Bloqueo sin PLC (3)



Ejemplo: Bloqueo sin PLC (4)



Ejemplo: Bloqueo sin PLC (5)



## Comunicación directa entre controles

La comunicación se lleva a cabo via un enlace directo de I/O entre los controles, e.g. CAN bus

### Con tiempo de monitorización:

El tiempo de monitorización se introduce en el programa de robot. Este empieza a contar cuando se le ha autorizado a ocupar el área de trabajo. El robot espera hasta que pasa el tiempo y entonces el robot chequea de nuevo el permiso para entrar. Si todavía tiene el permiso, entonces el robot entra en el área de trabajo compartida, sino se queda parado. Dependiendo de la programación el robot puede volver a pedir la autorización.



## Comunicación directa entre controles

La comunicación se lleva a cabo via un enlace directo de I/O entre los controles, eg CAN bus.

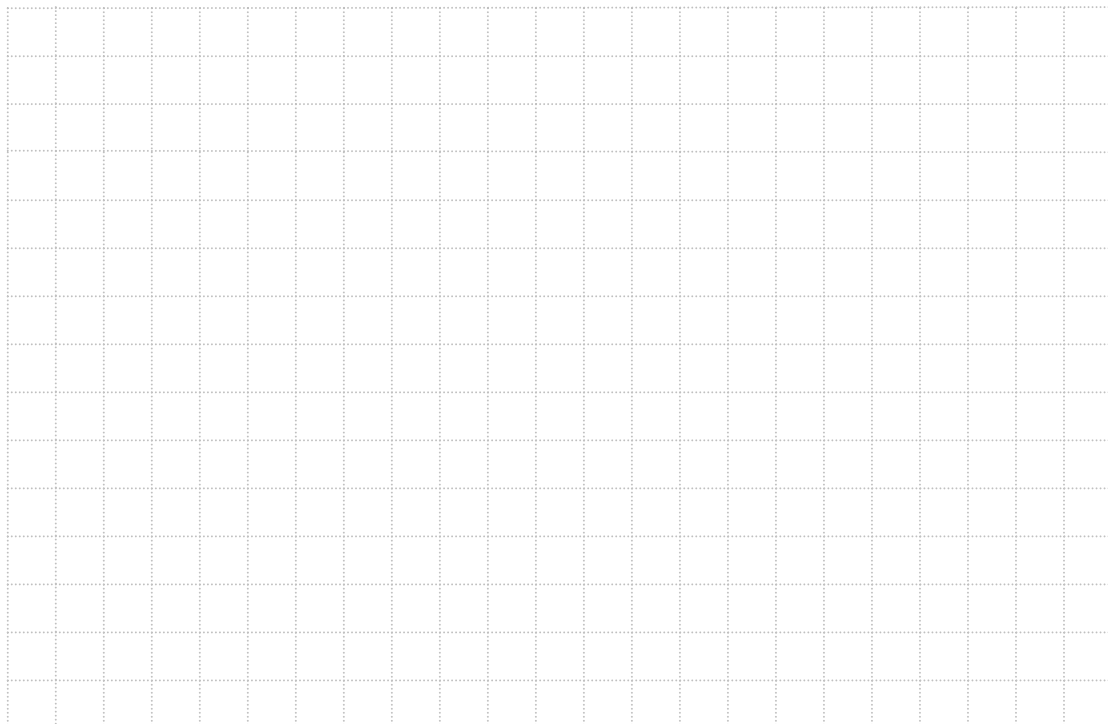
### Con tiempo de monitorización:

#### *Ventajas:*

- *Corto tiempo de comunicación (mayor que sin tiempo de monitorización)*
- *No se precisa programador de PLC*

#### *Desventajas:*

- *Si dos robots piden permiso a la vez para ocupar la zona de trabajo se produce el bloqueo de los dos*
- *Se necesita comunicación I/O entre PLC y robot y entre todos los robots que comparten un área de trabajo*



## Comunicación directa entre los controles

La comunicación se lleva a cabo via un enlace directo de I/O entre los controles, e.g. CAN bus

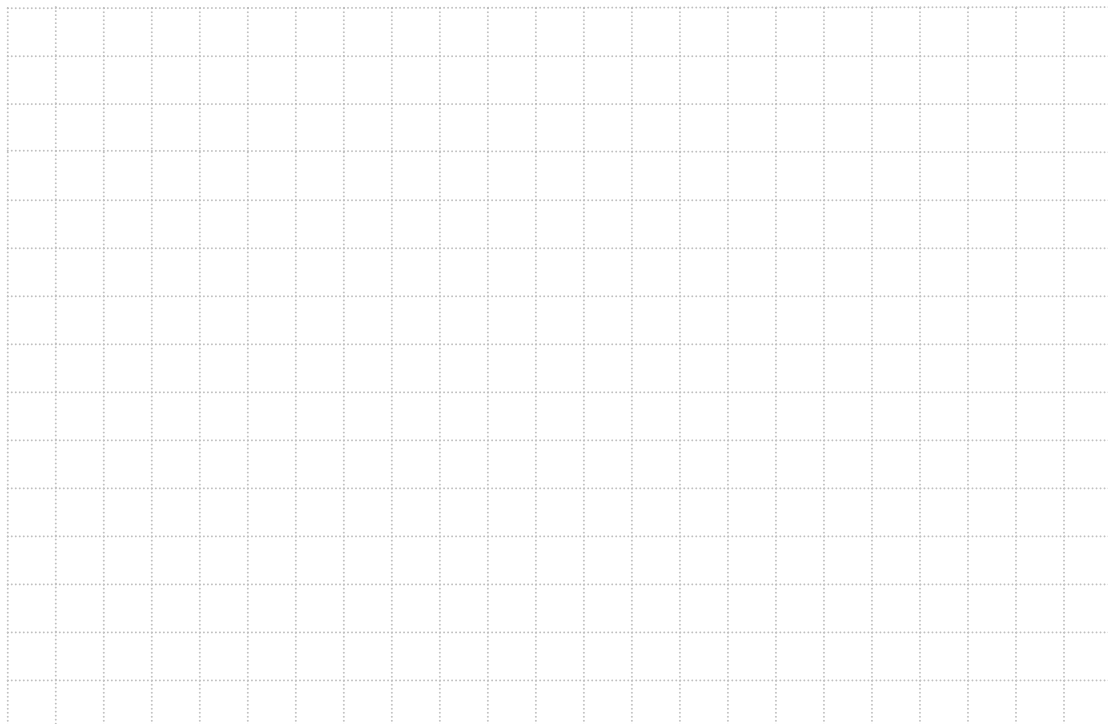
### Sin tiempo de monitorización:

#### *Ventajas:*

- *Corto tiempo de comunicación*
- *No se precisa programador de PLC*

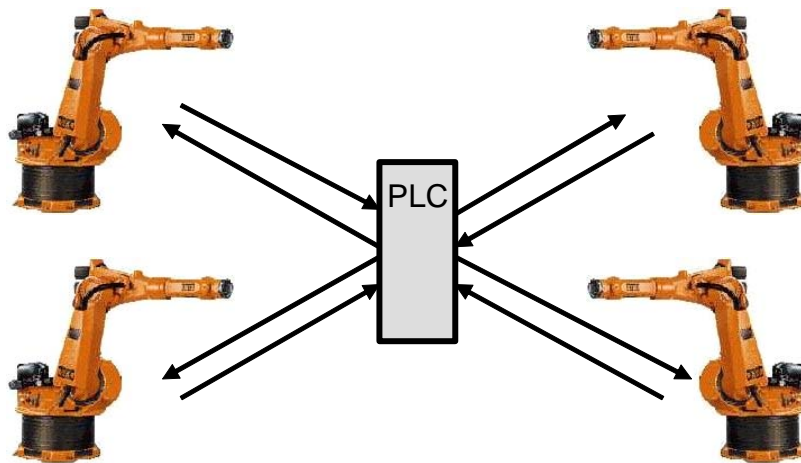
#### *Desventajas:*

- *Si dos robots piden permiso a la vez el resultado es una **COLISIÓN***
- *Se necesita comunicación I/O entre PLC y robot y entre todos los robots que comparten un área de trabajo*

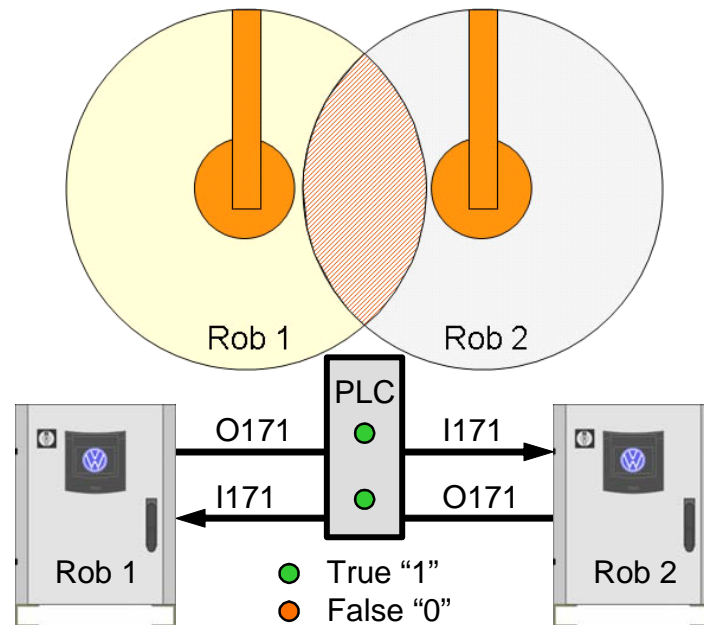


## Comunicación indirecta entre controles

La comunicación es dirigida y controlada por un PLC



Ejemplo: Bloqueo con PLC (1)

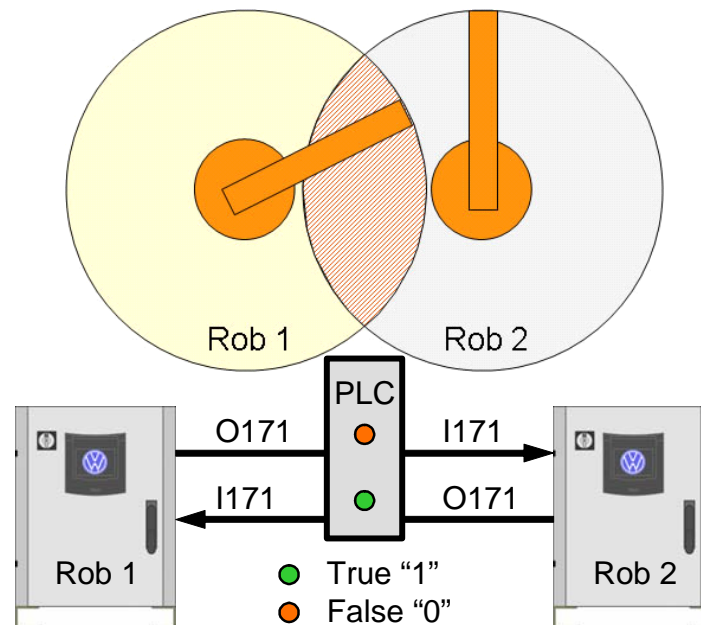


01/2006 14  
robot\_interlock\_en.ppt  
© Copyright by KUKA Roboter GmbH College

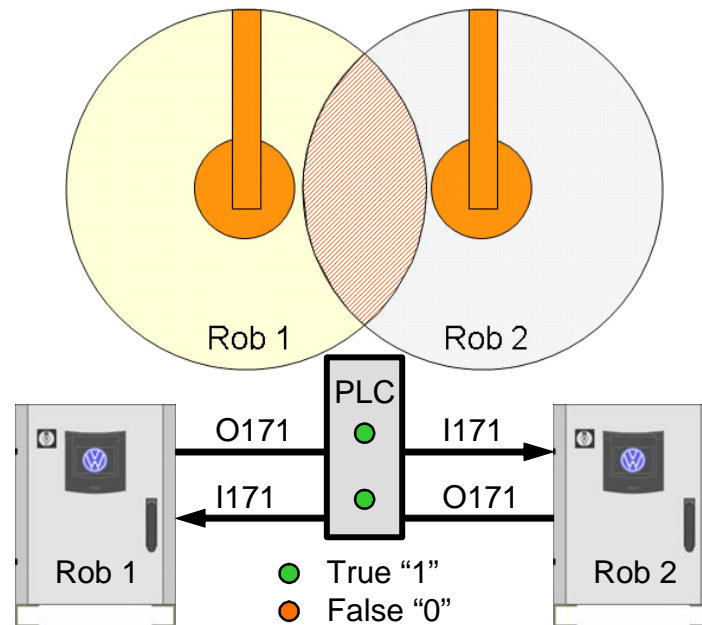




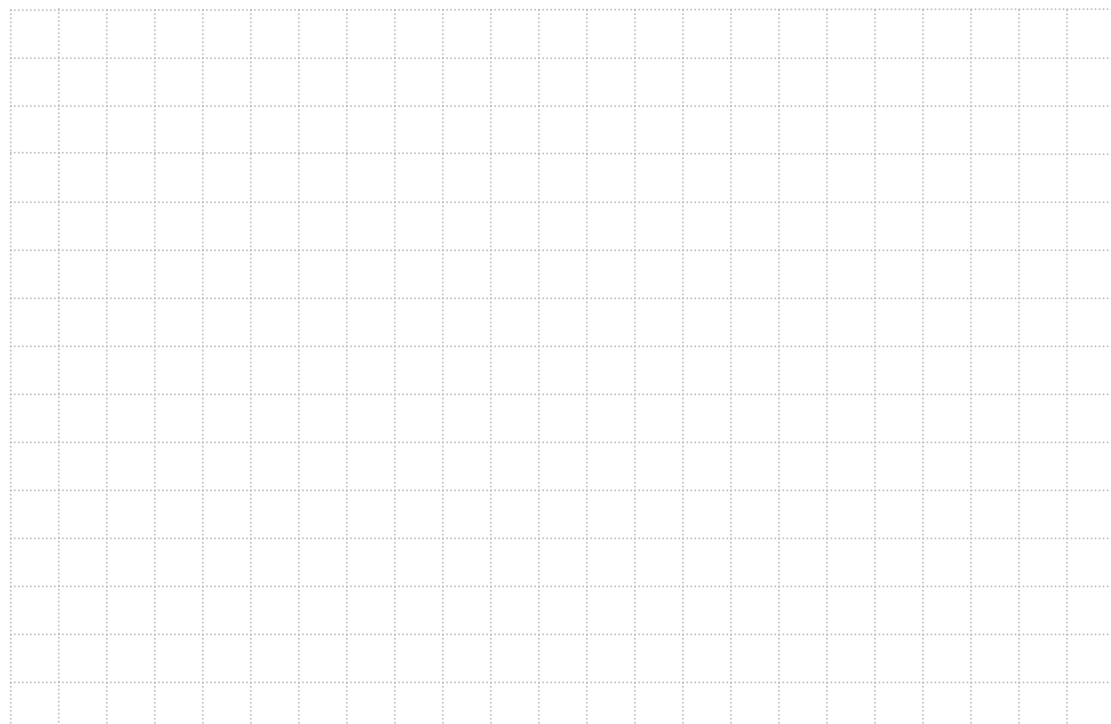
Ejemplo: Bloqueo con PLC (2)



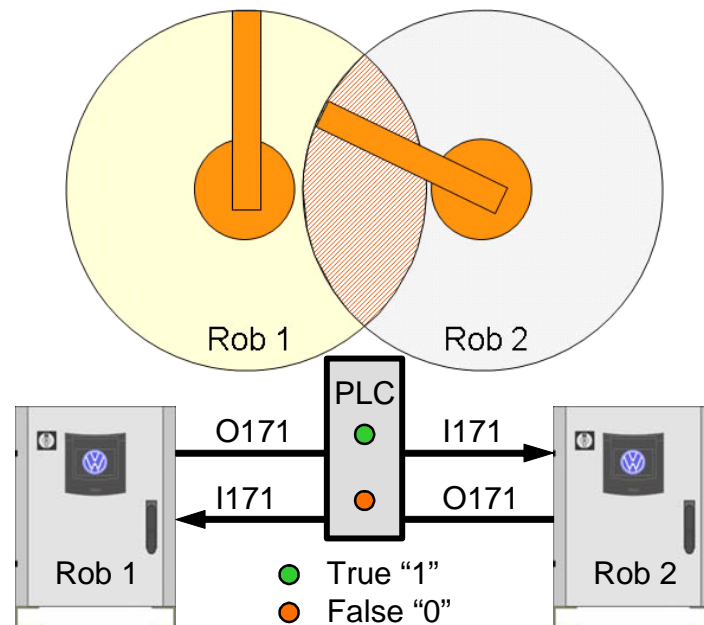
## Ejemplo: Bloqueo con PLC (3)



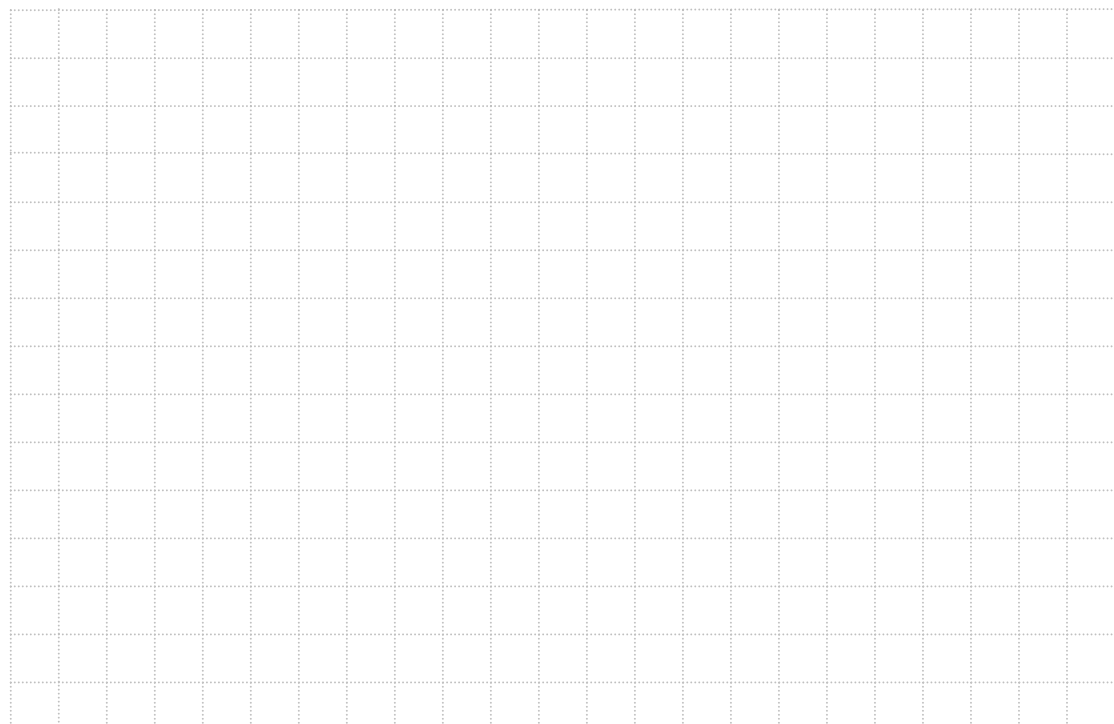
01/2006 16  
robot\_interlock\_en.ppt  
© Copyright by KUKA Roboter GmbH College



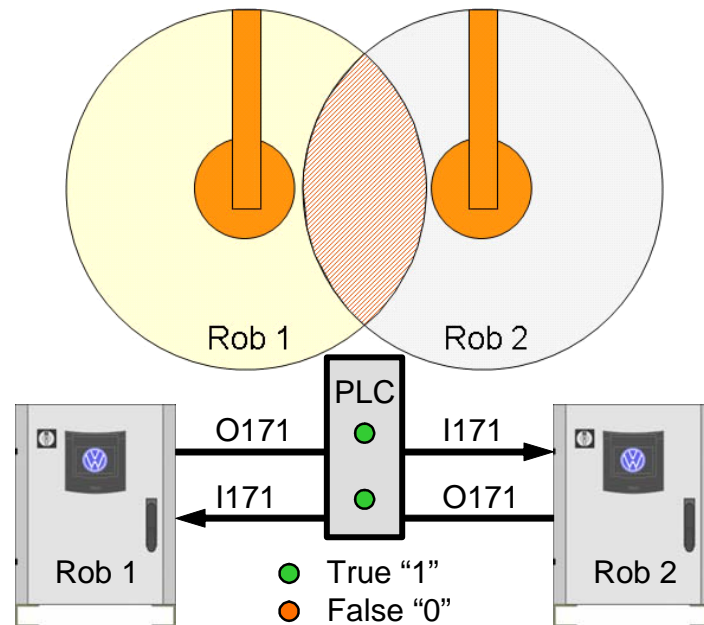
Ejemplo: Bloqueo con PLC (4)



01/2006 17  
robot\_interlock\_en.ppt  
© Copyright by KUKA Roboter GmbH College



Ejemplo: Bloqueo con PLC (5)



## Comunicación indirecta entre controles

Las señales son dirigidas via PLC sin lógica, i.e. no hay enlace I/O entre los controles de robot

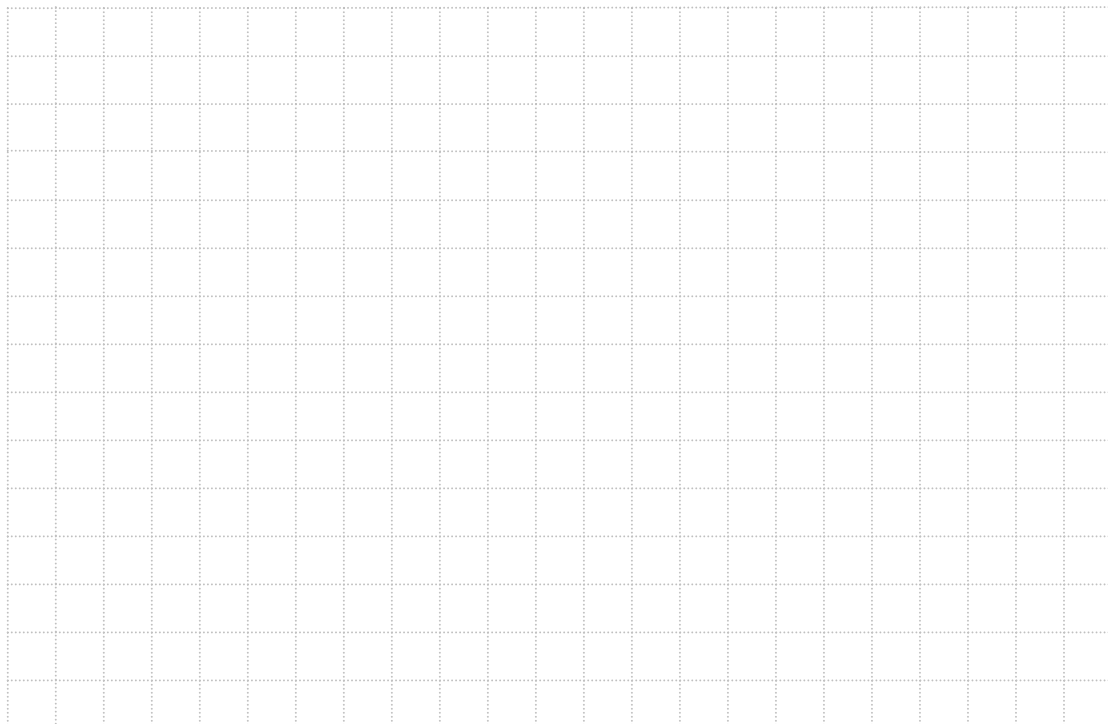
### Con tiempo de monitorización:

#### *Ventajas:*

- *Comunicación I/O sólo entre el PLC y el robot en cuestión*
- *Con un sistema estandarizado de señales no se precisa un programador de PLC*

#### *Desventajas:*

- *Alta probabilidad de bloqueo del robot si el tiempo de monitorización es demasiado alto*
- *Alta probabilidad de COLISIÓN si el tiempo de monitorización es demasiado bajo*
- *Elevado tiempo de comunicación debido al tiempo de monitorización*



## Comunicación indirecta entre controles

Las señales son dirigidas via PLC sin lógica, i.e. no hay enlace I/O entre los controles de robot

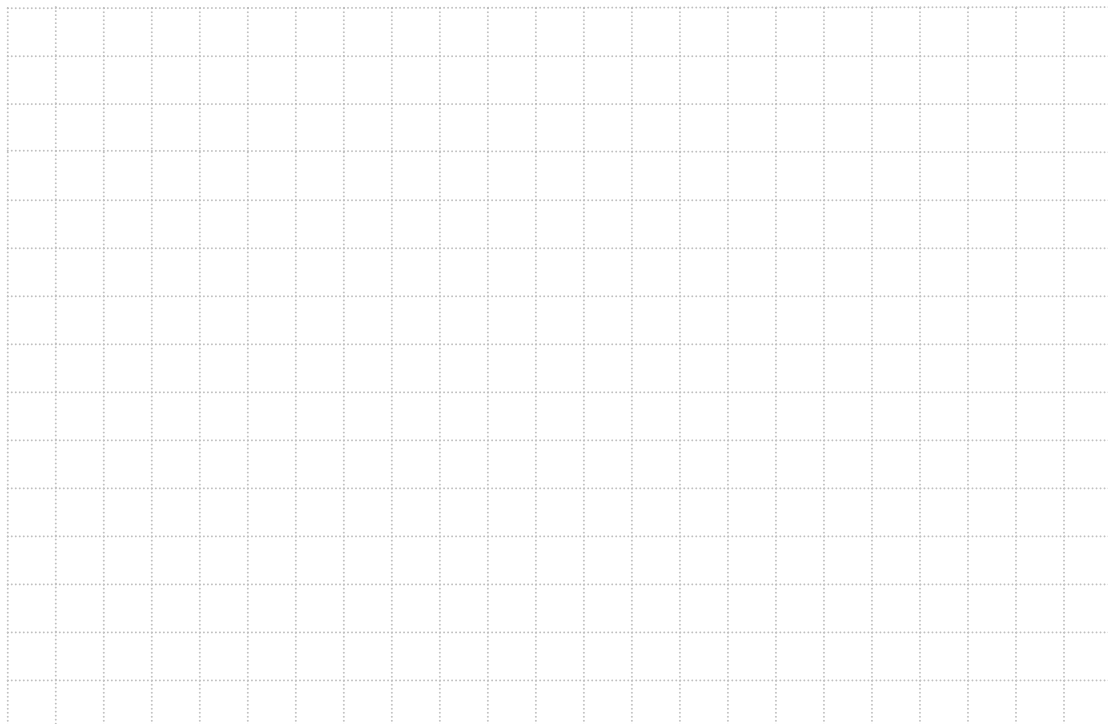
### Sin tiempo de monitorización:

#### *Ventajas:*

- *Comunicación I/O sólo entre el PLC y el robot en cuestión*
- *Con un sistema estandarizado de señales no se precisa un programador de PLC*

#### *Desventajas:*

- *Alto riesgo de COLISIÓN por incremento de la cantidad de señales tratadas por PLC (scan, secuencia de programa)*



## Control de prioridades de anticolisión via PLC

Comunicación I/O sólo entre PLC y robot. Las señales se enlazan por medio de operaciones lógicas en el PLC. Se define en la lógica del PLC qué robot tiene prioridad de todos los que han solicitado simultáneamente ocupar el área de trabajo compartida.

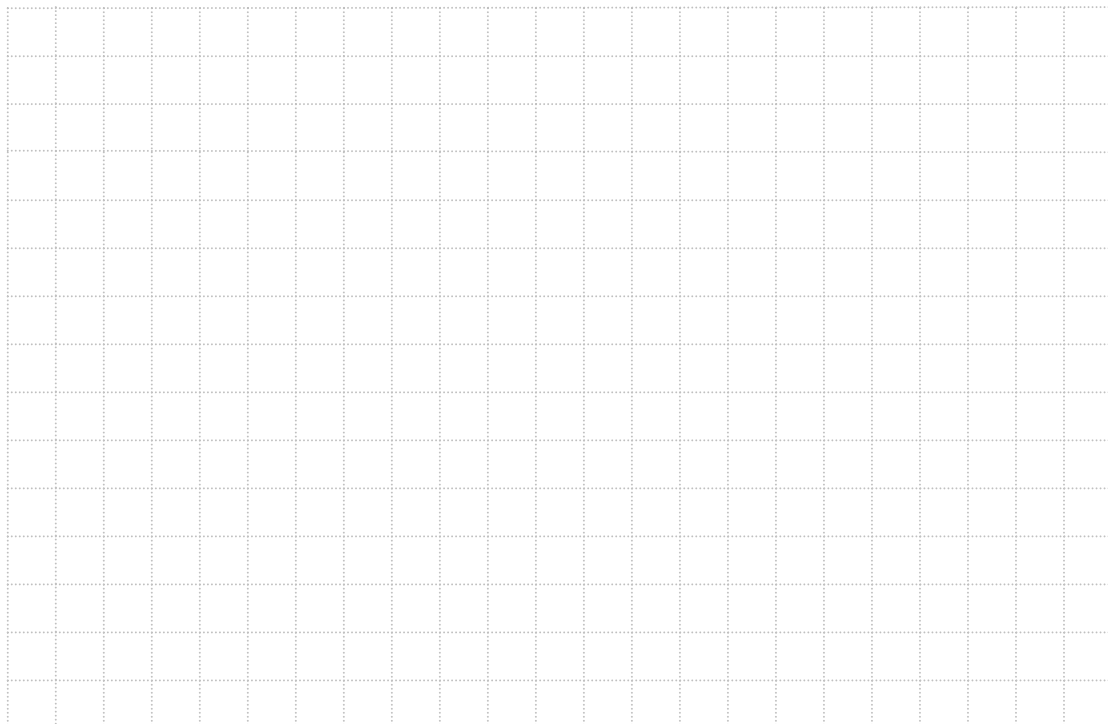
### Solicitud de permiso ↔ Permiso concedido

#### *Ventajas:*

- *Comunicación I/O sólo entre el PLC y el robot en cuestión*
- *Los robots no colisionan ni se bloquean*

#### *Desventajas:*

- *La programación requiere un programador de robot y otro de PLC*
- *Tiempo de comunicación = 1 scan de PLC scan más el tiempo de reacción del robot*



## Control de prioridades de anticollisión via PLC

Comunicación I/O sólo entre PLC y robot. Las señales se enlazan por medio de operaciones lógicas en el PLC. Se define en la lógica del PLC qué robot tiene prioridad de todos los que han solicitado simultáneamente ocupar el área de trabajo compartida.

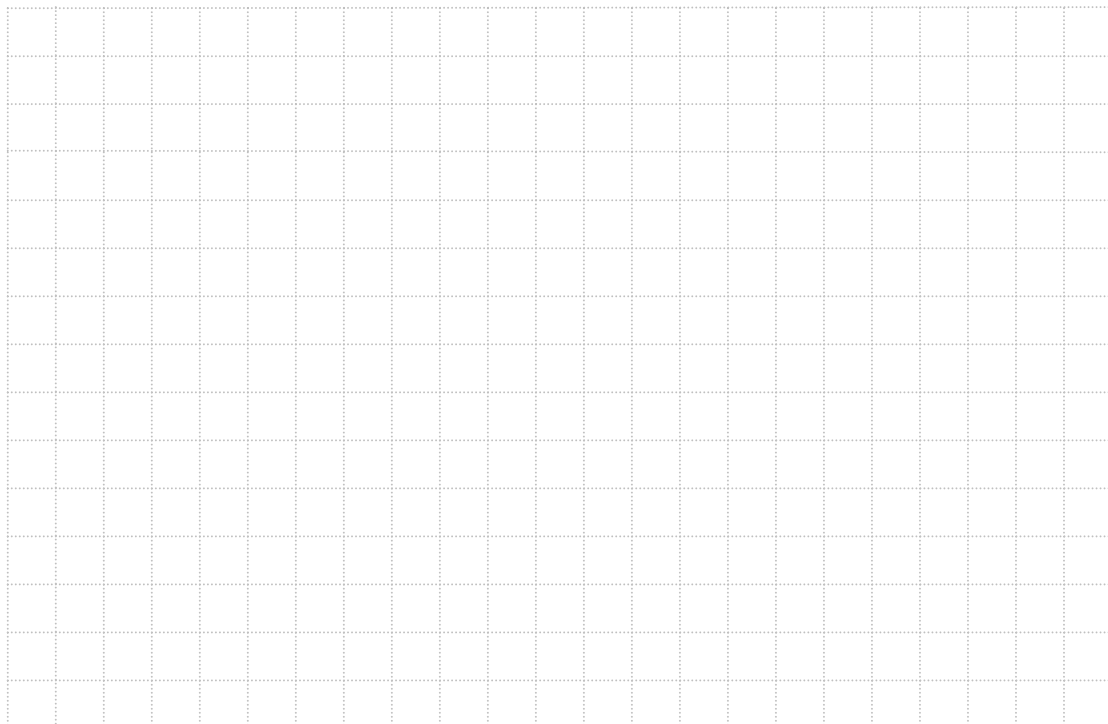
### Solicitud de permiso $\leftrightarrow$ Permiso concedido + señal "In Space"

#### *Ventajas:*

- *Comunicación I/O sólo entre el PLC y el robot en cuestión*
- *Los robots no colisionan ni se bloquean*
- *La señal "In Space" proporciona información adicional de que el robot en cuestión está dentro de la zona de trabajo compartida*

#### *Desventajas:*

- *Se requiere un programador de PLC y otro de robot*
- *Tiempo de comunicación = 1 scan de PLC scan más el tiempo de reacción del robot*
- *Más complejidad ya que intervienen más señales*





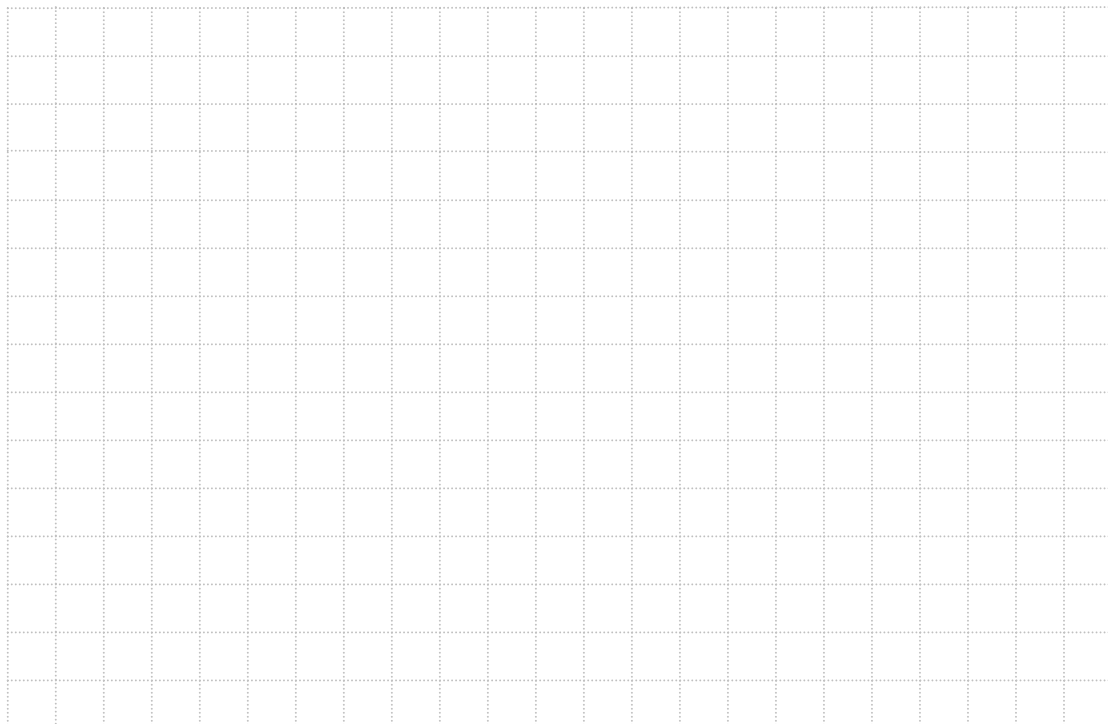
## **2.2 Monitorización de los Workspace**

## Anti-colisión con un robot: Vigilancia del área de trabajo

Opciones de bloqueo entre un robot y un área de trabajo: **Vigilancia del área de trabajo**

Pueden monitorizarse hasta 8 áreas de trabajo cúbicas o específicas de eje. Estas áreas de trabajo pueden solaparse para definir volúmenes más complejos.

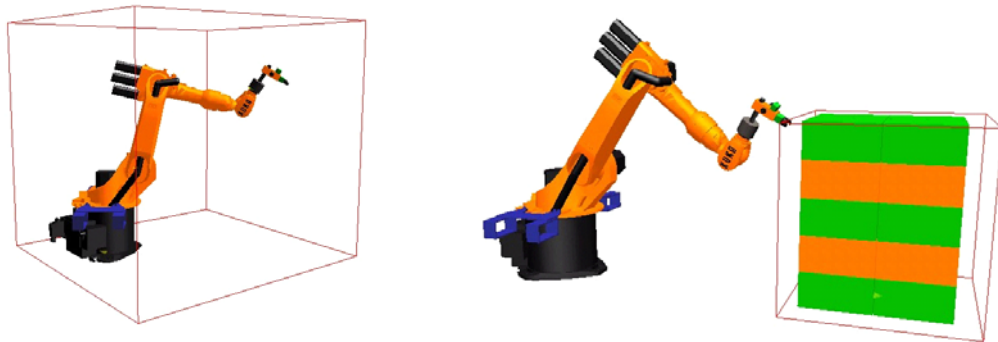
Funcionamiento:  
Violación del área de trabajo → Señal de salida → Procesamiento en KRL o PLC → opción: parar el robot y emitir mensaje



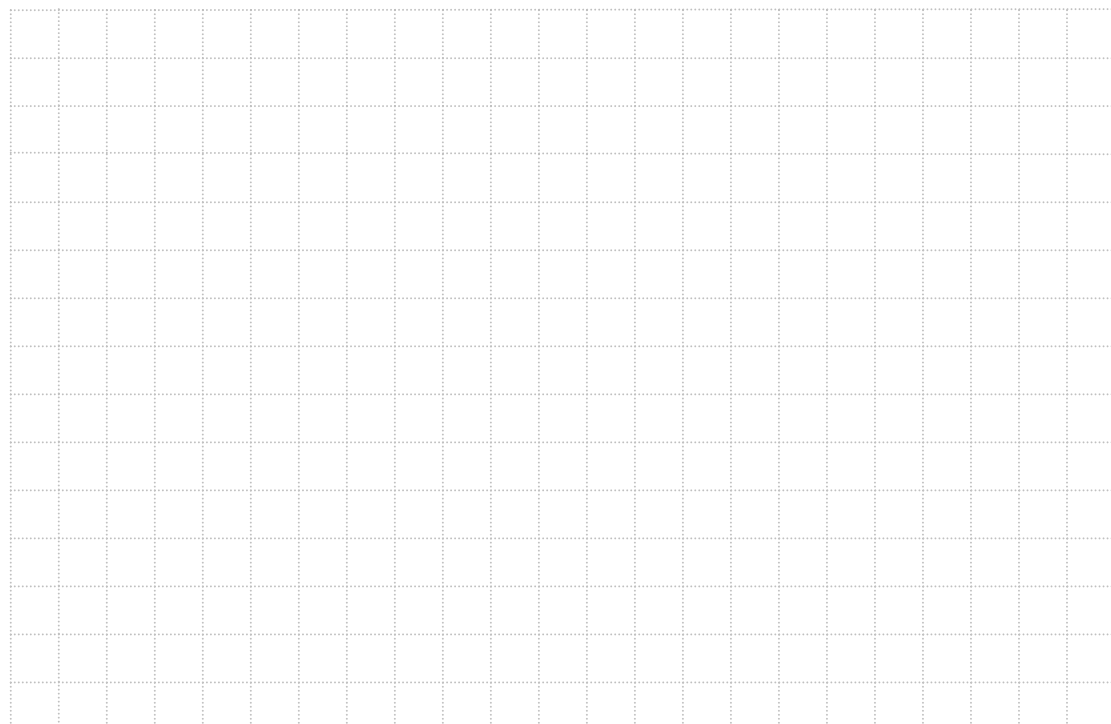
## Monitorización del área de trabajo cartesiana

Opciones de bloqueo entre robot y área de trabajo: **\$WORKSPACE**

El sistema se puede ajustar para monitorizar si el TCP seleccionado entra o abandona el área definida.

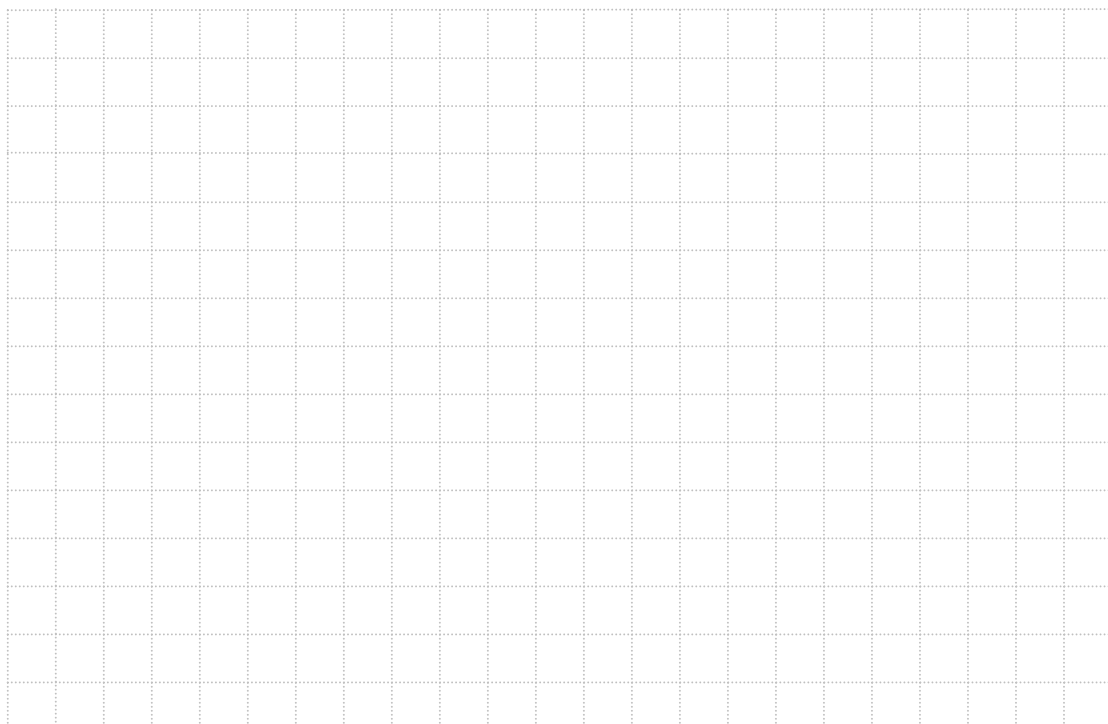
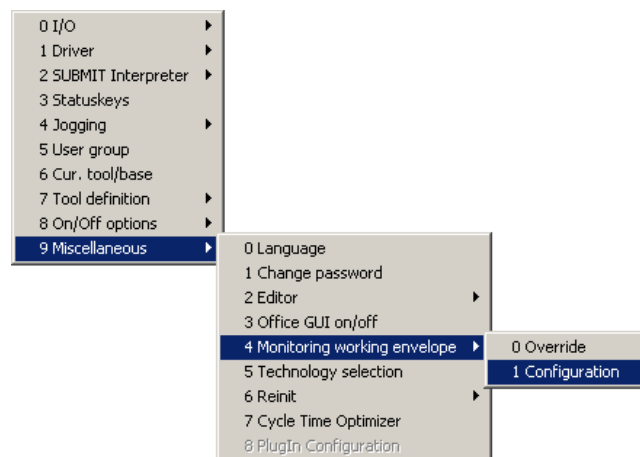


**CAUTION:** La utilización de datos incorrectos en “\$TOOL” puede acarrear consecuencias inesperadas



## Monitorización del área de trabajo cartesiana

### Guía de configuración: CONFIGURACIÓN



## Monitorización del área de trabajo cartesiana

The screenshot shows the 'Cartesian workspaces' configuration window. It includes a 'No.' field with a dropdown arrow, a 'Name' field containing 'WORKSPACE 1', and an 'Origin' section with input fields for X (500.00), Y (500.00), Z (1000.00), A (0.00), B (0.00), and C (0.00). Below this is a 'Distance to origin' section with input fields for X1 (100.00), X2 (-100.00), Y1 (100.00), Y2 (-100.00), Z1 (100.00), and Z2 (-100.00). At the bottom is a 'Mode' dropdown menu set to 'OUTSIDE\_STOP'. Four orange callout boxes with arrows point to specific fields: 'Núm área de trab. (1-8)' points to the 'No.' dropdown; 'Nombre' points to the 'Name' field; 'Origen del área de trabajo relativo al sistema de coordenadas universal' points to the 'Origin' section; 'Dimensiones del área de trabajo relativo al origen especificado arriba' points to the 'Distance to origin' section; and 'Modo de monitorización del área de trabajo' points to the 'Mode' dropdown.

Cartesian workspaces					
No.	Name: WORKSPACE 1				
Origin					
X:	500.00	A:	0.00		
Y:	500.00	B:	0.00		
Z:	1000.00	C:	0.00		
Distance to origin					
X1:	100.00	X2:	-100.00		
Y1:	100.00	Y2:	-100.00		
Z1:	100.00	Z2:	-100.00		
Mode:					
OUTSIDE_STOP					

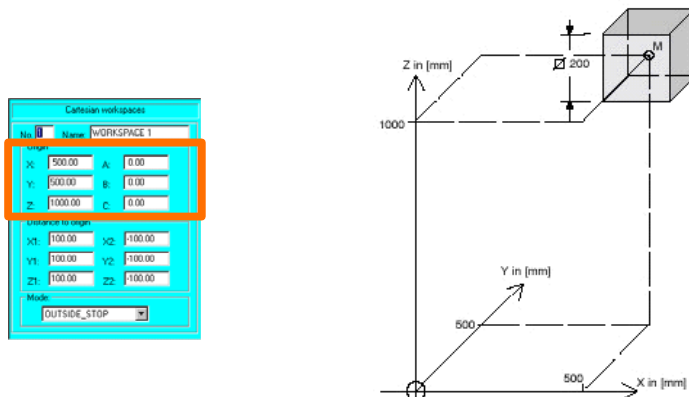


## Monitorización del área de trabajo cartesiana - \$WORKSPACE

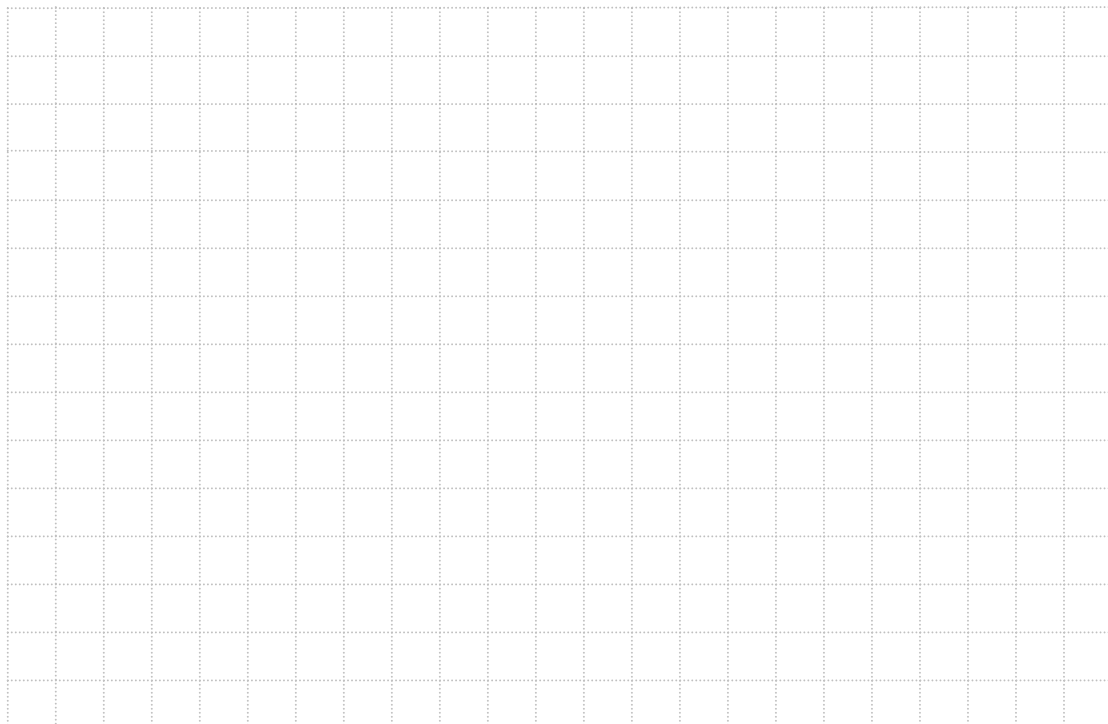
Estructura de \$WORKSPACE / origen

**X, Y, Z :** Posición del área de trabajo en los ejes principales, relativo al sistema de coordenadas universales

**A, B, C :** Orientación del área de trabajo relativa al sistema de coordenadas universales



**\$WORKSPACE[n]={X 500,Y 500, Z 1000, A 0, B 0, C 0, X1 ... }**

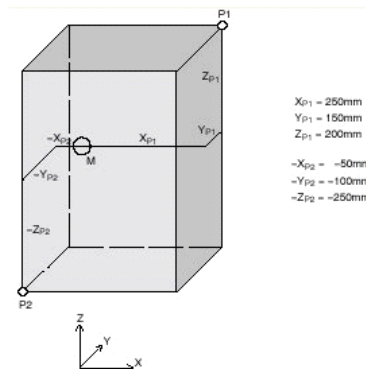
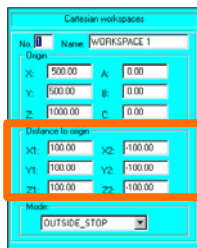


## Monitorización del área de trabajo cartesiana - \$WORKSPACE

Estructura del \$WORKSPACE / distancia desde el origen

**X1, Y1, Z1** : Define  $\Delta x1$ ,  $\Delta y1$  and  $\Delta z1$  relativo al primer punto y abre el cuboide

**X2, Y2, Z2** : Define  $\Delta x2$ ,  $\Delta y2$  and  $\Delta z2$  relativo al primer punto y expande el cuboide



Xp1 = 250mm  
Yp1 = 150mm  
Zp1 = 200mm  
-Xp2 = -50mm  
-Yp2 = -100mm  
-Zp2 = -250mm

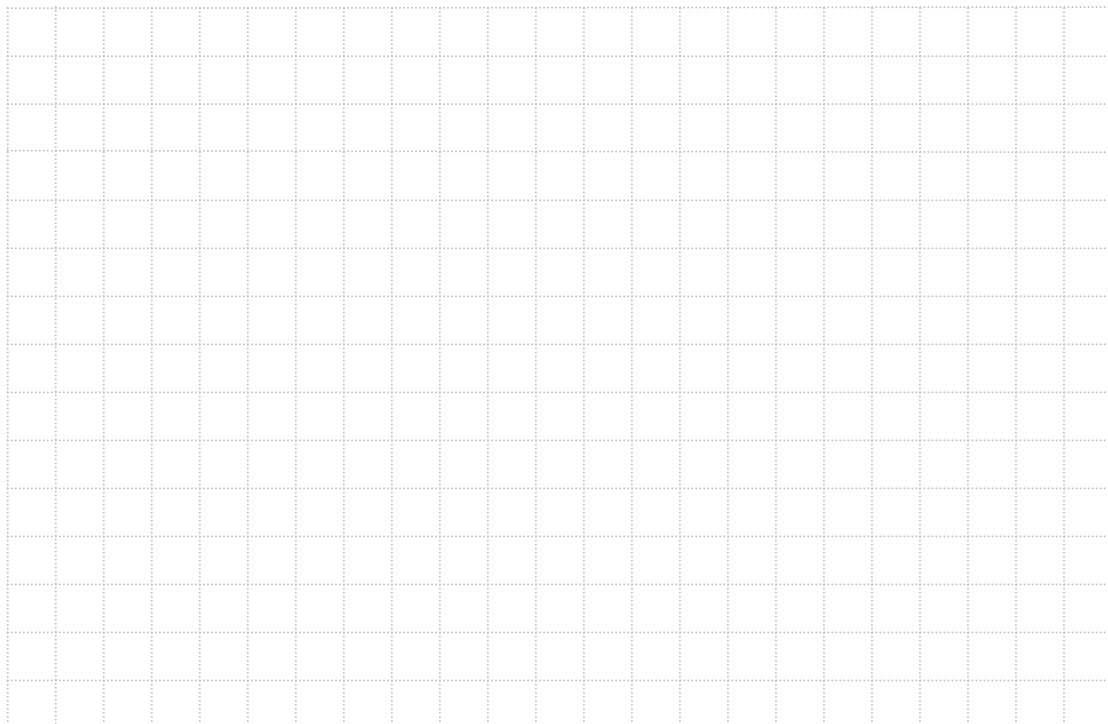
**\$WORKSPACE[n]={... X1 250,Y1 150, Z1 200, X2 -50, Y2 -100, Z2 -250, MODE ... }**

01/2006

6

robot\_workspace\_en.ppt

© Copyright by KUKA Roboter GmbH College



## Monitorización del área de trabajo cartesiana - \$WORKSPACE

### Opciones para el ajuste del "MODE"

#OFF:

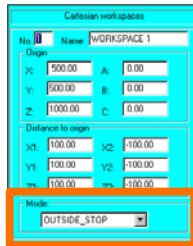
Sin monitorización

#INSIDE:

La salida especificada se activa si el TCP se encuentra **dentro** del área de trabajo

#OUTSIDE:

La salida especificada se activa si el TCP se encuentra **fuera** del área de trabajo



#INSIDE\_STOP:

La salida especificada se activa si el TCP se encuentra **dentro** del área de trabajo; además el robot se para

#OUTSIDE\_STOP:

La salida especificada se activa si el TCP se encuentra **fuera** del área de trabajo; además el robot se para

**\$WORKSPACE[n]={..., MODE #INSIDE }**

01/2006

7

robot\_workspace\_en.ppt

© Copyright by KUKA Roboter GmbH College





## Monitorización del área de trabajo específica de los ejes

Las áreas definidas por los límites de software pueden ajustarse aún más utilizando esta función con el fin de proteger el robot, la herramienta o la pieza de trabajo.

El grado de libertad de cada eje depende directamente de la posición actual del mismo.

Particularmente para:

- Robots montados en pared
- Robots montados en pedestal
- Robots de paletizado



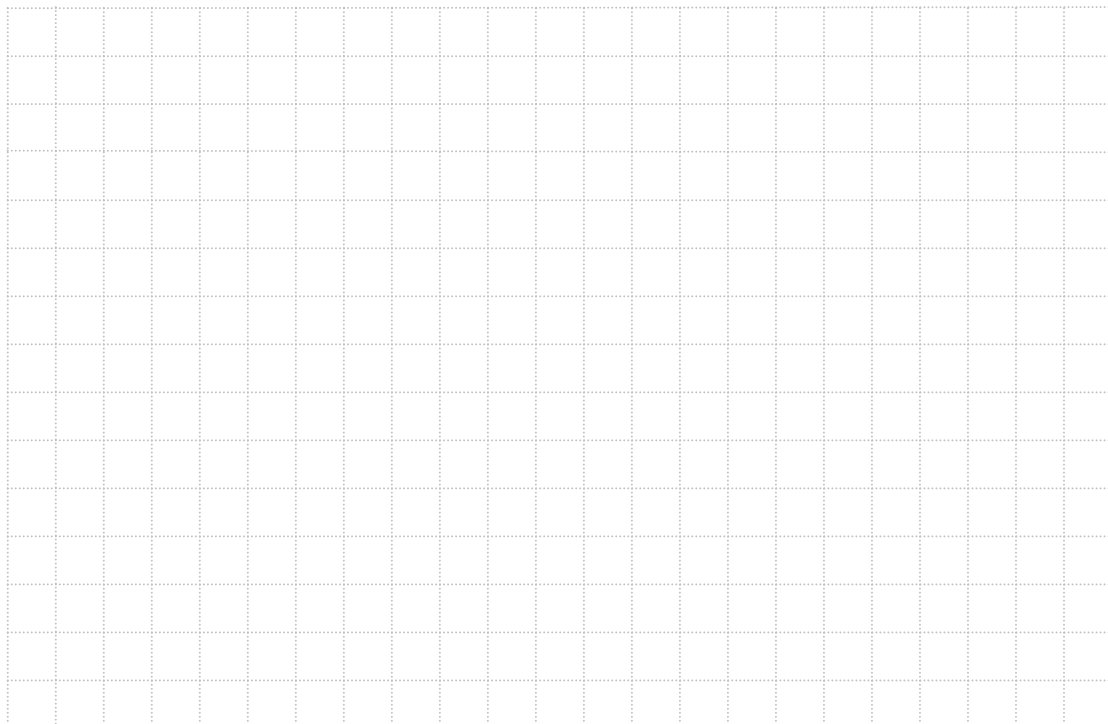
**En modo de desplazamiento manual se produce en frenado por rampa al violar un área de trabajo; en el resto de modos el frenado es dinámico.**

01/2006

8

robot\_workspace\_en.ppt

© Copyright by KUKA Roboter GmbH College



## Monitorización del área de trabajo específica de los ejes

**Núm área de trab. (1-8)** →

**Especificación del límite superior y el inferior del ángulo de cada eje** →

**Modo de monitorización del área de trabajo** →

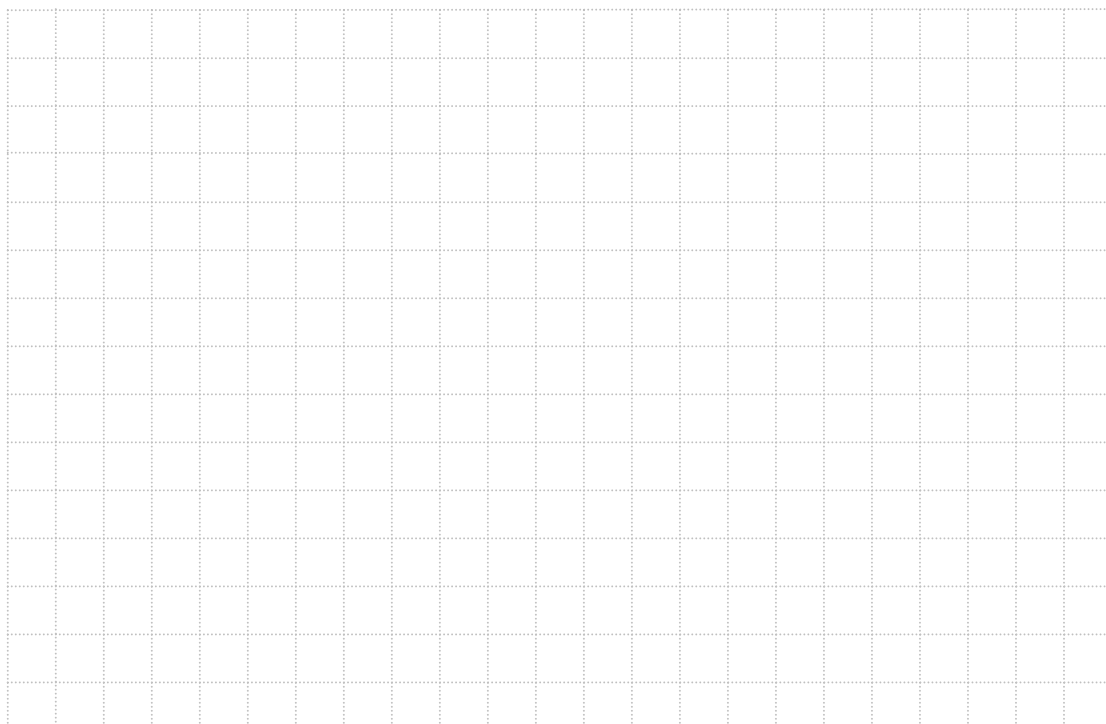
**Nombre** →

Axis specific workspaces

No.  Name:

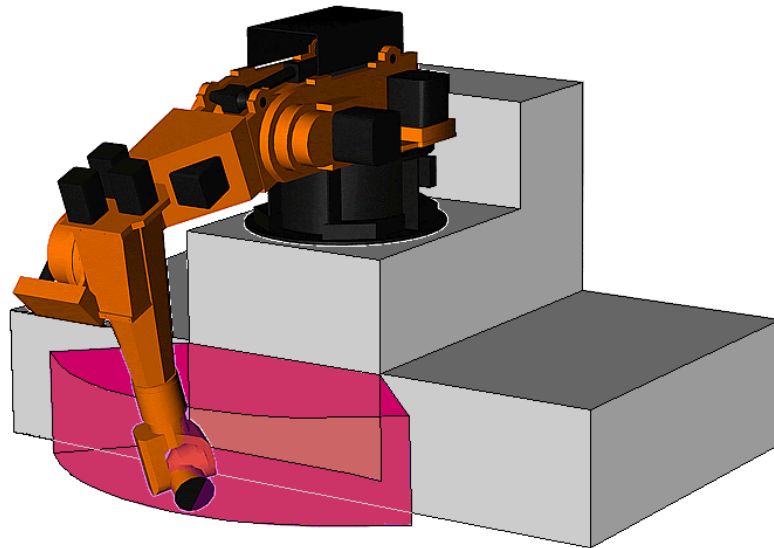
Axis	Min	Max	Min	Max
A1:	-160.00	-45.00	E1:	0.00
A2:	20.00	35.00	E2:	0.00
A3:	0.00	0.00	E3:	0.00
A4:	0.00	0.00	E4:	0.00
A5:	0.00	0.00	E5:	0.00
A6:	0.00	0.00	E6:	0.00

Mode:



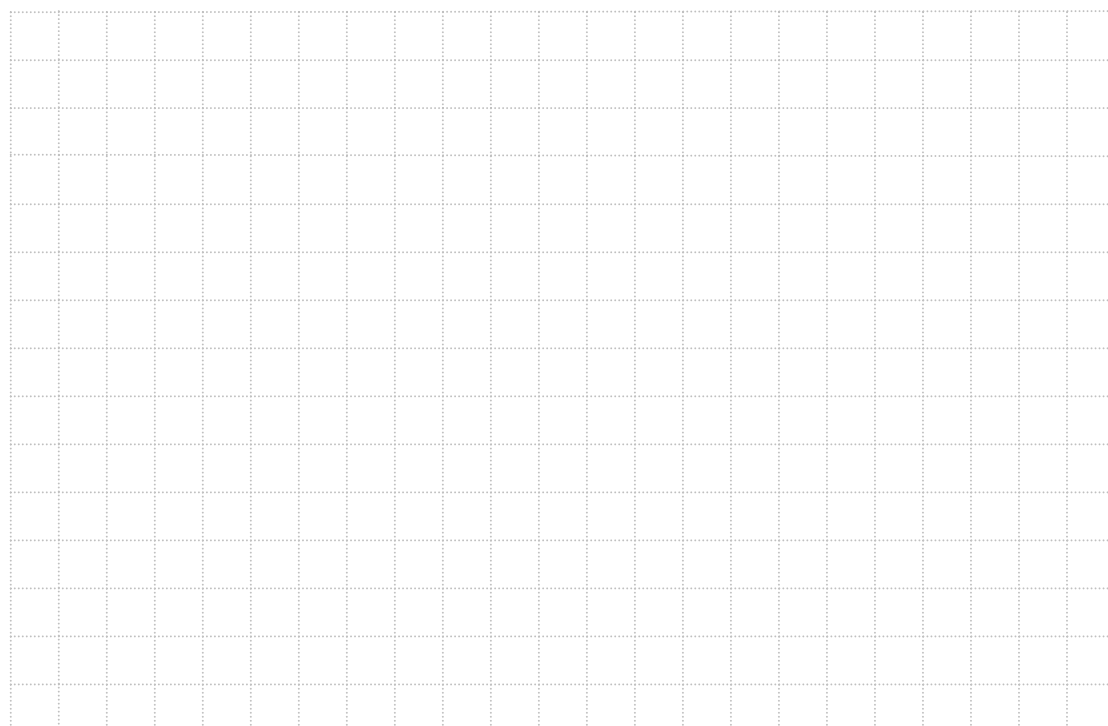
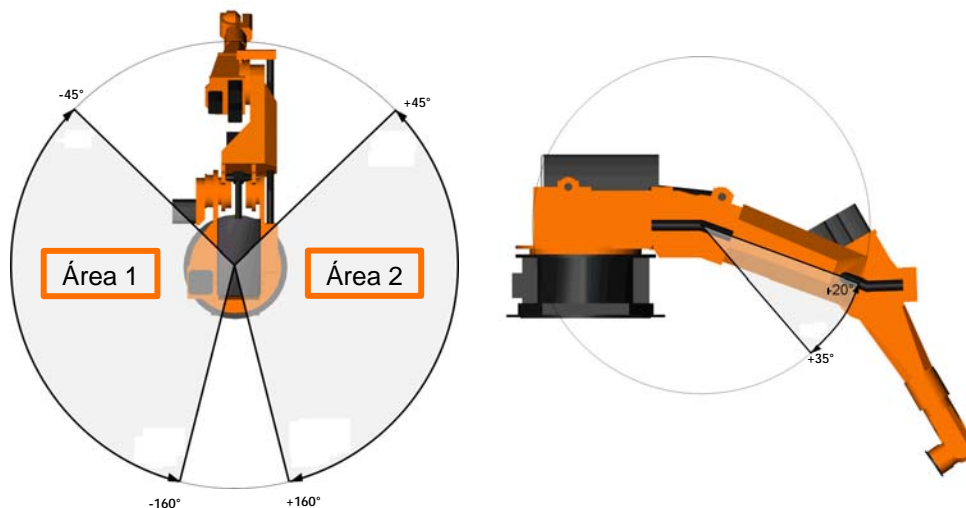
## Monitorización del área de trabajo específica de los ejes

Ejemplo:



## Monitorización del área de trabajo específica de los ejes

Ejemplo:



Monitorización del área de trabajo específica de los ejes

Ejemplo de configuración:

Axis specific workspaces

No.  Name:

Axis

	Min	Max		Min	Max
A1:	-160.00	-45.00	E1:	0.00	0.00
A2:	20.00	35.00	E2:	0.00	0.00
A3:	0.00	0.00	E3:	0.00	0.00
A4:	0.00	0.00	E4:	0.00	0.00
A5:	0.00	0.00	E5:	0.00	0.00
A6:	0.00	0.00	E6:	0.00	0.00

Mode:

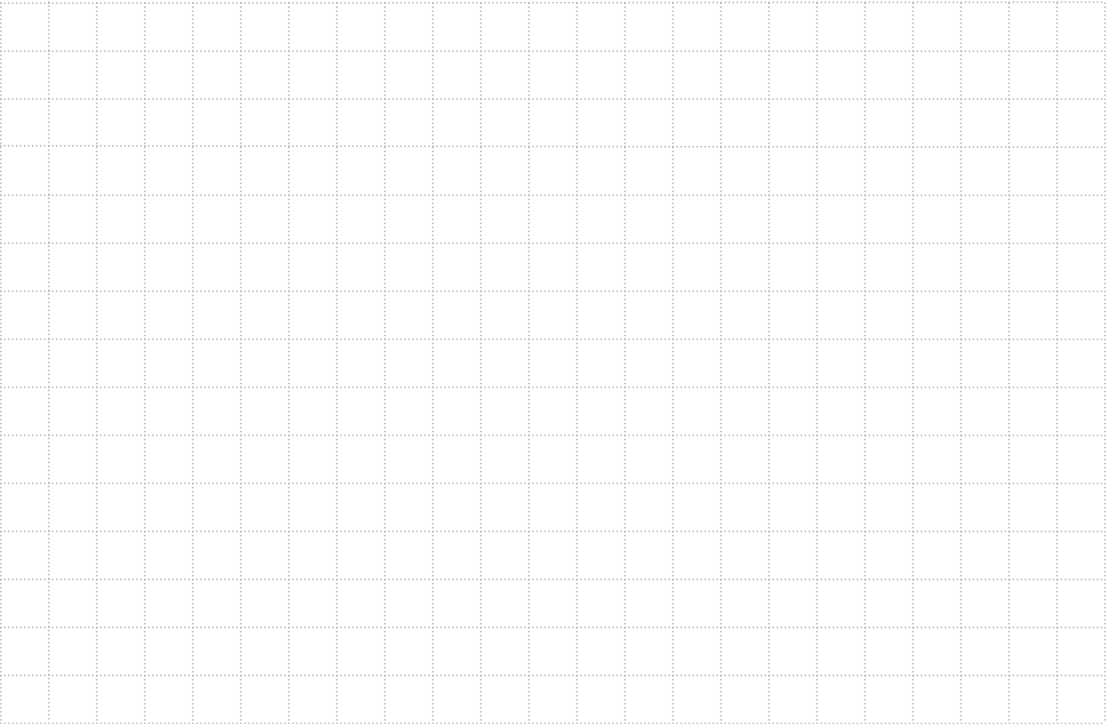
Axis specific workspaces

No.  Name:

Axis

	Min	Max		Min	Max
A1:	45.00	160.00	E1:	0.00	0.00
A2:	20.00	35.00	E2:	0.00	0.00
A3:	0.00	0.00	E3:	0.00	0.00
A4:	0.00	0.00	E4:	0.00	0.00
A5:	0.00	0.00	E5:	0.00	0.00
A6:	0.00	0.00	E6:	0.00	0.00

Mode:

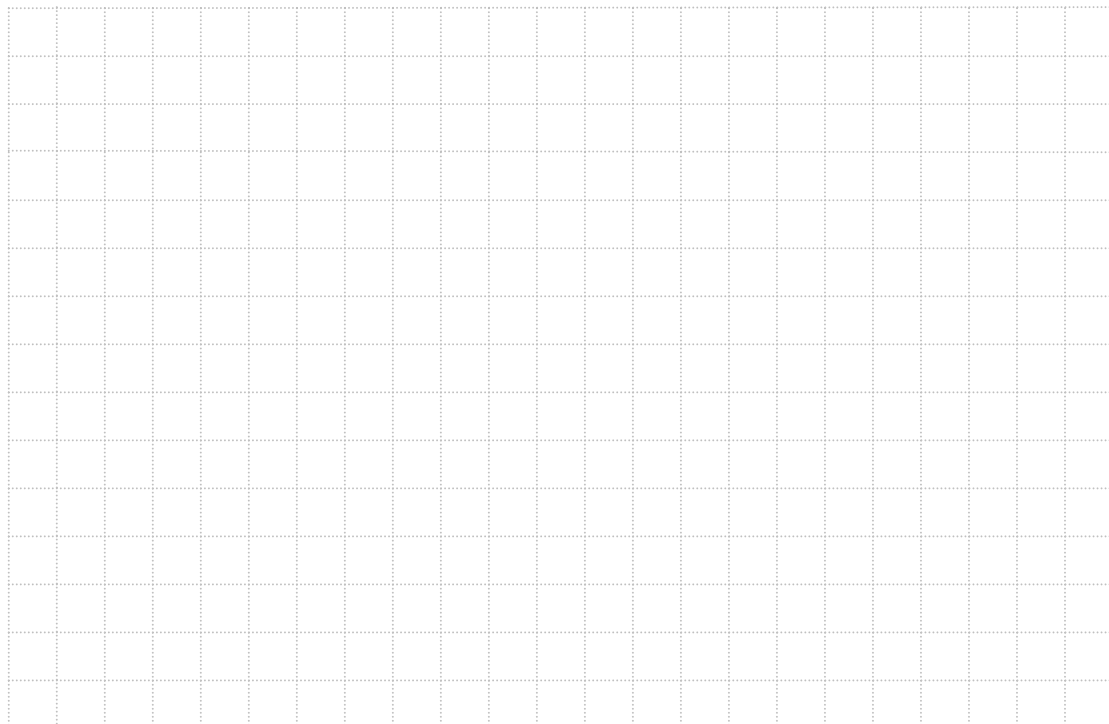


## Monitorización del área de trabajo - \$WORKSTATE

Cada salida se activa en función del ajuste del MODE.

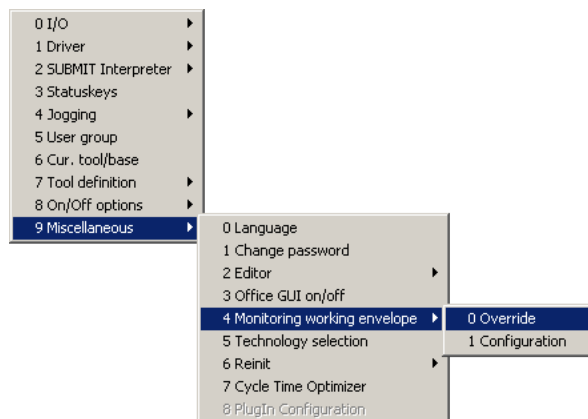
Signals	
<b>Cartesian</b>	<b>Axis spec.</b>
1: 984	1: FALSE
2: 985	2: FALSE
3: 986	3: FALSE
4: 987	4: FALSE
5: FALSE	5: FALSE
6: FALSE	6: FALSE
7: FALSE	7: FALSE
8: FALSE	8: FALSE

**SIGNAL \$WORKSTATE1 \$OUT[984]**

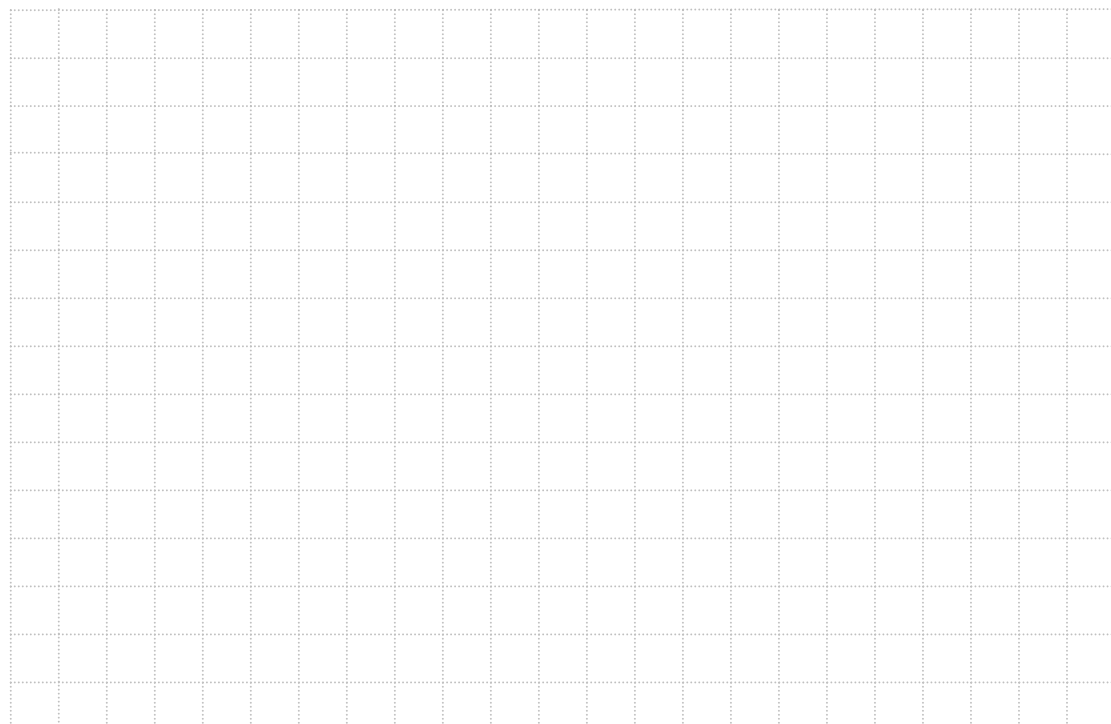


## Monitorización del área de trabajo - General

### Menú de puenteo de la vigilancia del área de trabajo



Esta función hace posible mover el robot fuera del área de trabajo vigilada. Esto sólo es posible en modo T1.



## Monitorización del área de trabajo - General

Las áreas de trabajo pueden definirse, activarse o desactivarse en los ficheros \*.SRC.

Comando KRL: `$WORKSPACE[1].MODE = #OFF`

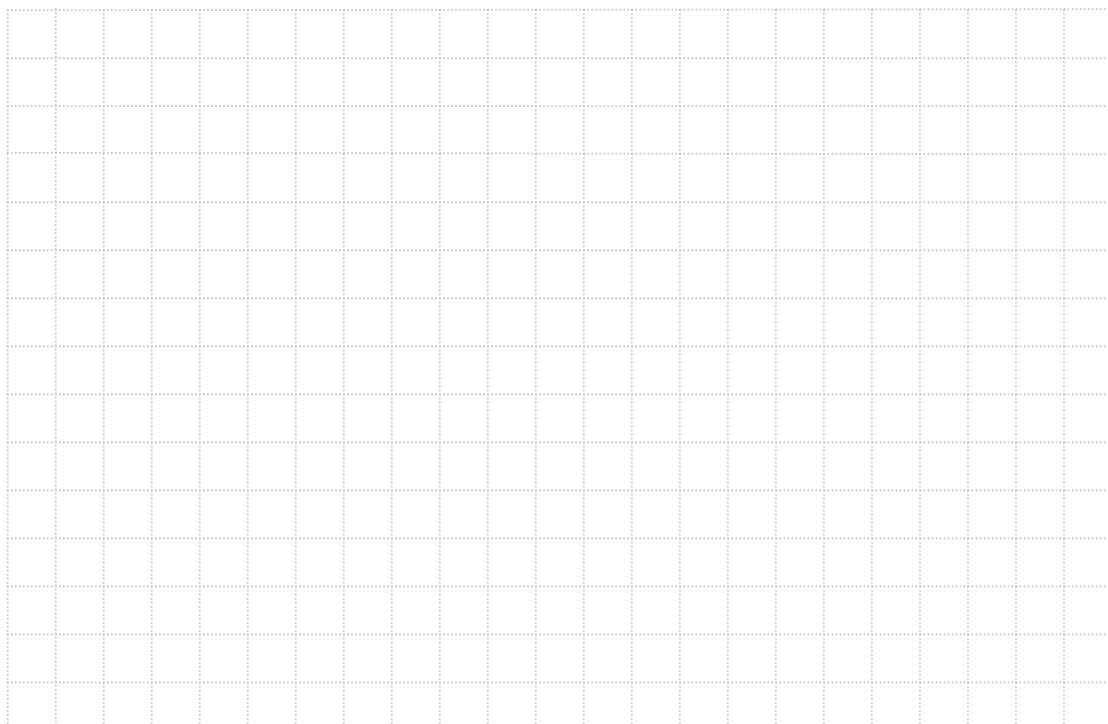
Manualmente, via modificación de variable:

Name:

Actual Value:

New Value:

Module:





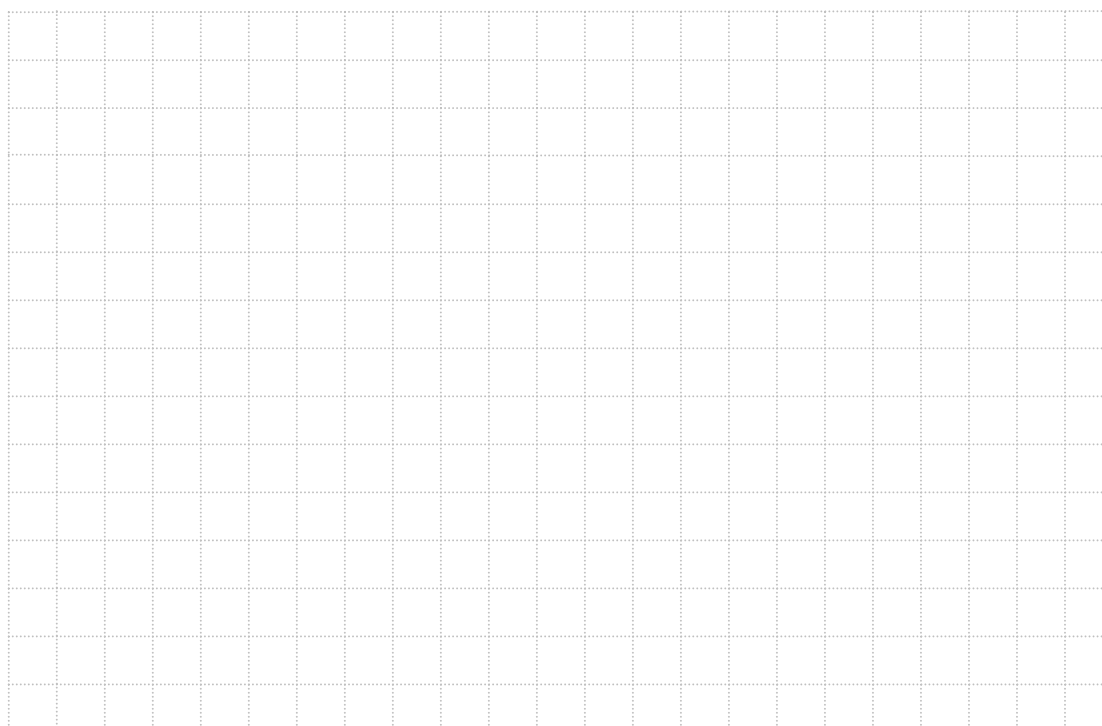
### **3. Programación de mensajes**

#### **3.1 KRL-Sintaxis para programación de mensajes**

## Programación de mensajes de usuario

En programas de aplicación y paquetes de tecnología (KRL), es posible mostrar mensajes definidos por el usuario en la KCP. Aquí podemos hacer una distinción básica de los mismos entre notificación, estado, confirmación y mensajes de diálogo. El principal requerimiento es una interface estandarizada entre el kernel del sistema y la interface de usuario.

C..	Time	no.	Source	Message
!	18:14	1	User	ERROR no air pressure



## Variables para la visualización de mensajes

Estructura para la visualización de mensajes: `$MSG_T`

```
Decl MSG_T nombre = {VALID TRUE, RELEASE TRUE, TYP #STATE , MODUL[ ] "--  
",  
KEY[ ] "-----", PARAM_TYP #VALUE, PARAM[ ] " ",  
DLG_FORMAT[ ] " ", ANSWER 0}
```

**MSG\_T** → Nombre fijo de la estructura

**nombre** → Puede seleccionarse libremente para mensajes personalizados, `$MSG_T`

## Variables para la visualización de mensajes

Estructura para la visualización de mensajes: `$MSG_T`

```
Decl MSG_T nombre = {VALID TRUE,RELEASE TRUE,TYP #STATE , MODUL[ ] "--",  
                     KEY[ ] "-----", PARAM_TYP #VALUE, PARAM[ ] " ",  
                     DLG_FORMAT[ ] " ", ANSWER 0}
```

**VALID** → TRUE genera mensaje;  
FALSE guarda el mensaje

**RELEASE** → TRUE borra mensajes de **TYP** (tipo) #STATE y #QUIT  
FALSE deja los mensajes



## Variables para la visualización de mensajes

Estructura para la visualización de mensajes: `$MSG_T`

```
Decl MSG_T name = {VALID TRUE,RELEASE TRUE,TYP #STATE , MODUL[ ] "--  
",  
KEY[ ] "-----", PARAM_TYP #VALUE, PARAM[ ] " ",  
DLG_FORMAT[ ] " ", ANSWER 0}
```

<b>TYP</b> →	#NOTIFY	Mensaje de notificación
	#STATE	Mensaje de estado
	#QUIT	Mensaje de confirmación
	#DIALOG	Mensaje de diálogo

**MODUL[ ]** → Origen; mostrado con el texto del mensaje, en la columna "abs." (máximo 24 caracteres).

## Variables para la visualización de mensajes

Estructura para visualización de mensajes: `$MSG_T`

```
Decl MSG_T name = {VALID TRUE, RELEASE TRUE, TYP #STATE, MODUL[ ] "--",  
KEY[ ] "-----", PARAM_TYP #VALUE, PARAM[ ] " ",  
DLG_FORMAT[ ] " ", ANSWER 0}
```

**KEY[ ]** → texto del mensaje que será mostrado (máximo 80 caracteres).

<b>PARAM_TYP</b> →	#VALUE	para textos personalizados (=default)
	#WORDS	ignora PARAM[ ]
	#KEY	para mensajes de una base de datos

## Variables para la visualización de mensajes

Estructura para la visualización de mensajes: `$MSG_T`

```
Decl MSG_T name = {VALID TRUE,RELEASE TRUE,TYP #STATE , MODUL[ ] "--  
",  
KEY[ ] "-----", PARAM_TYP #VALUE, PARAM[ ] " ",  
DLG_FORMAT[ ] " ", ANSWER 0}
```

**PARAM[ ]** → parámetro adicional que puede ser insertado en el texto del mensaje KEY[ ]. Una reserva de espacio %1 debe ser insertada en la posición deseada. Solo funciona con PARAM\_TYP #VALUE. Este parámetro adicional puede contener un máximo de 20 caracteres.

## Variables para la visualización de mensajes

Estructura para la visualización de mensajes: \$MSG\_T

```
Decl MSG_T name = {VALID TRUE, RELEASE TRUE, TYP #STATE, MODUL[ ] "--",  
KEY[ ] "-----", PARAM_TYP #VALUE, PARAM[ ] " ",  
DLG_FORMAT[ ] " ", ANSWER 0}
```

**DLG\_FORMAT[ ]** → Nombres de las softkeys. Estos deben estar separados por medio de una línea vertical | (ALT 124). Debe comprobarse que el nombre de las softkeys individuales no exceda de 10 caracteres (p.ej. max. 70 caracteres).

**ANSWER** → El número de la softkey pulsada en respuesta al mensaje de diálogo es guardado aquí. La numeración de las softkeys empieza en la izquierda (1) y se incrementa en uno para cada softkey.





## **3.2 Mensajes de aviso**

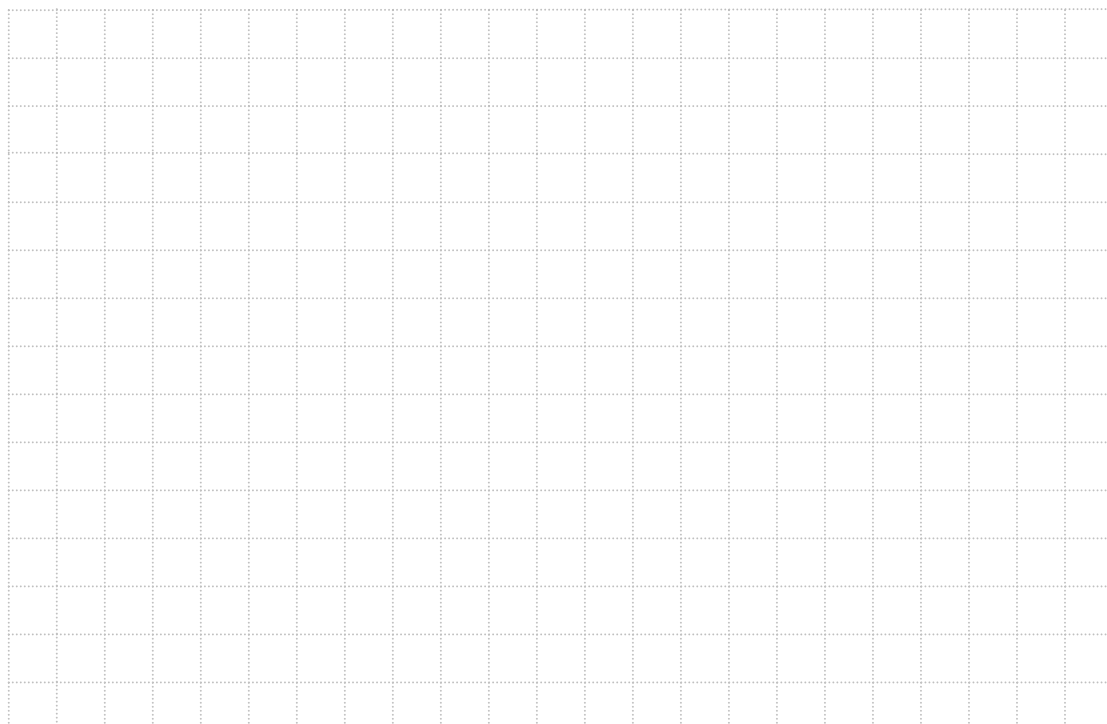
## Mensaje de notificación

```

1  DECL MSG_T EMPTY_MSG
2
3  EMPTY_MSG={MSG_T: VALID FALSE,RELEASE FALSE,TYP #NOTIFY,MODUL[] "
  ",KEY[] " ",PARAM_TYP #VALUE,PARAM[] " ",DLG_FORMAT[] " ",ANSWER 0}
4
5  ; ***** notify message *****
6  $MSG_T=EMPTY_MSG ;           initialization
7  $MSG_T.MODUL[]="USER";       text in Source
8  $MSG_T.KEY[]="no vacuum";     message text
9  $MSG_T.PARAM_TYP=#VALUE
10 $MSG_T.TYP=#NOTIFY;          notification message
11 $MSG_T.VALID=TRUE;           fire the message
12
13 WHILE $MSG_T.VALID
14   WAIT SEC 0.05
15 ENDWHILE
16 WAIT SEC 0.2
    
```

**\$MSG\_T.VALID** solo permanece a TRUE hasta que el texto del mensaje ha sido mostrado (loop sensible).

C..	Time	no.	Source	Message
!	18:14	1	USER	no vacuum




### **3.3 Mensajes de confirmación**

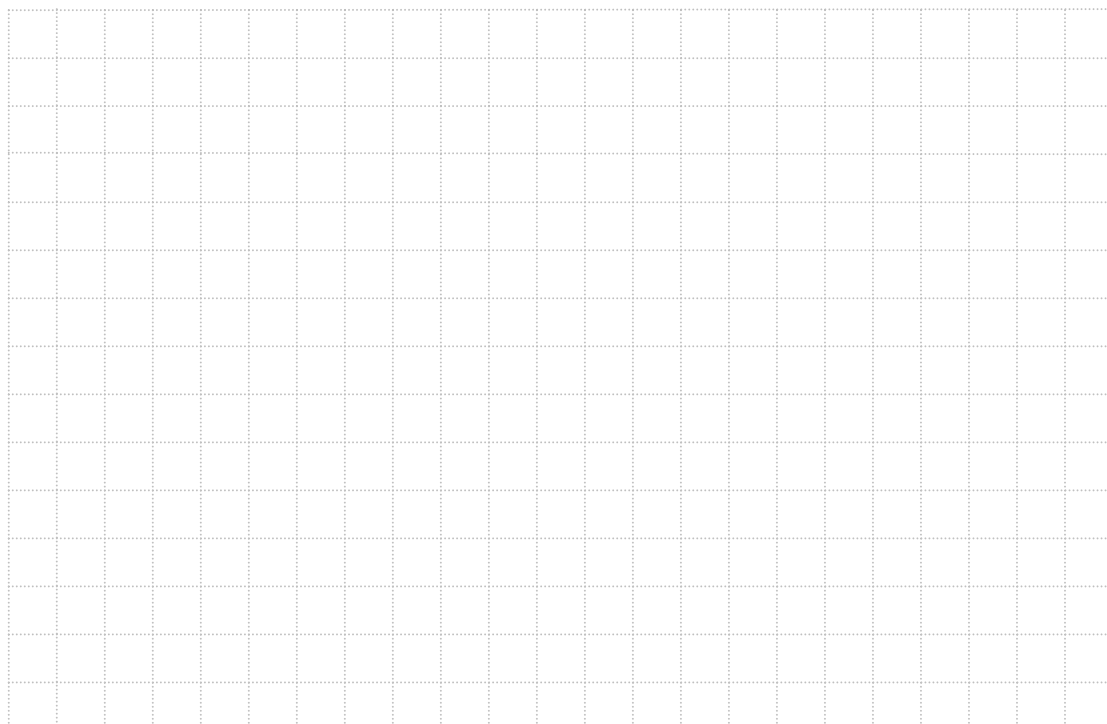
## Mensajes de confirmación

```

1  DECL MSG_T EMPTY_MSG
2
3  EMPTY_MSG={MSG_T: VALID FALSE,RELEASE FALSE,TYP #NOTIFY,MODUL[] "
  ",KEY[] " ",PARAM_TYP #VALUE,PARAM[] " ",DLG_FORMAT[] " ",ANSWER 0}
4
5  ; ***** quit message *****
6  $MSG_T=EMPTY_MSG ;           initialization
7  $MSG_T.MODUL[]="USER";       text in Source
8  $MSG_T.KEY[]="no water";      message text
9  $MSG_T.PARAM_TYP=#VALUE
10 $MSG_T.TYP=#QUIT;            acknowledgement message
11 $MSG_T.VALID=TRUE;           fire the message
12
13 WHILE $MSG_T.VALID
14   WAIT SEC 0.05
15 ENDWHILE
16 WAIT SEC 0.2
    
```

**\$MSG\_T.VALID** permanece a TRUE hasta que el mensaje de texto es confirmado (loop sensible).

C..	Time	no.	Source	Message
	18:14	1	USER	no water




## **3.4 Mensajes de estado**

## Mensaje de estado

```

1  DECL MSG_T EMPTY_MSG
2
3  EMPTY_MSG=(MSG_T: VALID FALSE,RELEASE FALSE,TYP #NOTIFY,MODUL[] "
  ",KEY[] " ",PARAM_TYP #VALUE,PARAM[] " ",DLG_FORMAT[] " ",ANSWER 0)
4
5  ; ***** status message *****
6  $MSG_T=EMPTY_MSG ;           initialization
7  $MSG_T.MODUL[]="USER";      text in Source
8  $MSG_T.KEY[]="Error air supply"; message text
9  $MSG_T.PARAM_TYP=#VALUE
10 $MSG_T.TYP=#STATE;          status message
11 $MSG_T.VALID=TRUE;          fire the message
12
13 WHILE $IN[15]==FALSE;       error input 15
14   ENDWHILE
15
16 $MSG_T.RELEASE=TRUE;        delete status message
  
```

**\$MSG\_T.VALID** permanece a TRUE hasta que **\$MSG\_T.RELEASE=TRUE** (borrar mensaje) sea generado (no loop).

C..	Time	no.	Source	Message
	18:16	1	USER	Error air supply

## Mensajes de estado

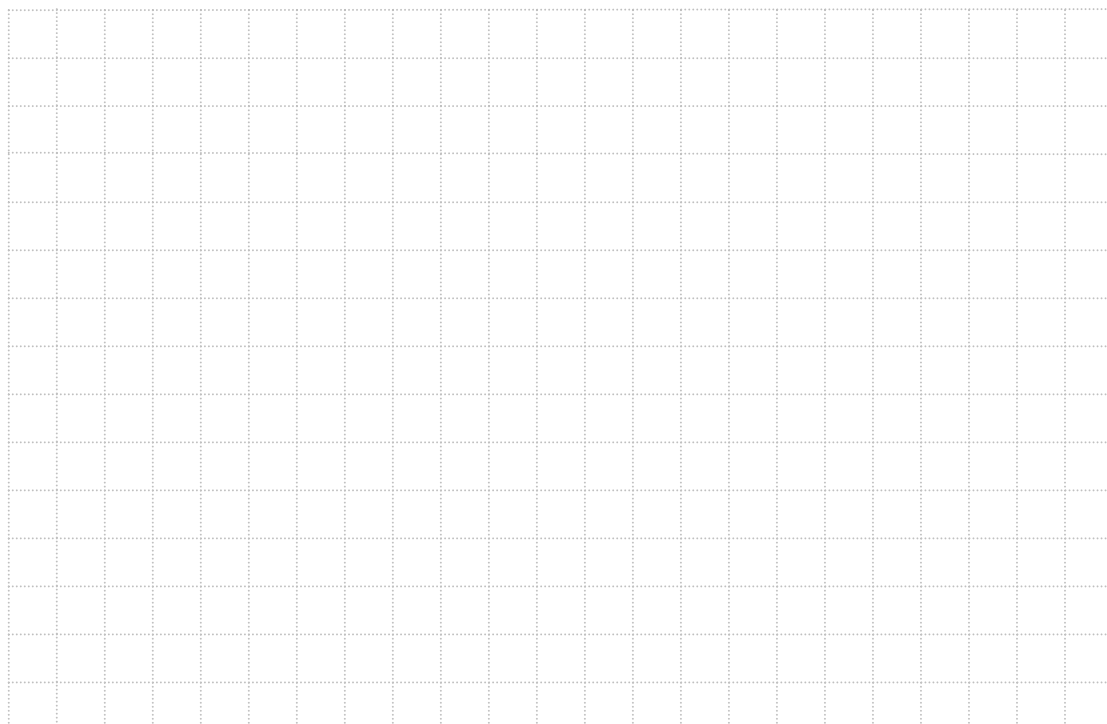
```

1  DECL MSG_T EMPTY_MSG
2
3  EMPTY_MSG={MSG_T: VALID FALSE,RELEASE FALSE,TYP #NOTIFY,MODUL[] "
4  ",KEY[] " ",PARAM_TYP #VALUE,PARAM[] " ",DLG_FORMAT[] " ",ANSWER 0}
5
6  ; ***** status message *****
7  $MSG_T=EMPTY_MSG ;           initialization
8  $MSG_T.MODUL[]="USER";       text in Source
9  $MSG_T.KEY[]="Error air supply"; message text
10 $MSG_T.PARAM_TYP=#VALUE
11 $MSG_T.TYP=#STATE;           status message
12 $MSG_T.VALID=TRUE;           fire the message
13
14 WHILE $IN[15]==FALSE;        error input 15
15   ENDWHILE
16
17 $MSG_T.RELEASE=TRUE;         delete status message

```

**\$MSG\_T.VALID permanece a TRUE hasta \$MSG\_T.RELEASE=TRUE (borrar mensaje) sea generado (no loop).**

C..	Time	no.	Source	Message



## **3.5 Mensajes de dialogo**

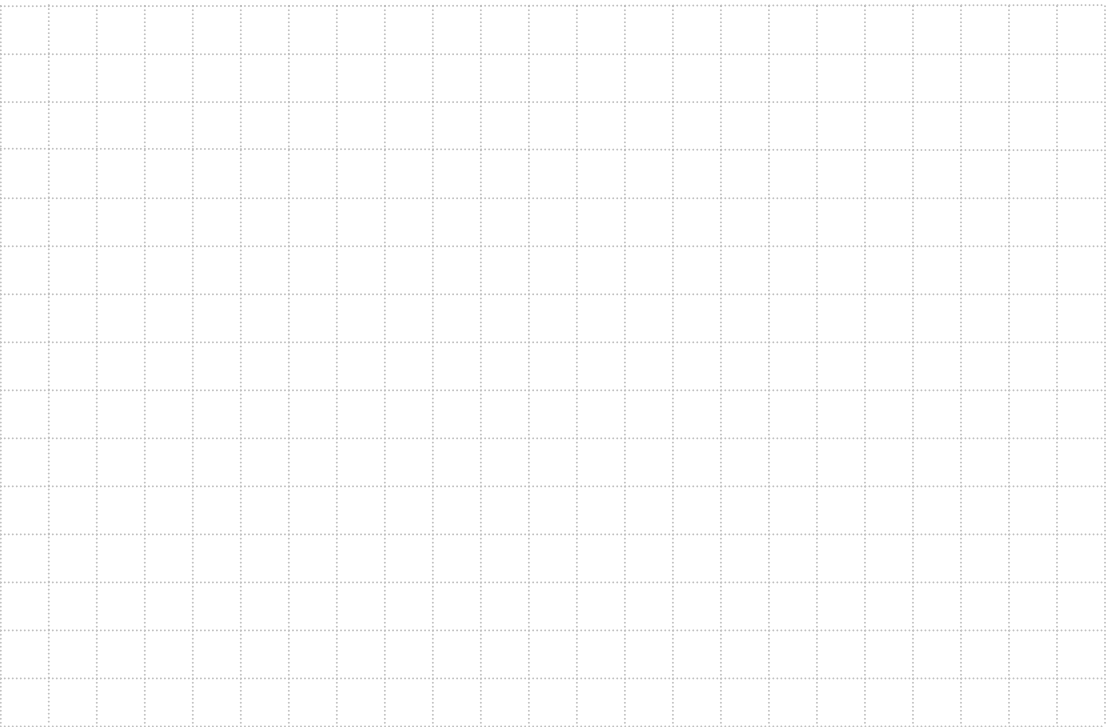
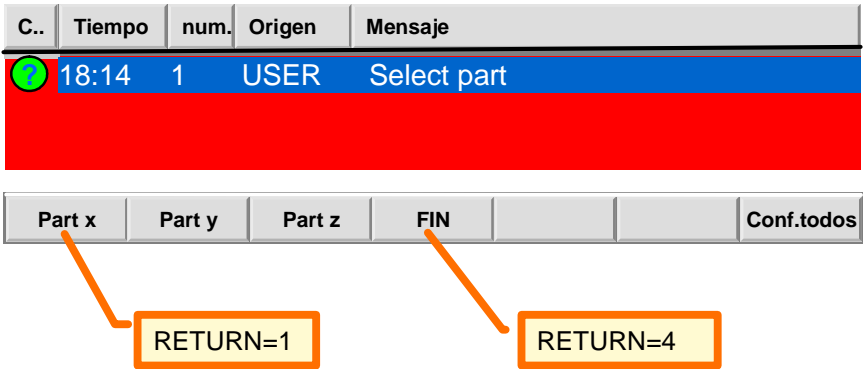


## Mensaje de diálogo

```
1  DECL MSG_T EMPTY_MSG
2  DECL INT REPLY
3  EMPTY_MSG={MSG_T: VALID FALSE,RELEASE FALSE,TYP #NOTIFY,MODUL[] "
4  ",KEY[] " ",PARAM_TYP #VALUE,PARAM[] " ",DLG_FORMAT[] " ",ANSWER 0}
5
6  ; ***** dialog message *****
7  $MSG_T=EMPTY_MSG ; initialization
8  $MSG_T.MODUL[]="USER";      text in Source
9  $MSG_T.KEY[]=",select part"; message text
10 $MSG_T.PARAM_TYP=#VALUE
11 $MSG_T.TYP=#DIALOG;         dialog message
12 $MSG_T.DLG_FORMAT[ ]="Part x|Part y|Part z|END"; labeling Softkeys
13 $MSG_T.VALID=TRUE;          fire the message
14
15 WHILE $MSG_T.VALID
16   WAIT SEC 0.05
17 ENDWHILE
18 $MSG_T.ANSWER=REPLY;        save the answer (button) in REPLY
```

**\$MSG\_T.VALID** permanece a TRUE hasta que se pulsa una softkey (loop sensible).

Mensaje de diálogo



## **3.6 Mensajes con variables**

## Mostrar un mensaje con una variable

### Condición:

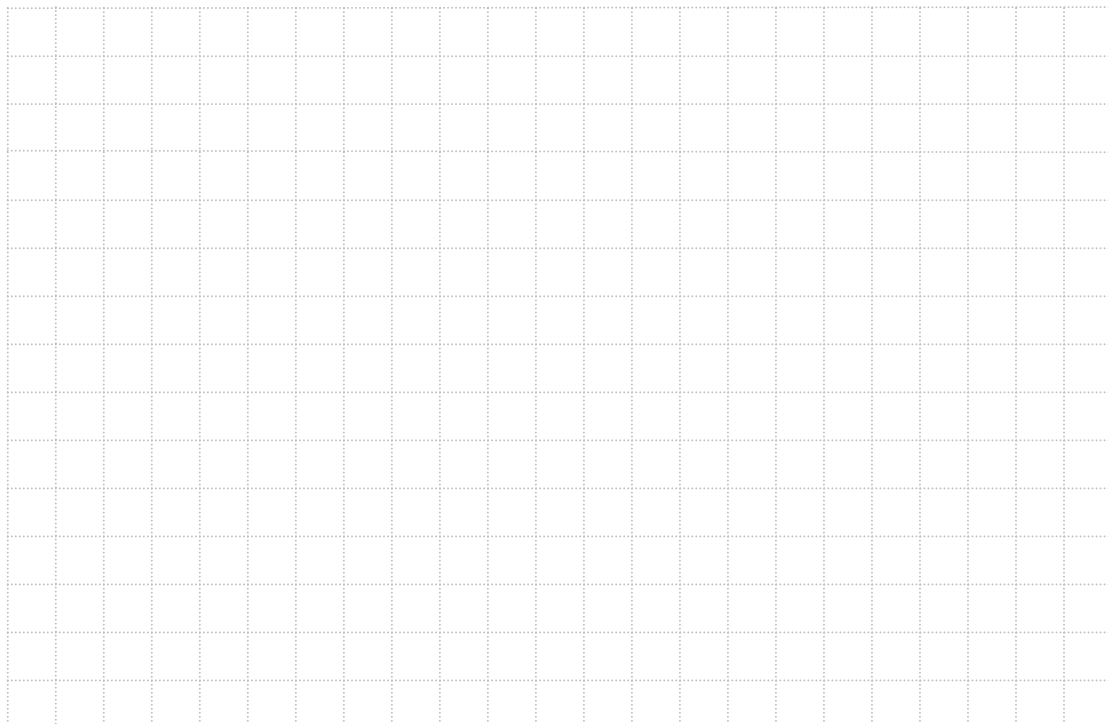
1. Una reserva de espacio **%1** debe estar presente en el texto del mensaje `$MSG_T.KEY[ ]`.  
Ejemplo: `$MSG_T.KEY[ ]="PART no. %1 is ready"`
2. El valor de `$MSG_T.PARAM[ ]` se escribe en la reserva de espacio. Se asigna el valor como string por medio de "Text".

01/2006

1

message\_parameter\_en.ppt

© Copyright by KUKA Roboter GmbH College



## Mostrando un mensaje con una variable: TEXT

### Ejemplo 1: transferencia de texto simple

```
...  
$MSG_T.KEY[ ]="Fault at %1"  
$MSG_T.PARAM[ ]="gripper"  
...  
; Generar mensaje 1  
...  
$MSG_T.PARAM[ ]="gun"  
...  
; Generar mensaje 2
```

Fallo en garra

Mensaje 1

Fallo en pinza

Mensaje 2



Mostrando un mensaje con una variable: VARIABLE

Ejemplo 2: variable tipo integer

```
INT part
...
part=part+1
$MSG_T.KEY[ ]="Part no. %1 is ready"
SWRITE($MSG_T.PARAM[ ],STATE,OFFSET,"%d",part)
; Generate message
...
```

*SWRITE se describe a continuación*

Part no. 1 is ready

1er mensaje

Part no. 2 is ready

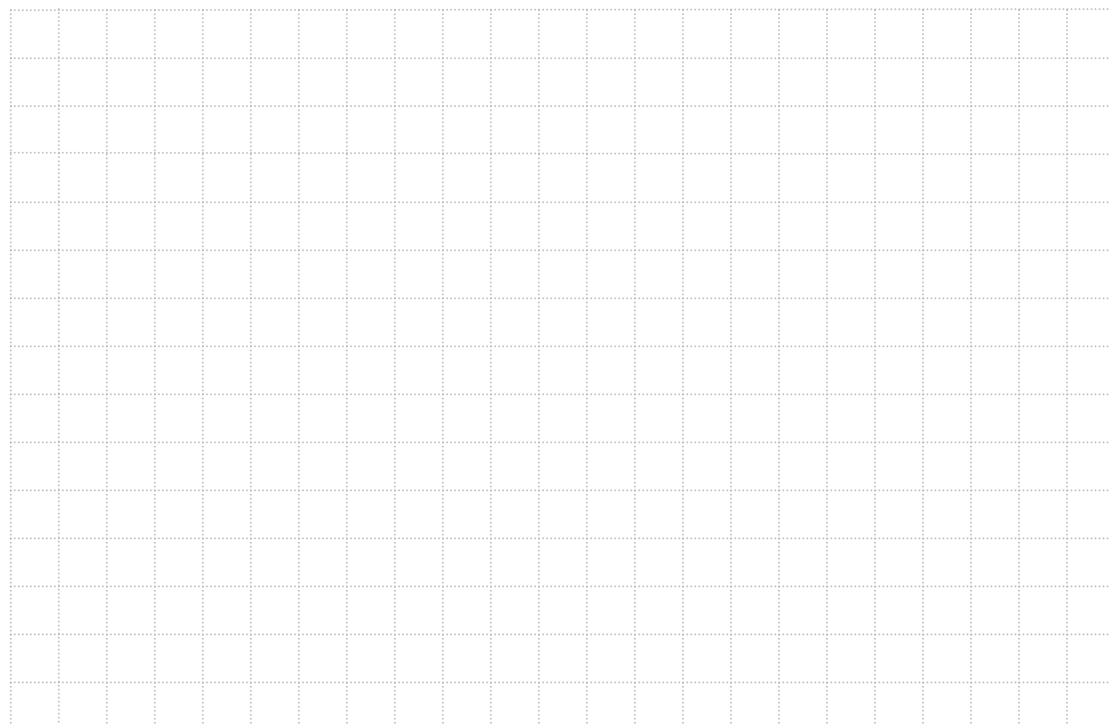
2o mensaje

## SWRITE (String)

Sintaxis de la instrucción SWRITE:

*SWRITE (String, Status, OFFSET, Format, VALUE)*

Argumento	Tipo	Explicación
<i>String</i>	CHAR [ ]	El string manipulado se escribe en esta cadena de caracteres.

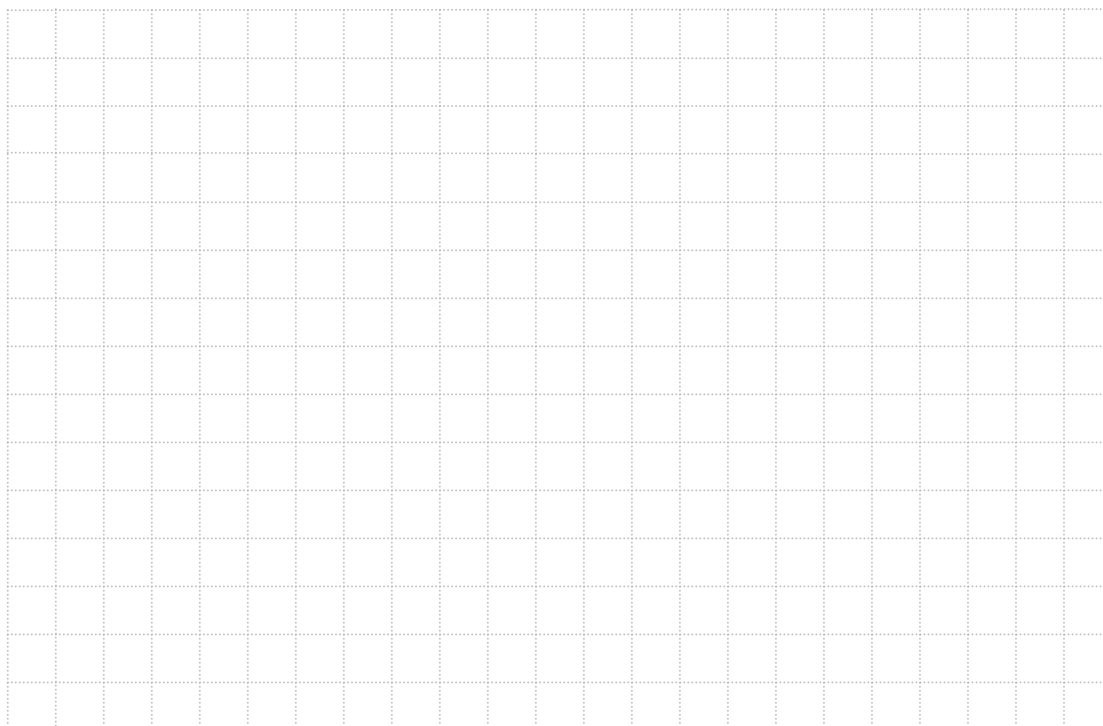


## SWRITE (Status)

Sintaxis de la instrucción SWRITE:

*SWRITE (String1, **Status**, OFFSET, Format, VALUE)*

Argumento	Tipo	Explicación
<b>Status</b>	STATE_T	<ul style="list-style-type: none"><li>•Esta estructura devuelve información sobre el estado del kernel del sistema, que puede evaluar el usuario.</li><li>•<b>STATE.MSG_NO</b>: si ocurre un error durante la ejecución de la instrucción, esta variable contiene el número del error.</li><li>•<b>CMD_OK</b>: Instrucción ejecutada con éxito.</li><li>•<b>CMD_ABORT</b>: Instrucción no ejecutada con éxito.</li><li>•Variable State: HITS número de formatos escritos correctamente.</li></ul>



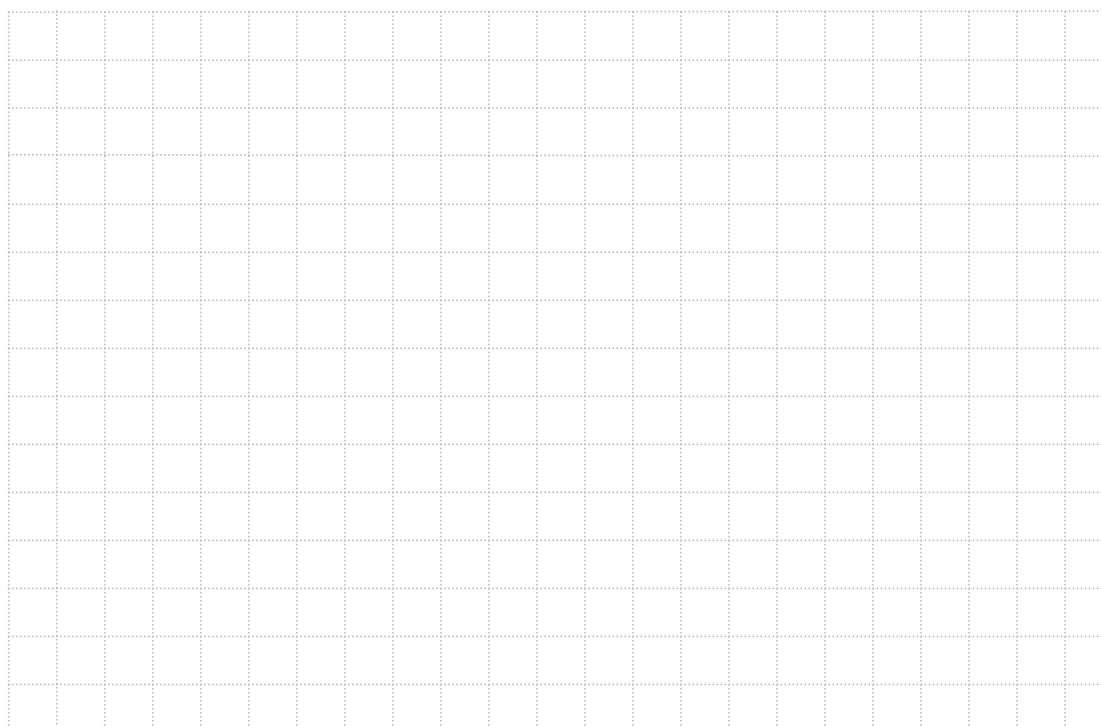


## SWRITE (OFFSET, Format, VALUE)

Sintaxis de la instrucción SWRITE:

*SWRITE (String1, Status, **OFFSET**, **Format**, **VALUE**)*

Argumento	Tipo	Explicación
<b>OFFSET</b>	INT	Especifica la posición desde la que String2 se debe copiar en String1.
<b>Format</b>	CHAR [ ]	La variable "Format" contiene el formato del texto que debe ser generado. Puede observarse en la tabla siguiente.
<b>VALUE</b>	INT REAL BOOL CHAR [ ]	El contenido de esta variable se copia en String2 con el formato especificado. Los valores Booleanos se muestran como 0 o 1, los valores ENUM como números.

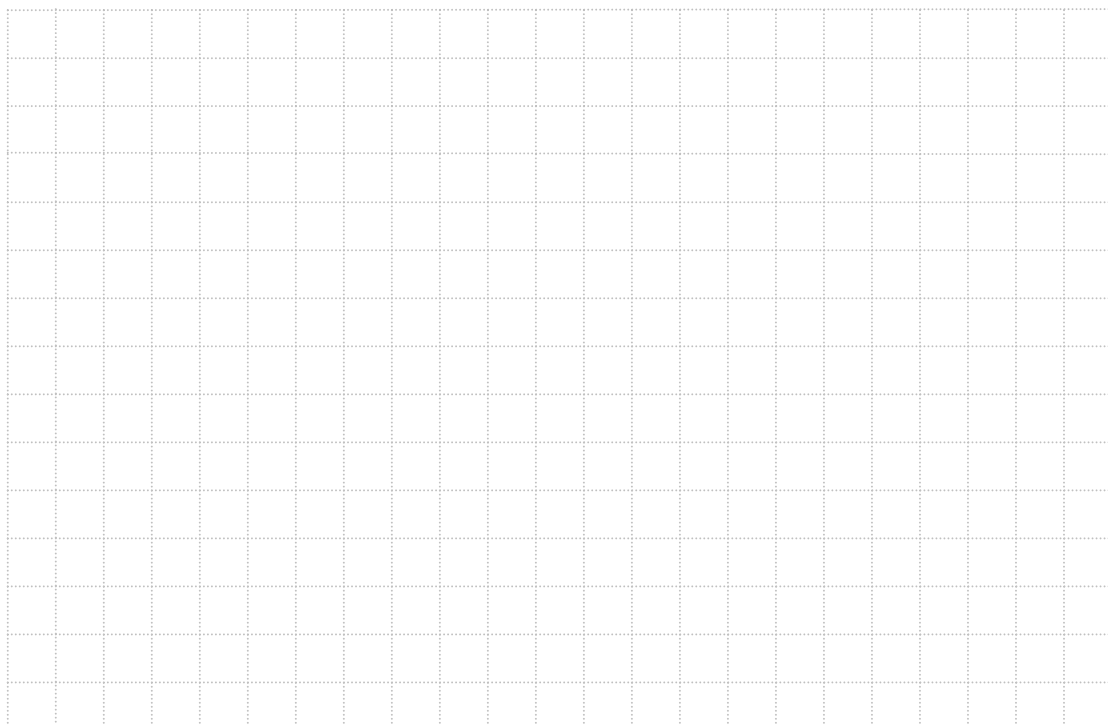


## SWRITE (table de formato)

La conversión de caracteres permitida para la cadena **Format array** es:  
c, d, f, s y %.

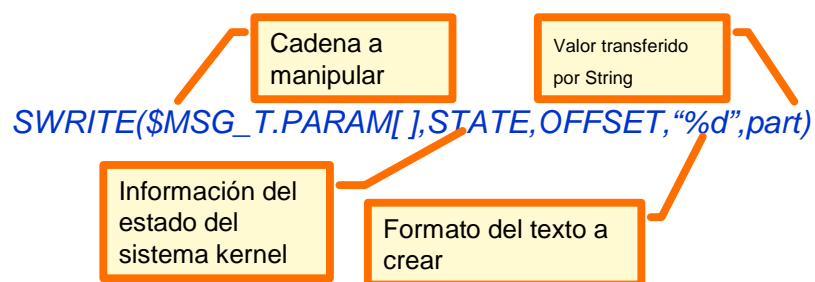
Nuestro ejemplo:  
INT variable → %d

Permissible data type	%d	%f	%c	%s
(SIGNAL) INT	X	X	-	-
INT array	-	-	-	-
REAL	-	X	-	-
REAL array	-	-	-	-
(signal) BOOL	X	-	-	-
BOOL array	-	-	-	-
ENUM	X	-	-	-
ENUM array	-	-	-	-
CHAR	X	-	X	-
CHAR array	-	-	-	X



Ejemplo de SWRITE con PARAM[ ]

SWRITE de nuestro programa de ejemplo



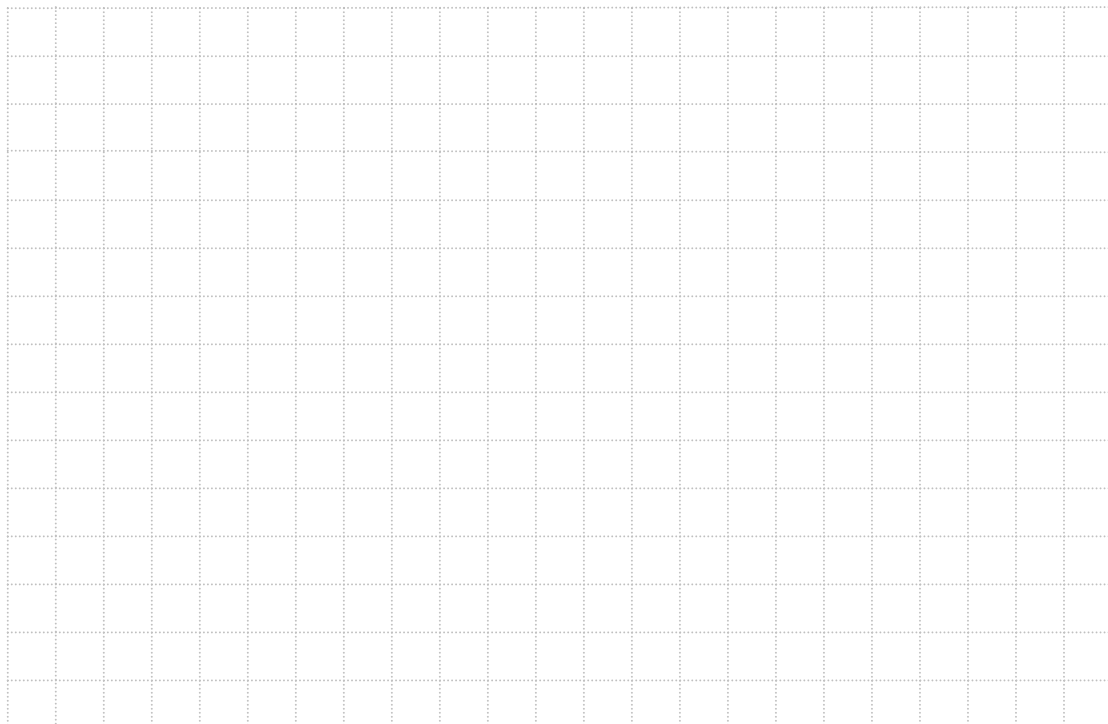
## Ejemplo de SWRITE con PARAM[ ]

Ejemplo de programa con todas las declaraciones requeridas y la inicialización de OFFSET

```
INT part
DECL INT OFFSET
DECL STATE_T STATE
...
part = part +1
$MSG_T.KEY[ ]="Part no. %1 is ready"
OFFSET=0
SWRITE($MSG_T.PARAM[ ],STATE,OFFSET,"%d",part)
; Generar mensaje
...
```

Part no. 1 is ready

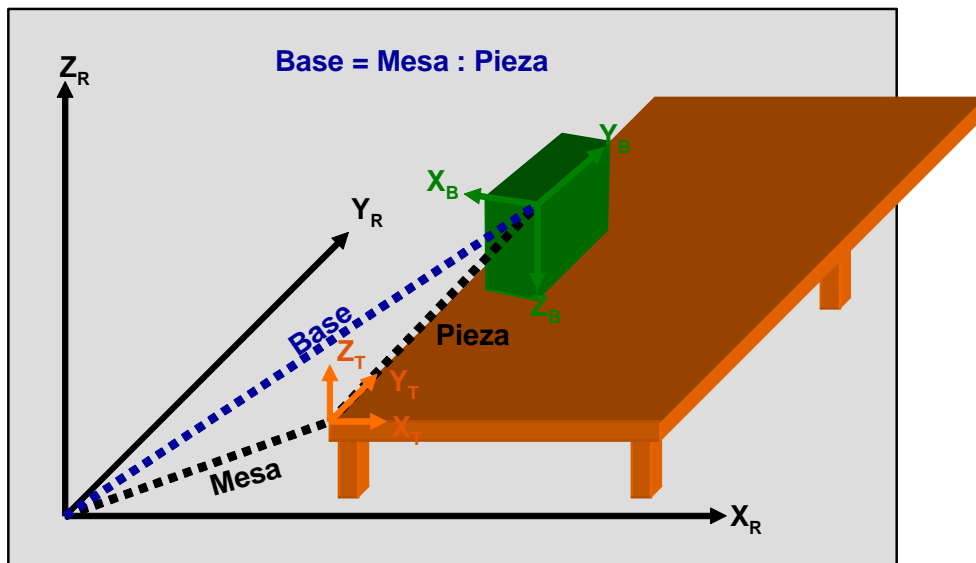
1er mensaje



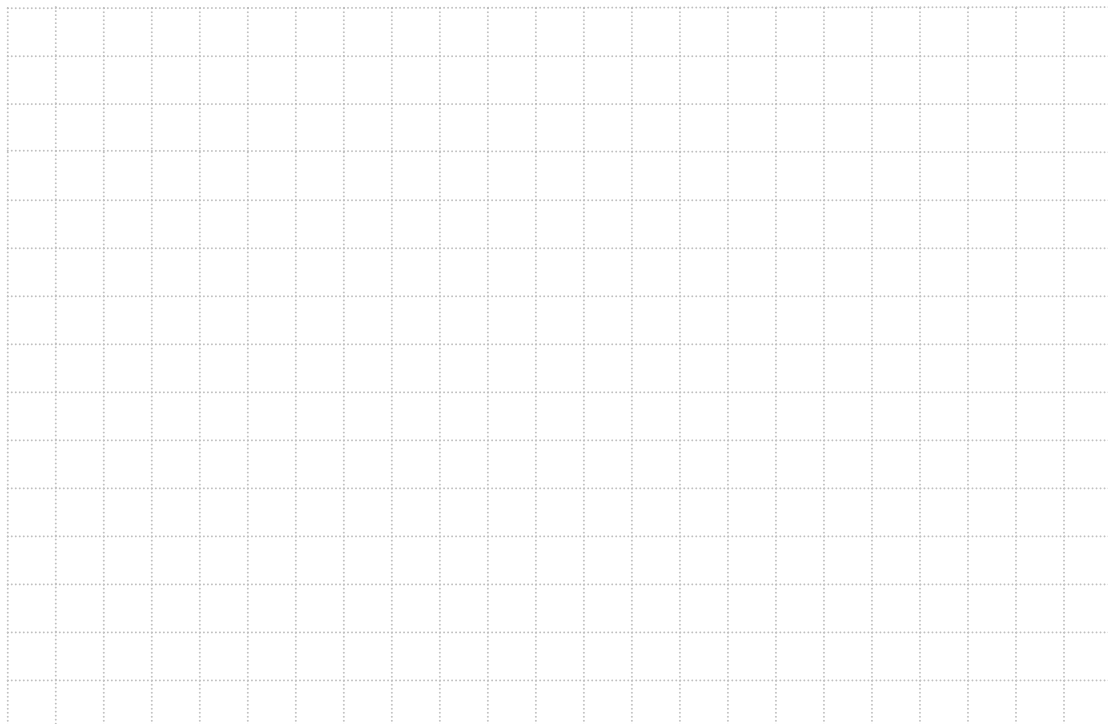
## **4. Operador geométrico**

## Operador geométrico

El operador geométrico realiza un acoplamiento lógico (operación lógica) entre los operandos tipo FRAME y POS.



01/2006 1  
geometric\_operator\_en.ppt  
© Copyright by KUKA Roboter GmbH College



## Combinación lógica de frames

Combinaciones de tipos de datos en el operador geométrico

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



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

## Ejemplo: Retorno en la dirección de la tool



### Prerrequisitos:

La dirección de la tool tiene que ser medida a lo largo del eje X. También es importante que esta tool esté activa.

### Tarea:

Retorno desde la mesa (100 mm), independientemente donde se encuentre el robot.

### Solución:

`LIN $POS_ACT : {x -100,y 0,z 0,a 0,b 0,c 0}`

Nota: \$POS\_ACT (variable de sistema la cual contiene la actual posición del robot)



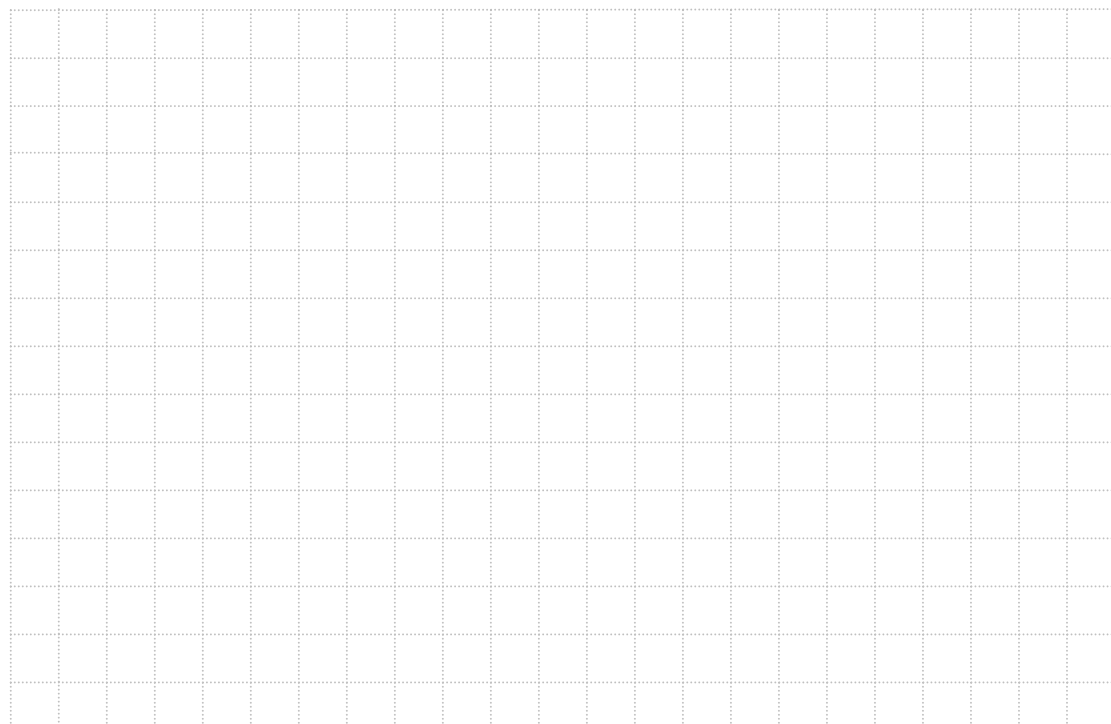
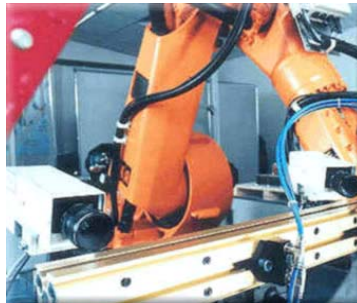


## **5. Programación de interrupciones**

### **5.1 Teoría y KRL-sintaxis para programación de interrupciones**

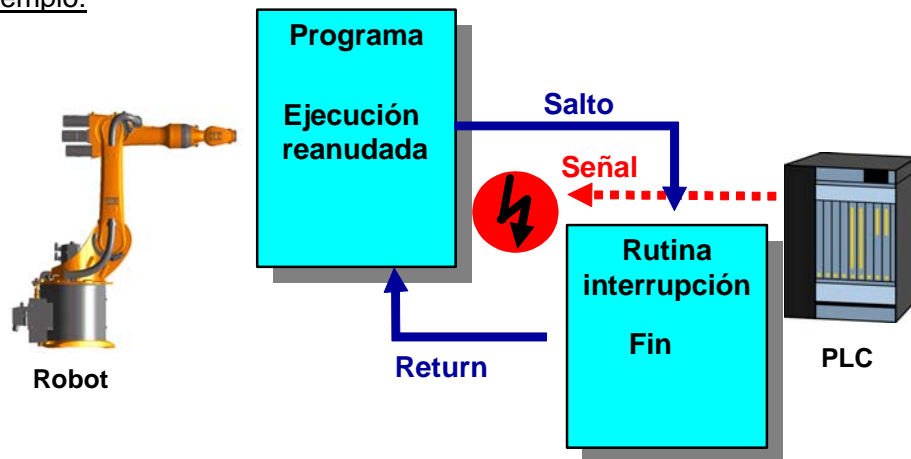
## Interrupciones - Descripción

Las rutinas de interrupción se utilizan para hacer reaccionar al robot inmediatamente ante un evento interno o externo, p. ej. Un programa en ejecución puede ser interrumpido e iniciarse un subprograma o función. Los movimientos del robot y las rutinas de interrupción también pueden funcionar en paralelo.



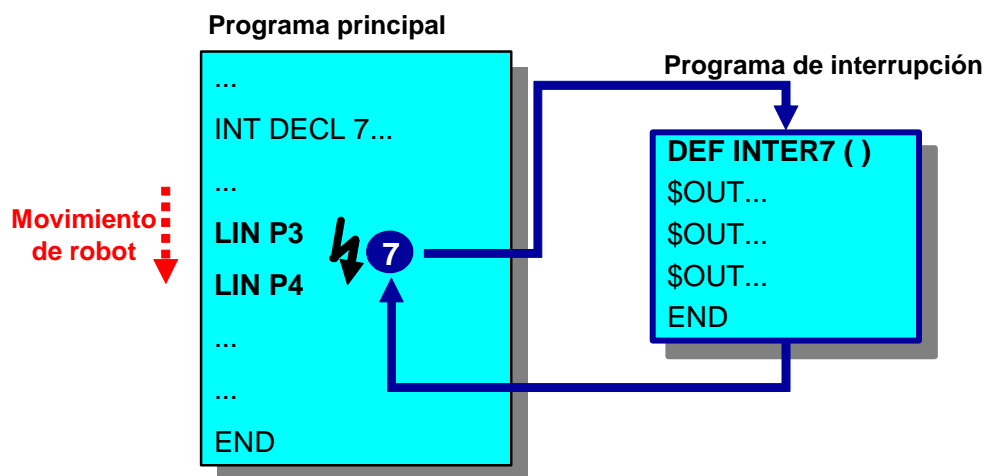
Interrupción - Ejemplo

Ejemplo:



## Interrupción – Procesamiento paralelo de programa principal e interrupción

La rutina de interrupción se ejecuta de forma paralela a los movimientos del robot (P3 a P4)



## Interrupción - Sintaxis

### Declaración:

```
INTERRUPT DECL Prioridad WHEN Evento DO Subprograma
```

- Un máximo de 32 interrupciones pueden ser declaradas simultaneamente
- Un máximo de 16 interrupciones pueden ser activadas simultaneamente
- Nivel de prioridad 1-39 y 81-128
- El nivel 1 tiene la máxima prioridad
- Este evento es detectado por medio de un impulso cuando tiene lugar (edge-triggered)
- Las interrupciones se tienen que declarar en la sección de instrucciones

```
DEF Main ( )  
INI  
INTERRUPT DECL 20 WHEN $IN[22]==TRUE DO SP1( )  
...  
END  
...
```

## Activando y desactivando interrupciones

**INTERRUPT ON** *Número*

Activar una interrupción

**INTERRUPT OFF** *Número*

Desactivar una interrupción

Si la instrucción 'INTERRUPT ON' no contiene número de interrupción (prioridad), todas las interrupciones declaradas serán activadas.

INI

INTERRUPT DECL 20 WHEN \$IN[22]==TRUE DO SP1( )

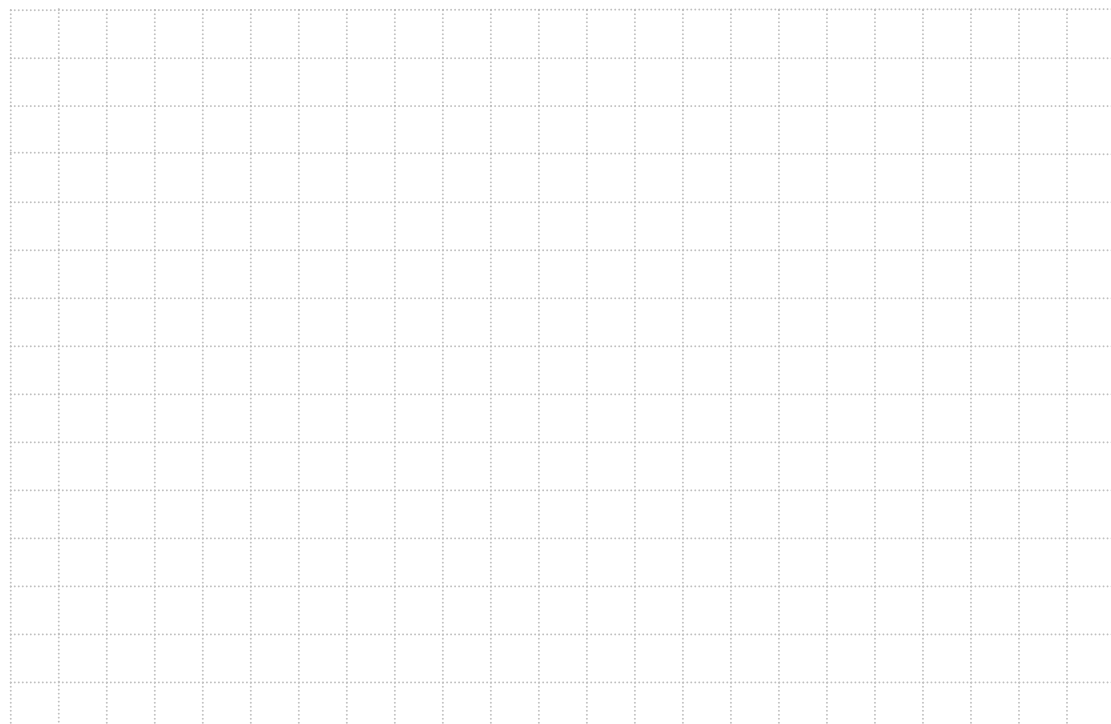
...

**INTERRUPT ON 20**

La interrupción es reconocida y ejecutada.

**INTERRUPT OFF 20**

END



## Activando y desactivando interrupciones

**INTERRUPT ON** *Número*

Activar interrupción

**INTERRUPT OFF** *Número*

Desactivar interrupción



Si existe el riesgo de que una interrupción sea lanzada incorrectamente varias veces debido a sensores (**“rebote de contactos”**), puede prevenirse desactivando la interrupción en la primera línea del programa de interrupción. De todos modos, una interrupción desactivada durante el proceso de interrupción no puede volver a ser reconocida. Si la interrupción debe permanecer activa, debe volver a activarse antes de volver al programa principal.



## Habilitar y deshabilitar interrupciones

**INTERRUPT DISABLE** *Número*

Deshabilitar interrupción

**INTERRUPT ENABLE** *Número*

Habilitar interrupción

Si las instrucciones no contienen número de interrupción (prioridades), todas las interrupciones son tenidas en cuenta.

INI

...

INTERRUPT ON 20

...

**INTERRUPT DISABLE 20**

La interrupción es reconocida y guardada.  
No se ejecutará hasta que no se habilite.

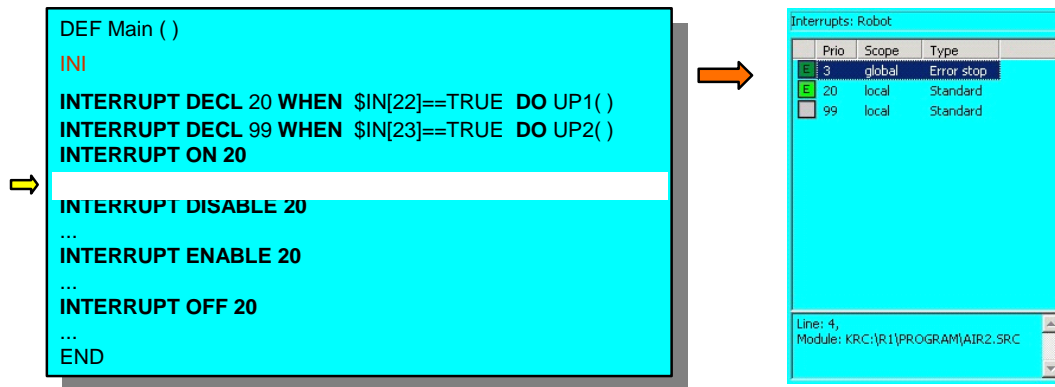
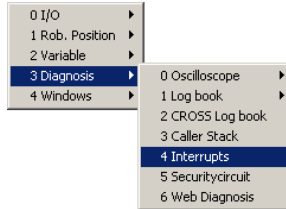
**INTERRUPT ENABLE 20**

END



## Interrupción – Diagnóstico

**Llamada al diagnóstico de interrupciones**



**La doble declaración es posible. Prestar atención, pues ya existen interrupciones declaradas en la Fold "INI" (p.ej. INTERRUPT 3).**

## Interrupción - Diagnóstico

Interrupts: Robot		
Prio	Scope	Type
3	global	Error stop
20	local	Standard
99	local	Standard

Line: 4,  
Module: KRC:\R1\PROGRAM\AIR2\BRC

Tipo de interrupción\*

Interrupción local o global

Prioridad

Estado de la interrupción\*\*

Nombre del módulo y línea de programa de la declaración de la interrupción

### \*Tipo de interrupción

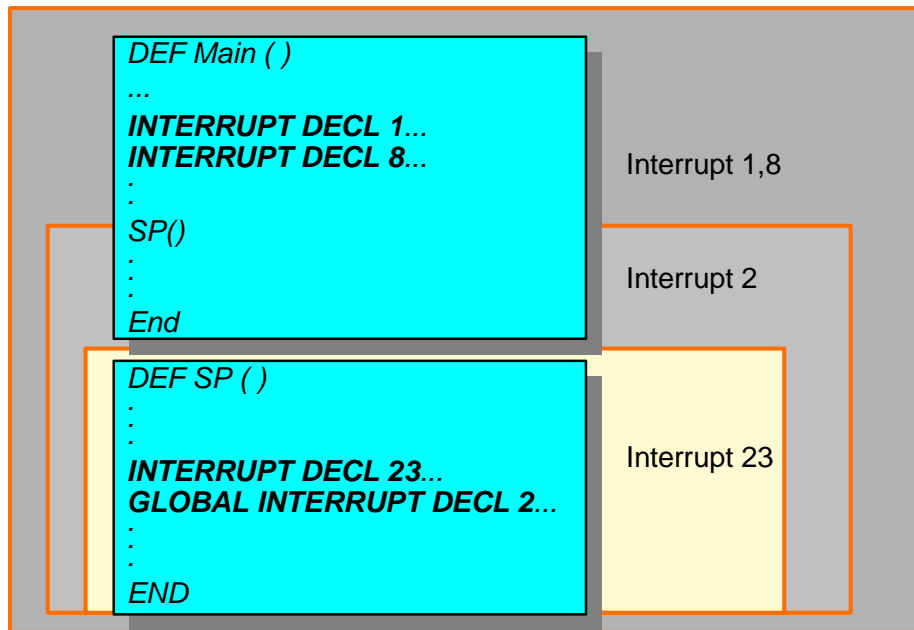
*El tipo de interrupción depende del evento definido en la declaración de la interrupción:*

\$STOPMESS	? Parada por error
\$ALARM_STOP	? Parada de Emergencia
\$MEAS_PULSE[1..5]	? Medición (Medición rápida)
e.g. \$IN[1...4096]	? Standard
Trigger-subprogram	? Trigger

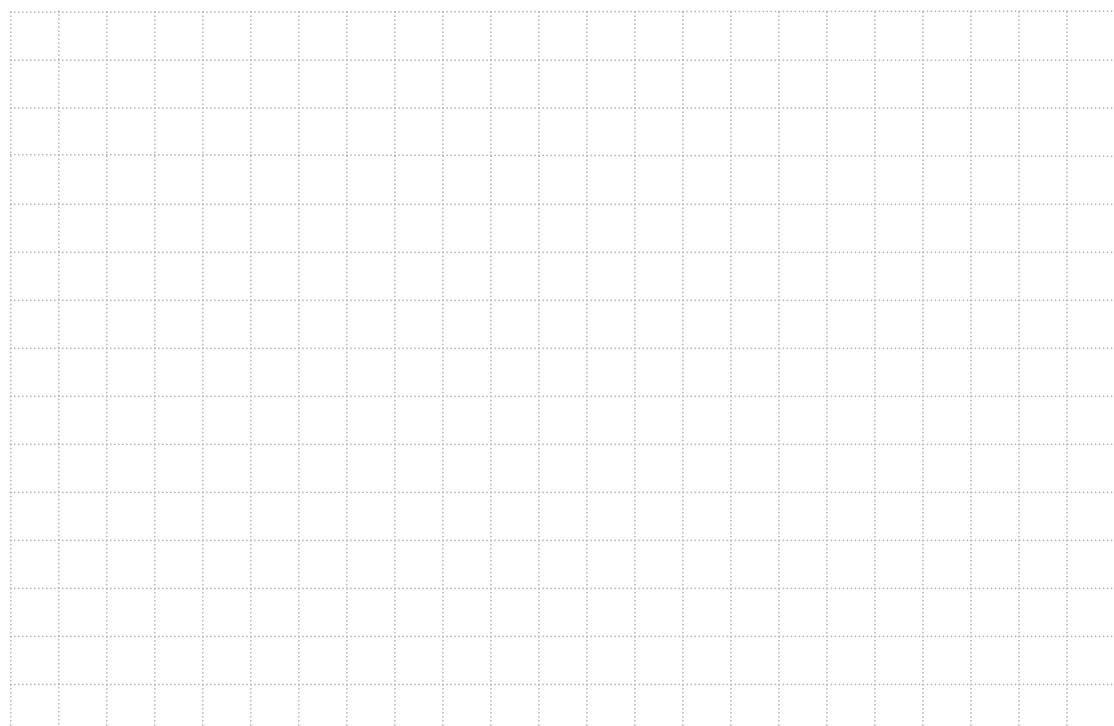
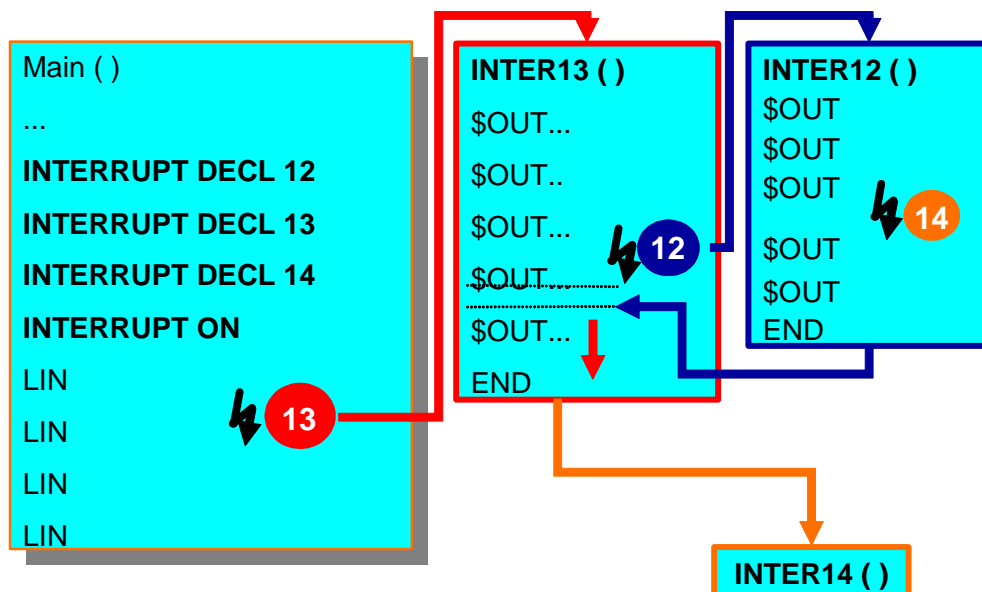
### \*\*Estado interrupción

E	Interrupción activada – INTERRUPT ON/ENABLE
D	Interrupción desactivada – INTERRUPT DISABLE
	Interrupción inactiva – INTERRUPT OFF o no activada

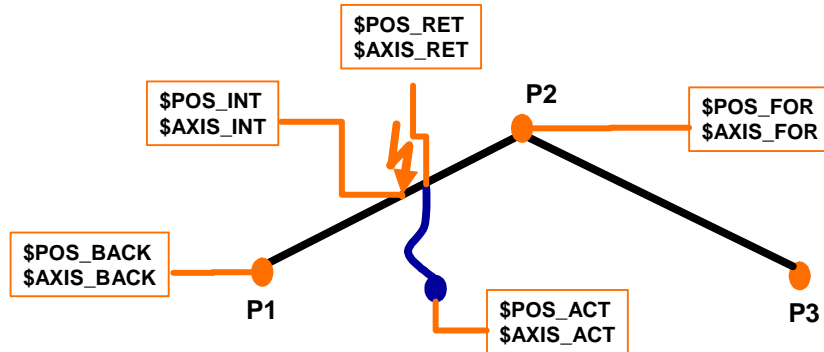
## Interrupción – Rango de acción



Interrupción - Prioridad

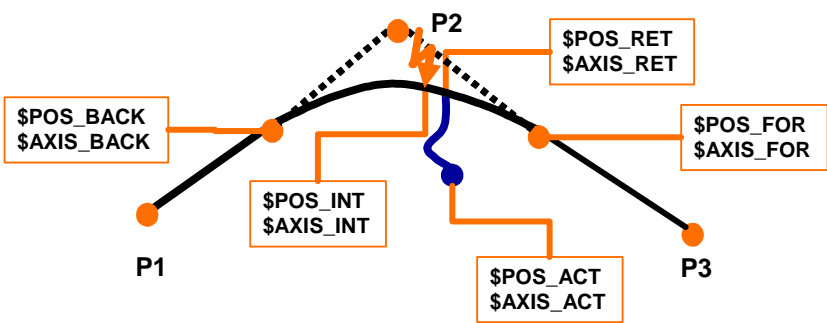


## Interrupción – Variables de sistema útiles (Posicionamiento exacto)

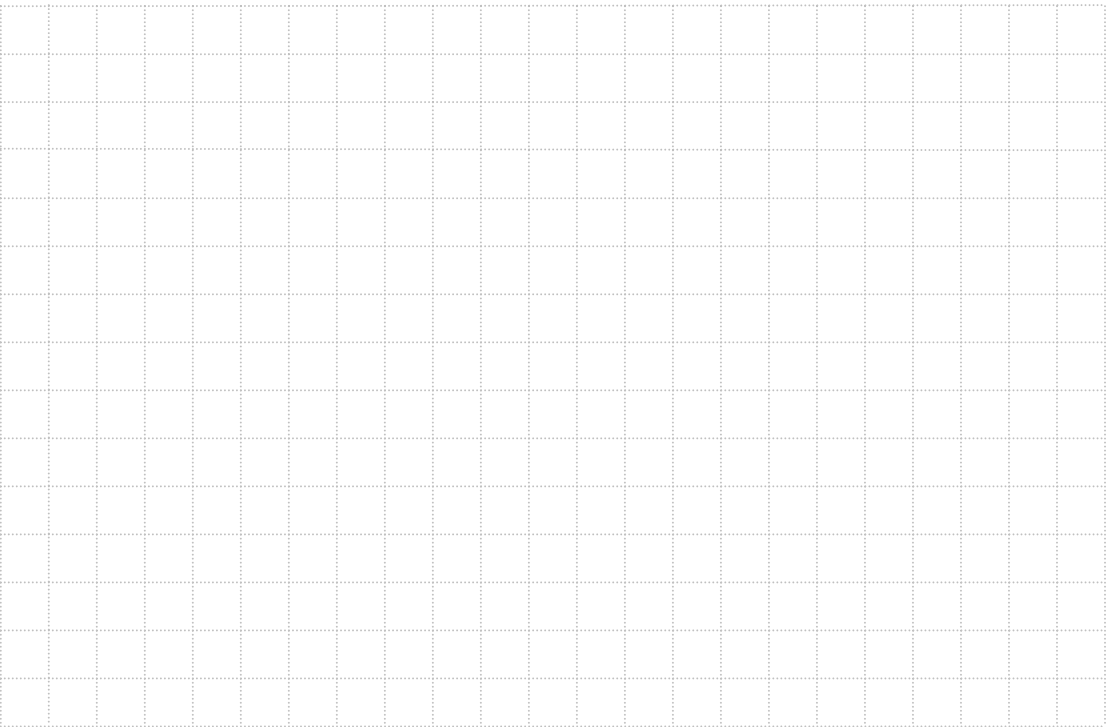


Específico-ejes	Cartesiano	Descripción
<b>\$AXIS_INT</b>	<b>\$POS_INT</b>	Posición en la que se lanzó la interrupción
<b>\$AXIS_ACT</b>	<b>\$POS_ACT</b>	Posición actual
<b>\$AXIS_RET</b>	<b>\$POS_RET</b>	Posición en que el robot dejó la trayectoria
<b>\$AXIS_BACK</b>	<b>\$POS_BACK</b>	Posición del punto de inicio de la trayectoria
<b>\$AXIS_FOR</b>	<b>\$POS_FOR</b>	Posición del punto final de la trayectoria

Interrupción – Variables de sistema útiles (posición aproximada)



Específico-ejes	Cartesiano	Descripción
<b>\$AXIS_INT</b>	<b>\$POS_INT</b>	Posición en que se lanzó la interrupción
<b>\$AXIS_ACT</b>	<b>\$POS_ACT</b>	Posición actual
<b>\$AXIS_RET</b>	<b>\$POS_RET</b>	Posición en que el robot dejó la trayectoria
<b>\$AXIS_BACK</b>	<b>\$POS_BACK</b>	Posición del punto de inicio de la trayectoria
<b>\$AXIS_FOR</b>	<b>\$POS_FOR</b>	Posición del punto final de la trayectoria

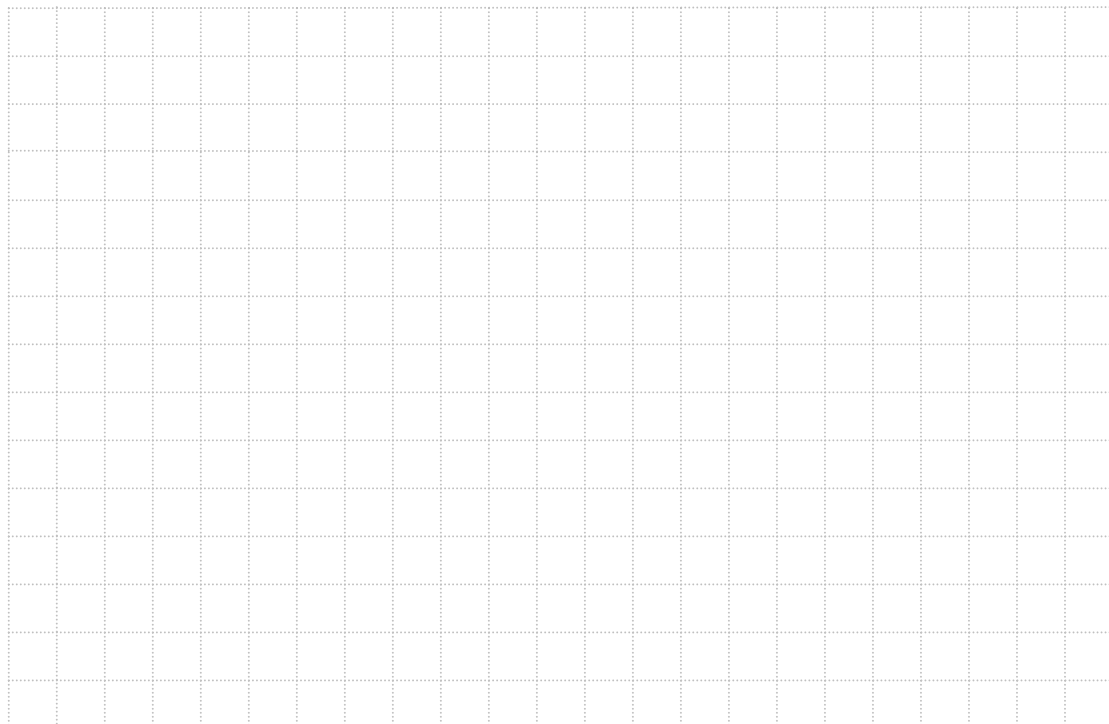


## Uso de banderas cíclicas (cyclical flags)

Ninguna operación lógica está permitida en la instrucción de declaración de la interrupción. De hecho, para poder definir eventos complejos, se debe trabajar con banderas cíclicas (cyclical flags), pues es el modo de posibilitar la actualización constante.

```
DEF Main ( )  
...  
$CYCFLAG[3] = $IN[12] AND ($IN[34] OR $IN[3])  
INTERRUPT DECL 10 WHEN $CYCFLAG[3] DO IR_PROG ( )  
INTERRUPT ON  
...  
END
```

Con esta secuencia de programa se pueden controlar y combinar simultaneamente 3 entradas.



## **5.2 Cancelar movimiento con interrupciones**



## Interrupción – instrucción BRAKE

### **BRAKE**

Frena y detiene el movimiento del robot

### **BRAKE F**

Frena y detiene el movimiento del robot con los máximos valores

La instrucción **BRAKE** frena el movimiento del robot. El movimiento del robot es reanudado tan pronto como se haya completado la rutina de interrupción.

```
DEF INT4 ( )
```

```
...
```

```
BRAKE F
```

```
$OUT...
```

```
END
```



La instrucción **BRAKE** solo puede ser utilizada en un programa de interrupción

## Interrupción – Cancelar rutinas de interrupción

### RESUME

- Cancela programas de interrupción y subprogramas hasta el nivel en que fue declarada la interrupción activa.
- Esto puede significar terminar movimientos del robot.



La instrucción RESUME solo puede ser ejecutada en un programa de interrupción

Cuando se activa la sentencia RESUME, el puntero de avance de programa no debe estar en el nivel donde se declaró la interrupción, al menos un nivel por debajo.

## Condiciones para cancelar una rutina de interrupción



Las siguientes condiciones deben cumplirse para poder cancelar un movimiento:

1. Combinación de 'Brake / Resume' en el programa de interrupción.
2. El movimiento debe estar al menos un nivel por debajo de la declaración de la interrupción.
3. El puntero de avance de programa no debe estar aún en el nivel en que se declaró la interrupción.



## Interrupción – Ejemplo: Cancelar rutinas de interrupción

Ejemplo:

```
DEF Prog1 ( )
INI
...
INTERRUPT DECL 21 WHEN $IN[1] DO Found ( )
...
PTP HOME
...
SEARCH ( )
$ADVANCE=3
...
END
```

**RESUME** cancela inmediatamente  
ambos subprogramas y salta  
al programa principal

```
DEF Search ( )
INTERRUPT ON 21
LIN Strpt
LIN Endpt
$ADVANCE = 0
...
END; of search
```

```
DEF Found ( )
INTERRUPT OFF 21
BRAKE
LIN $POS_INT
;pick up part
...
RESUME; cancel
END
```

## **6. Funciones de conmutación relativas a la trayectoria (Trigger)**

## Trigger – Acciones de disparo dependientes de la trayectoria

A diferencia de las interrupciones, que son independientes del movimiento del robot, algunas aplicaciones también requieren acciones de disparo dependientes de la trayectoria de movimiento. Aplicaciones típicas son:

- Disparo de la corriente on/off en procesos de soldadura al arco.
- Cerrar o abrir la pistola de soldadura en procesos de soldadura por resistencia.
- Arranque o paro del flujo de adhesivo en procesos de pegado o sellado.



## Funciones de disparo al inicio o el final de la trayectoria

### Sintaxis:

**TRIGGER WHEN DISTANCE = *Punto de disparo* DELAY = *Tiempo* DO**  
*Instrucción* <PRIO = *Prioridad*>



- A cada instrucción TRIGGER con una llamada a subprogram se le debe asignar una prioridad. Se permiten valores de 1-39 y de 81-128. Las prioridades son entonces las mismas que las de las interrupciones.
- Los valores de 40-80 están reservados para la asignación automática de prioridad por parte del sistema. Para esta asignación se escribe PRIO=--1.
- Se pueden usar hasta 8 Trigger al mismo tiempo.
- El tiempo máximo de disparo es > 1.000.000 (ms)

### Ejemplo:

```
...  
LIN PUNTO2  
...  
TRIGGER WHEN DISTANCE = 0 DELAY = 20 DO $OUT[4] = TRUE  
TRIGGER WHEN DISTANCE = 1 DELAY = -25 DO UP1() PRIO=-1  
LIN PUNTO3  
...  
LIN PUNTO4  
...
```

## Punto de inicio y final con posicionamiento exacta

### Ejemplo:

...

LIN P2

...

TRIGGER WHEN DISTANCE = 0 DELAY = 20 DO \$OUT[4] = TRUE

TRIGGER WHEN DISTANCE = 1 DELAY = -25 DO SP1() PRIO=-1

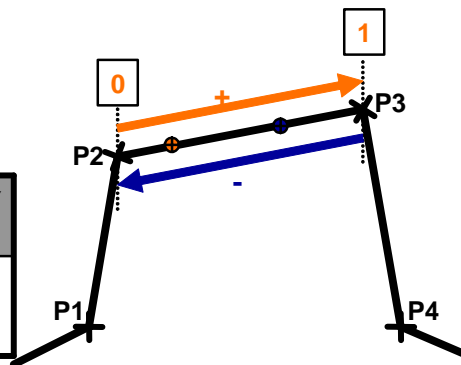
LIN P3

...

LIN P4

...

DISTANC E	Rango de disparo	DELAY
0	0 - 1	+
1	1 - 0	-

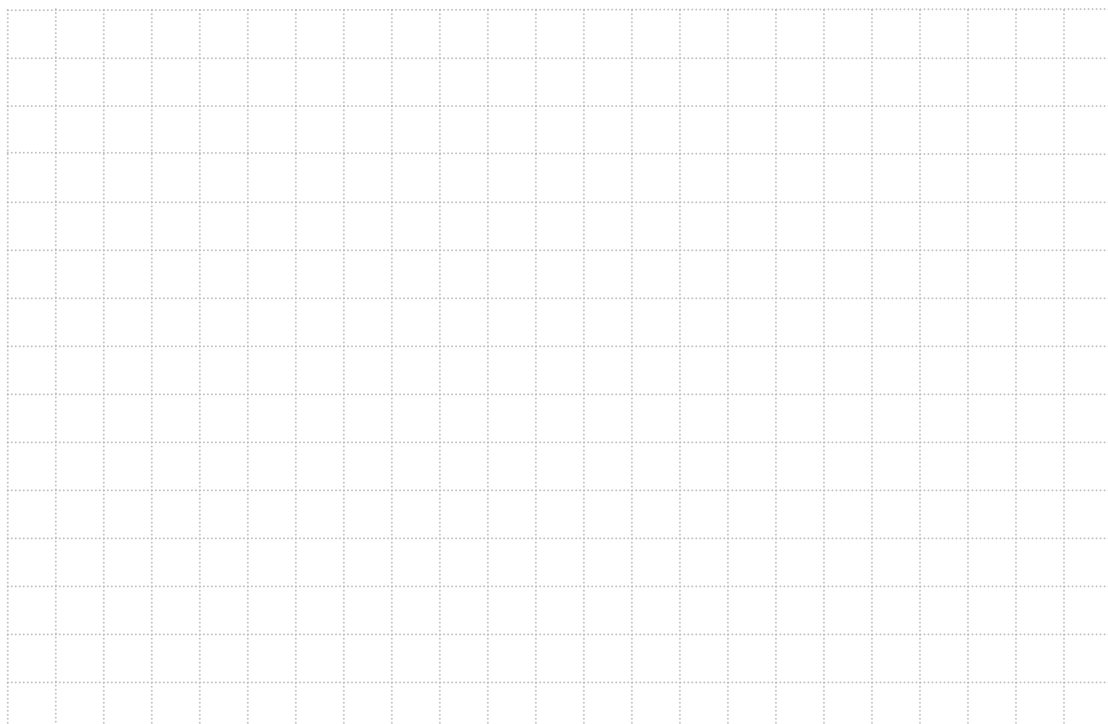


01/2006

3

trigger\_en.ppt

© Copyright by KUKA Roboter GmbH College





## Punto de inicio con parada exacta, punto final con posicionamiento aproximado

### Ejemplo:

...  
LIN P2

...  
TRIGGER WHEN DISTANCE = 0 DELAY = 20 DO \$OUT[4] = TRUE

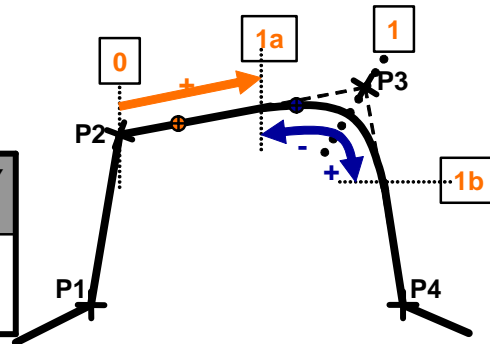
TRIGGER WHEN DISTANCE = 1 DELAY = -25 DO SP1() PRIO=-1

LIN P3 **C\_DIS**

...  
LIN P4

...

DISTANC E	Rango de disparo	DELAY
0	0 - 1a	+
1	1a - 1b	+/-



01/2006

4

trigger\_en.ppt

© Copyright by KUKA Roboter GmbH College



### Punto de inicio y final con posicionamiento aproximado

**Ejemplo:**

...

LIN P2 **C\_DIS**

...

```
...
    TRIGGER WHEN DISTANCE = 0 DELAY = 20 DO $OUT[4] = TRUE
```

```
TRIGGER WHEN DISTANCE = 1 DELAY = -25 DO SP1() PRIO=-1
```

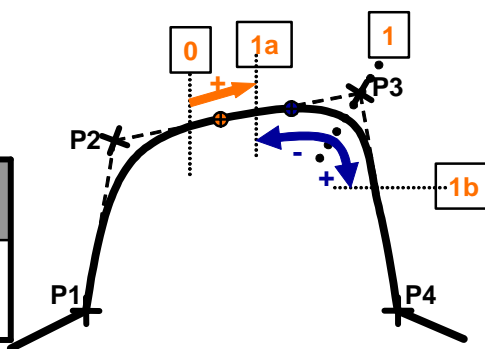
LIN P3 **C\_DIS**

■ ■ ■

LIN P4

...

DISTANC E	Rango de disparo	DELAY
0	0 - 1a	+
1	1a - 1b	+/-

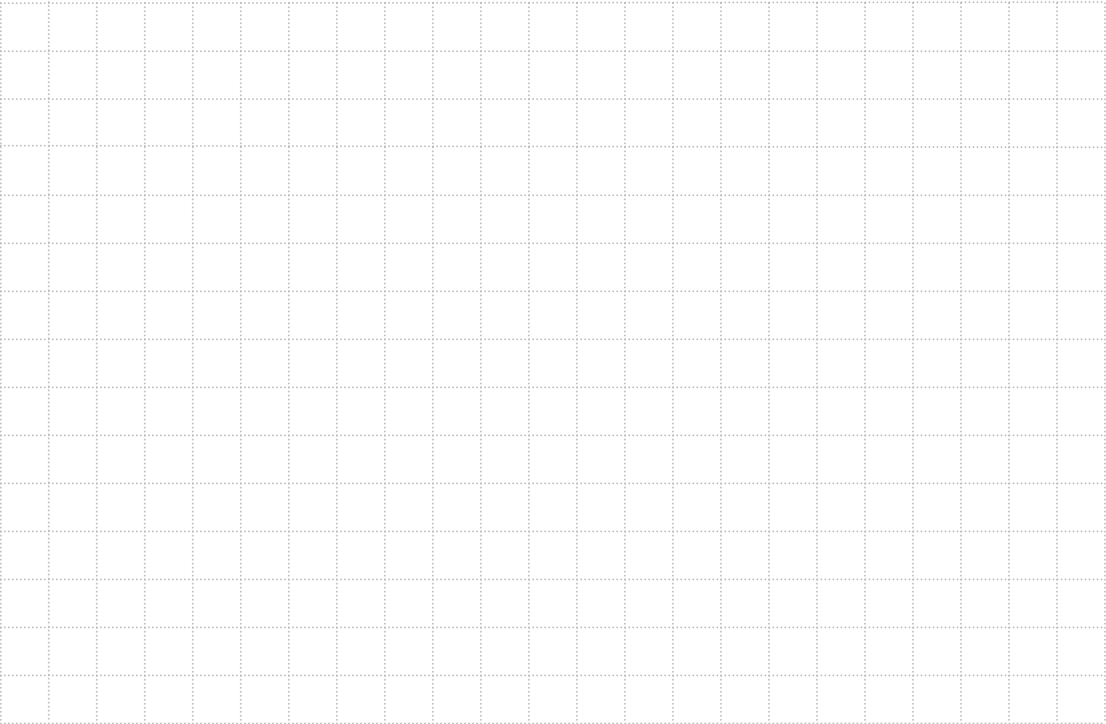
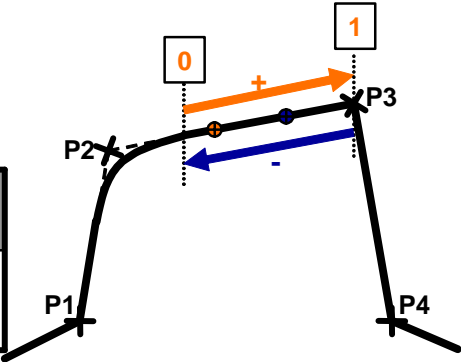


Punto de inicio con posicionamiento aproximado, punto final con parada exacta

Ejemplo:

```
...
LIN P2 C_DIS
...
TRIGGER WHEN DISTANCE = 0 DELAY = 20 DO $OUT[4] = TRUE
TRIGGER WHEN DISTANCE = 1 DELAY = -25 DO SP1() PRIO=-1
LIN P3
...
LIN P4
...
```

DISTANC E	Rango de disparo	DELAY
0	0 - 1	+
1	1 - 0	-



## Funciones de disparo en cualquier punto de la trayectoria

### Sintaxis:

**TRIGGER WHEN PATH = *Distance* DELAY = *Time* DO *Instrucción***  
**<PRIO = *Prioridad*>**



- A cada instrucción TRIGGER con una llamada a subprogram se le debe asignar una prioridad. Se permiten valores de 1-39 y de 81-128. Las prioridades son entonces las mismas que las de las interrupciones.
- Los valores de 40-80 están reservados para la asignación automática de prioridad por parte del sistema. Para esta asignación se escribe PRIO=--1.
- Se pueden usar hasta 8 Trigger al mismo tiempo.
- El tiempo máximo de disparo es > 1.000.000 (ms)

### Ejemplo:

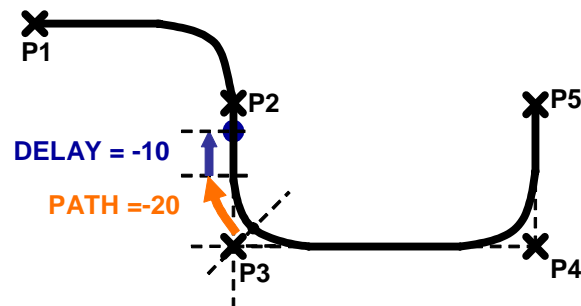
```
...  
LIN P2 C_DIS  
TRIGGER WHEN PATH = -20 DELAY = -10 DO $OUT[2]= TRUE  
LIN P3 C_DIS  
LIN P4 C_DIS  
LIN P5  
...
```

**El punto de referencia  
es el punto de  
destino!!**

## Ejemplo: Disparo sobre trayectoria

### Ejemplo:

```
...  
LIN P2 C_DIS  
TRIGGER WHEN PATH = -20 DELAY = -10 DO $OUT[2] = TRUE  
LIN P3 C_DIS  
LIN P4 C_DIS  
LIN P5  
...
```



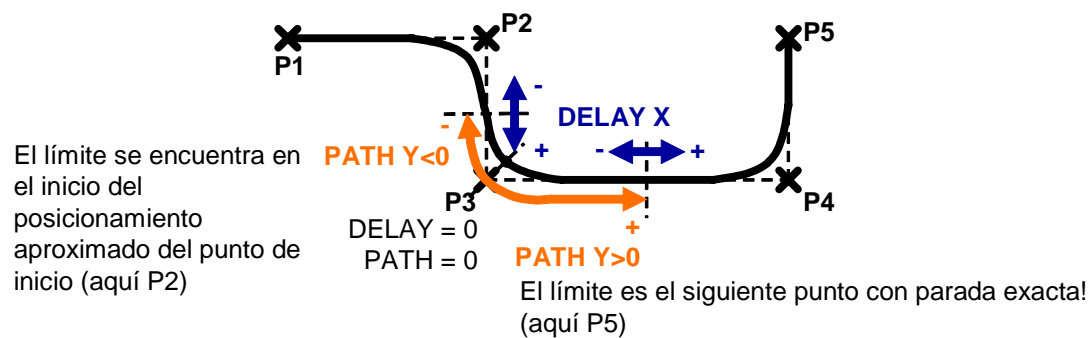
## Ejemplo: Rangos de disparo

### Ejemplo:

```

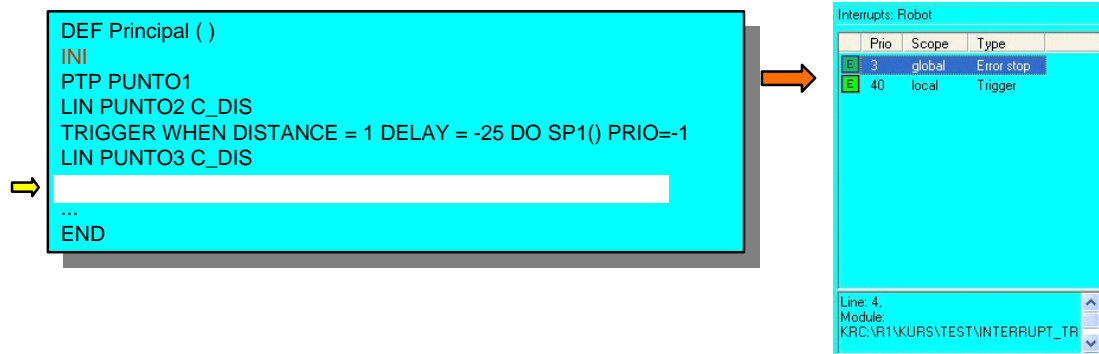
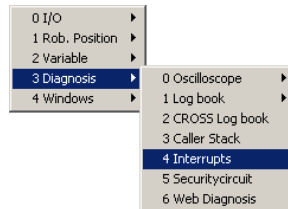
...
LIN P2 C_DIS
TRIGGER WHEN PATH = Y DELAY = X DO $OUT[2] = TRUE
LIN P3 C_DIS
LIN P4 C_DIS
LIN P5
...

```



## Interrupt/Trigger – Diagnosis

**Llamada a la  
diagnosis de  
interrupciones**



**Es posible hacer dobles declaraciones con prioridades. Hay que tener en cuenta que ya existen prioridades de interrupción predefinidas (e.g. INTERRUPT 3).**

## Casos especiales



- **BCO run:** Ejecución del último punto.
- **Aproximación no posible:** Valores positivos de Path se mantienen (ejecución no exacta); valores negativos de Path se ejecutan en el punto.
- **Cancelación del movimiento:** No se ejecutan las acciones de disparo.
- **Instrucciones TRIGGER relativas a la trayectoria para un movimiento PTP:** Se rechazan durante la ejecución.
- **PTP-CP posicionamiento aproximado:** Si el punto inicial es un PTP con posicionamiento aproximado, las acciones de disparo se ejecutarán antes del final del posicionamiento aproximado. Si el punto final es un PTP con posicionamiento aproximado, todas las acciones de disparo que no se hayan ejecutado lo harán en el centro del rango de posicionamiento aproximado.





## **7. Tratamiento automático de defectos**

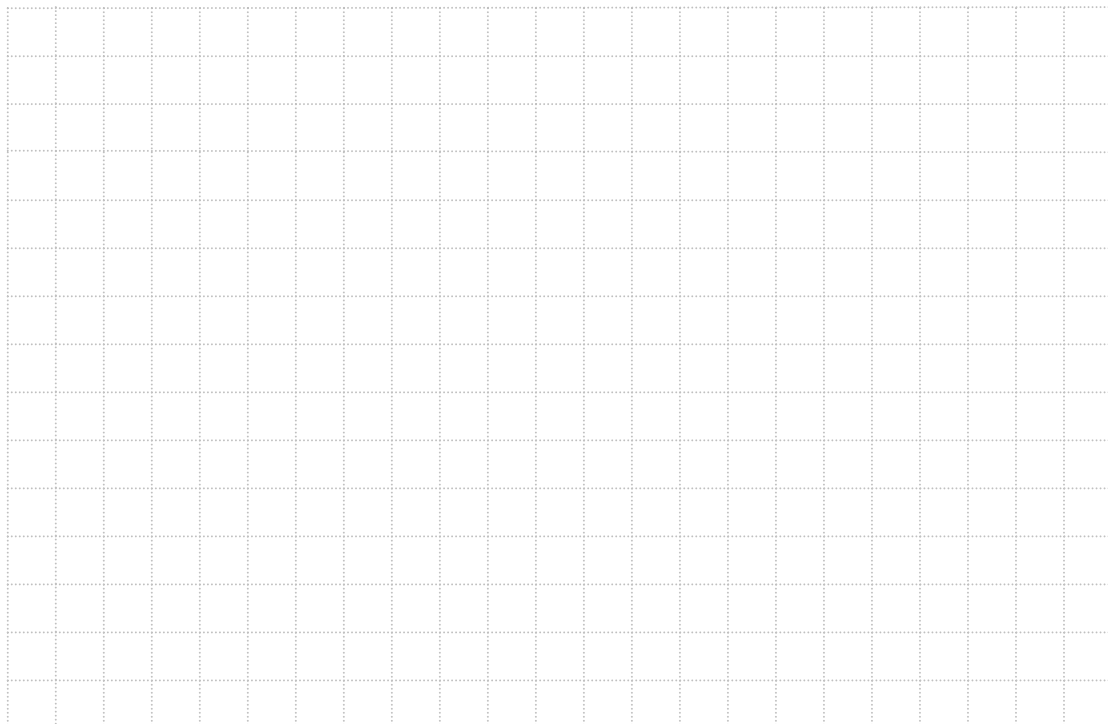
### **7.1 Estrategia ante defectos en general**

## Estrategia ante defectos

Una vez realizado y chequeado la programación de la aplicación en condiciones reales, nos debemos preguntar:

“¿Cómo reaccionará el programa ante un defecto?”

Cada sistema tiene sus requerimientos específicos. One possible approach for machine loading (e.g. machine tools) is to abort the loading process and then start the process again from a defined starting point.



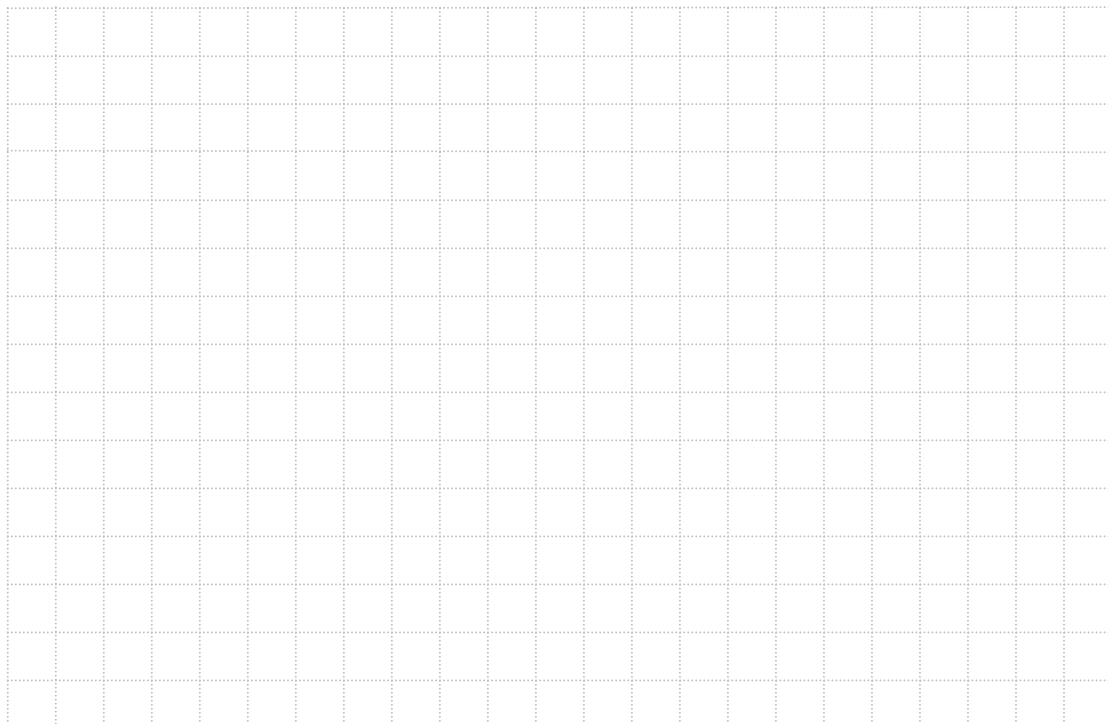
## Estrategia ante defectos

- Ante un **defecto**, el robot no debe continuar el programa en uso, sino se mueve a la posición home, siguiendo una trayectoria dependiendo de la sección particular que se encuentre, y aguarda un nuevo comando para comenzar la tarea.

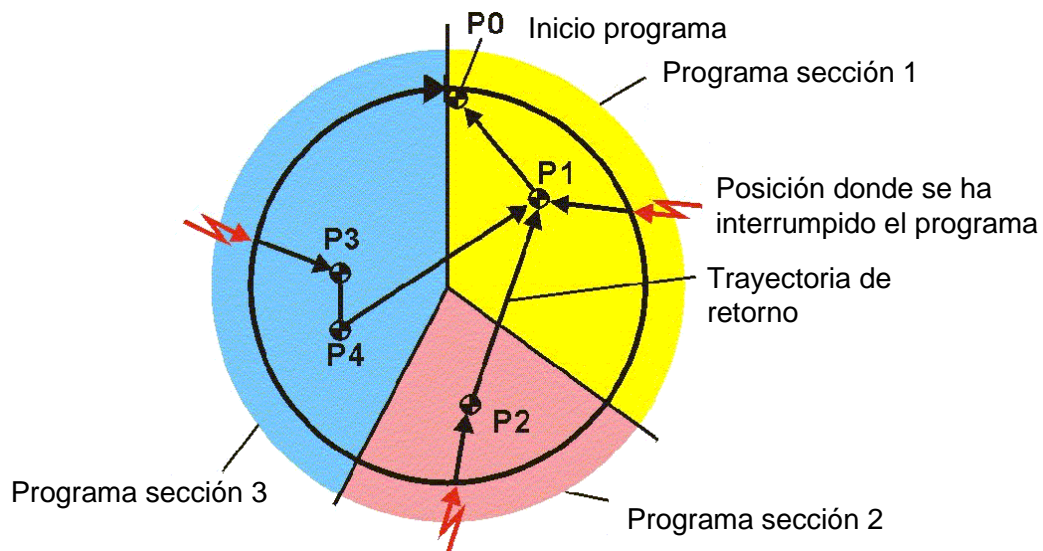
Como **defectos** se pueden incluir:

- Defectos de la periferia enviados por el PLC,
- Defectos directos de la periferia,
- Defectos del KR C .

Los movimientos de retorno deben ser libremente programables.



Estrategia ante defectos



- Puntos globales, se utilizan para los movimientos de retorno, siempre que sea posible.

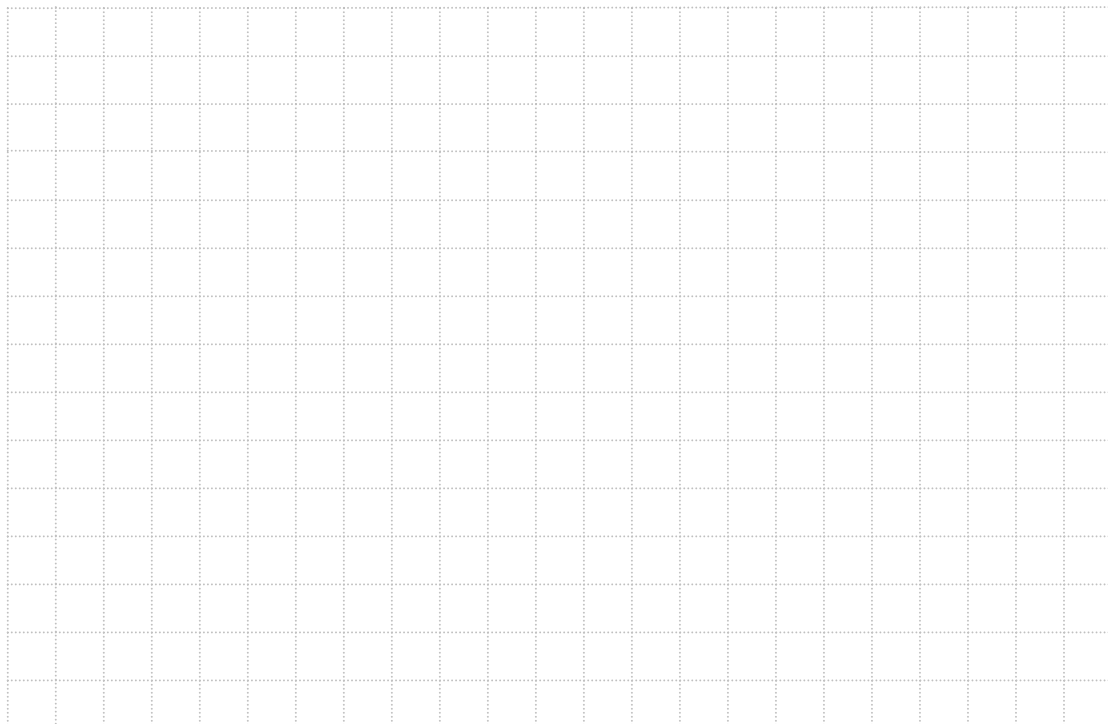
## Estrategia ante defectos – retorno automático o retorno manual por el personal

### ➤ ¿Qué estrategia se debe hacer ante un defecto?

Una vez que se haya eliminado el problema, el robot debe recomenzar o continuar automáticamente el proceso correspondiente. El operario solamente utiliza el PLC para reconocer la avería y para comenzar el proceso.

O

Los operarios deben mover el robot manualmente (modo T1) fuera de área problemática y llevarlo a posición home.



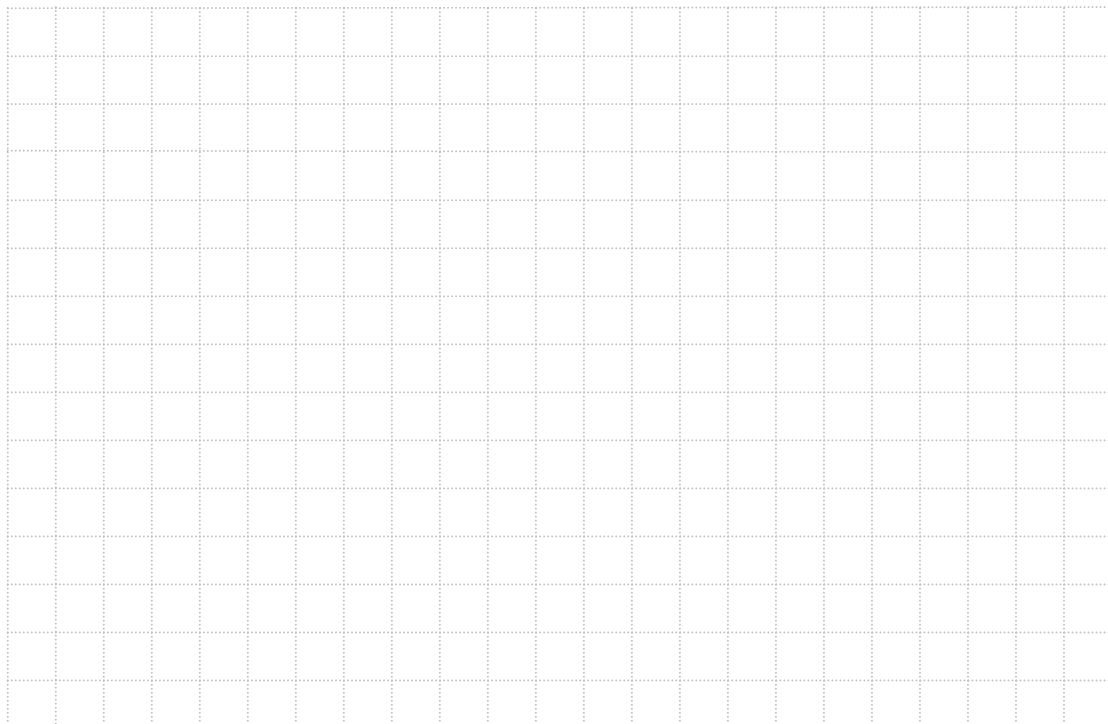
## Estrategia ante defectos – Trayectoria de movimiento

### ➤ ¿Es posible una estrategia de retorno global?

Si hay suficiente espacio sobre el robot para permitir al robot moverse libremente, ej en el caso de entarimar, éste puede moverse a una posición de retorno global y continuar la ejecución de programa desde allí.

O

Dependiendo de la posición en la cual ha ocurrido el defecto, si hay varios obstáculos entre el robot y la posición home, el robot se debe mover entre éstos sin colisión.



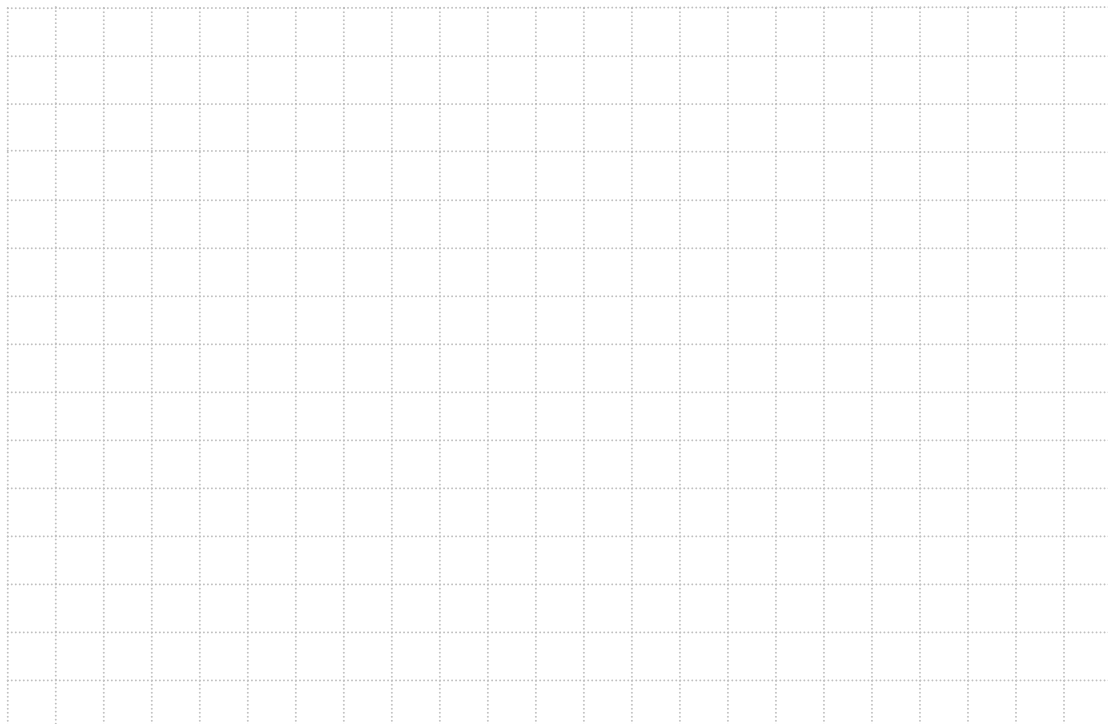
## Estrategia ante defectos – proceso

➤ ¿Qué procedimiento se debe adoptar ante un defecto?

Un robot utiliza una garra de vacío para coger una pieza y el vacío falla cuando el robot comienza a separarse. ¿Se debe repetir la operación o se debe parar el proceso inmediatamente por medio de un procedimiento de defectos?

O

Una robot pierde la pieza que lleva en la garra. ¿Debe frenarse el robot inmediatamente o se puede terminar sin interrupción?



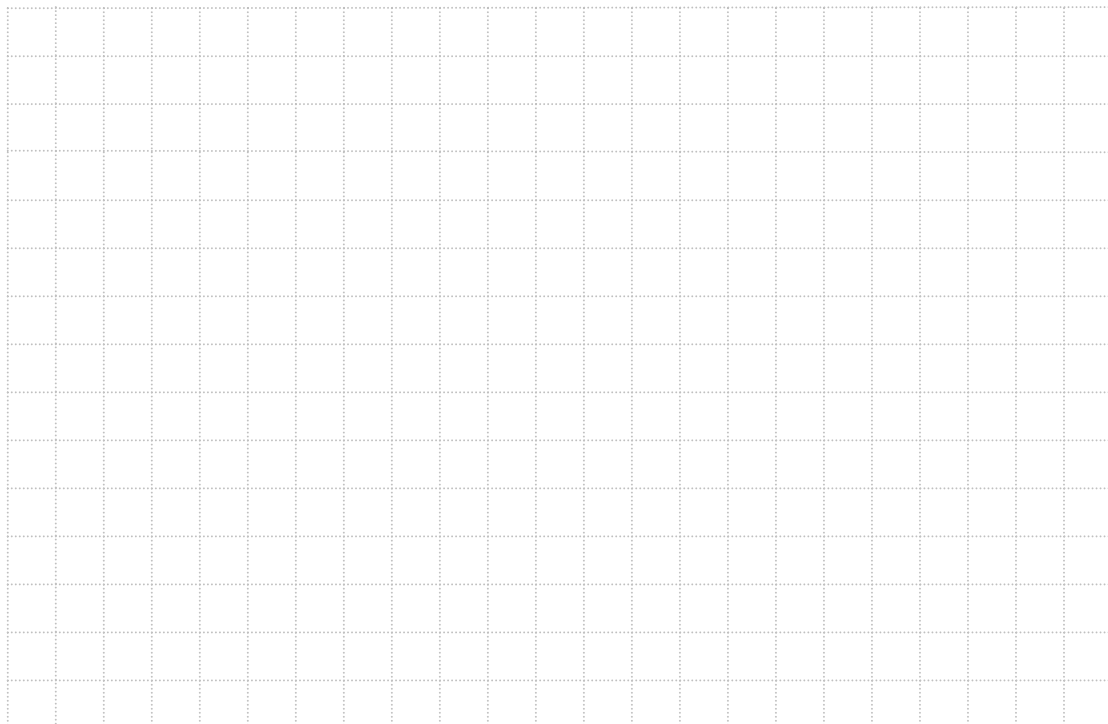
## Estrategia ante defectos – periferia

➤ ¿Deben resetearse las variables y las entradas/salidas?

Después de una avería, como resultado de la cual se ha perdido una pieza, el contador de piezas no debe ser incrementado.

O

Cuando el chequeo de la garra señala la pérdida de la pieza, la válvula de vacío se puede desactivar inmediatamente.

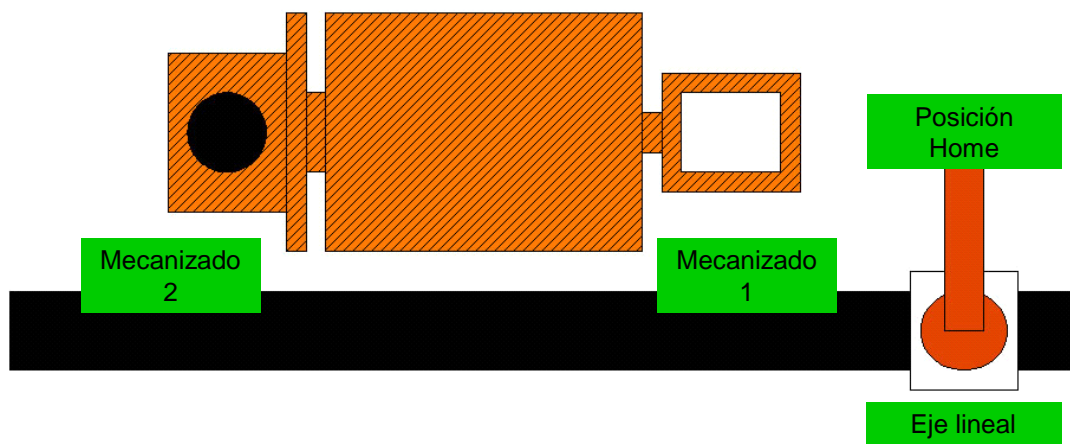




## **7.2 Estrategia del retorno automático**

Estrategia ante defectos – Sistema de 10 ejes (1)

Descripción del sistema:

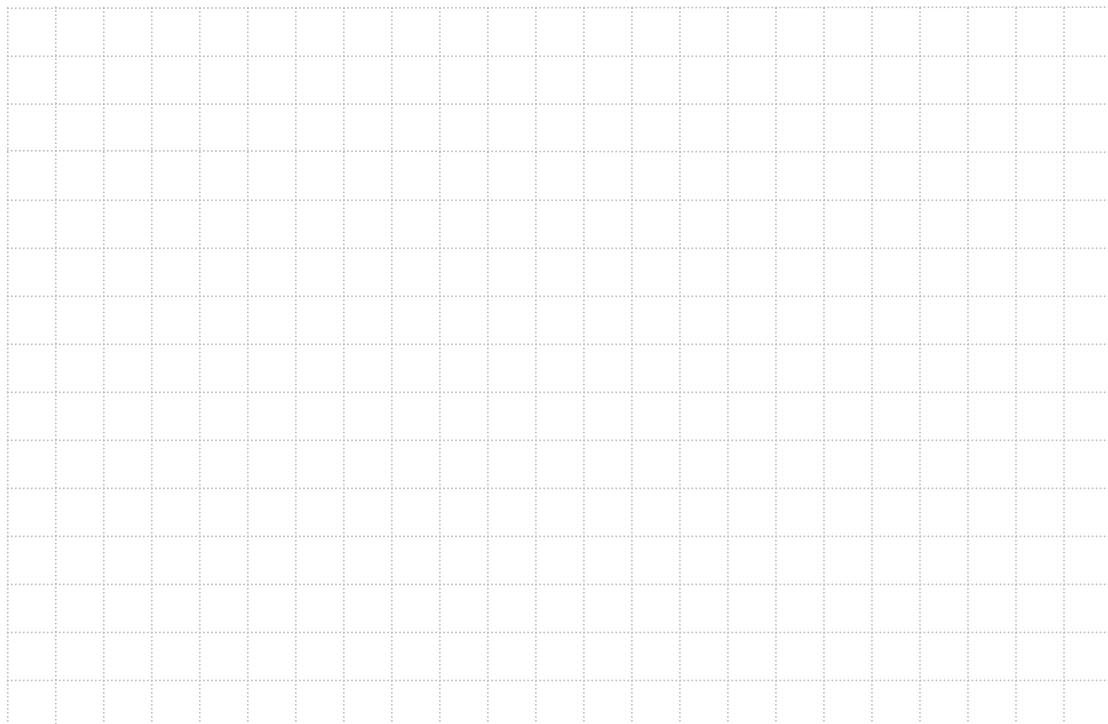


## Estrategia ante defectos – Sistema de 10 ejes (2)

### Descripción del programa:

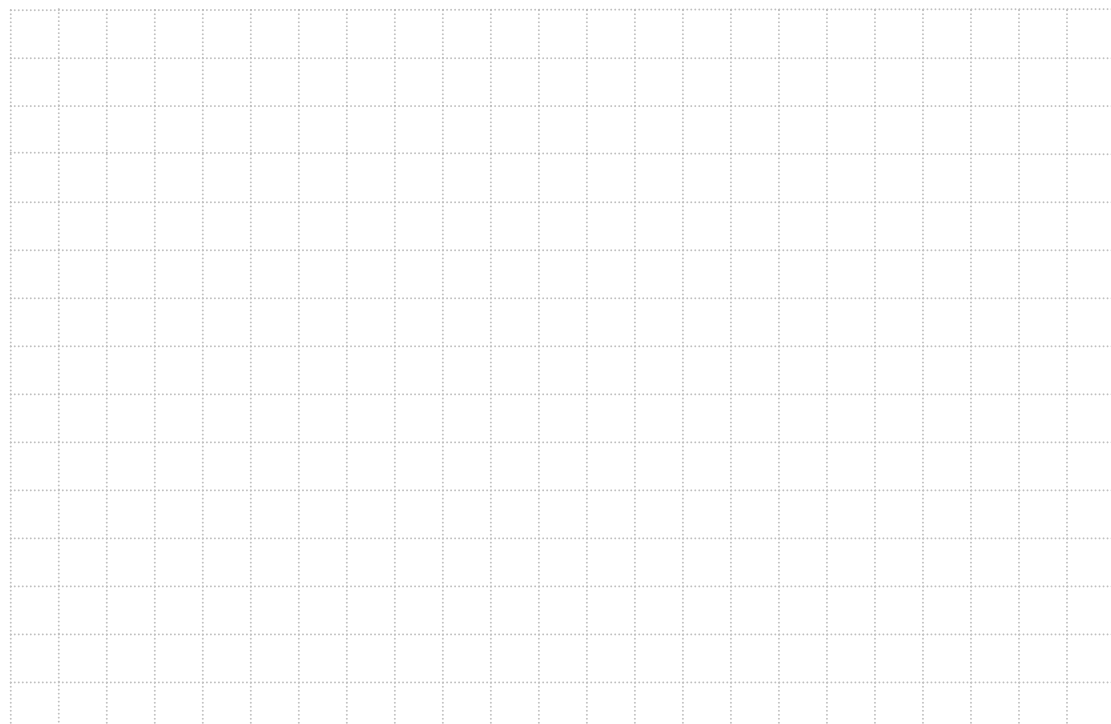
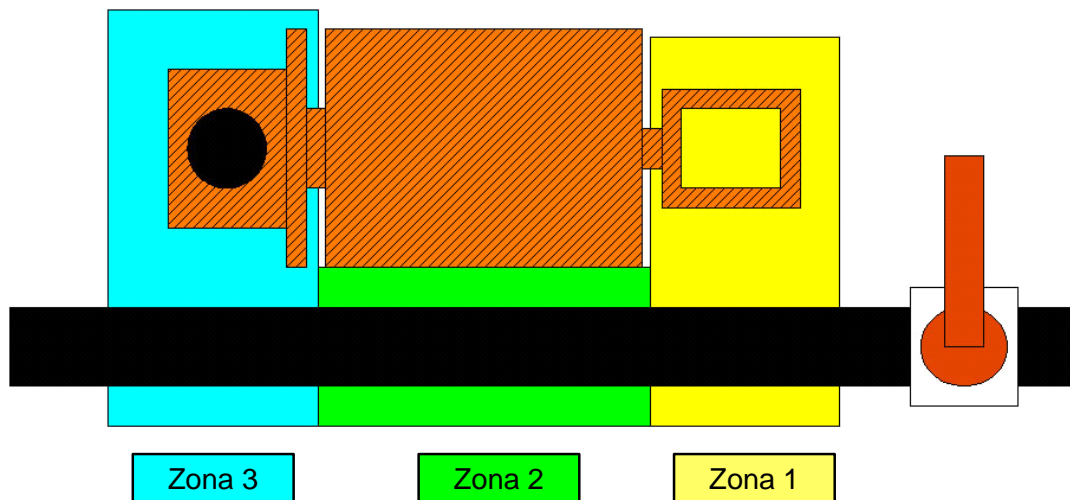
Partimos de un programa que mecaniza en la posición 1 y 2; este programa lo tendremos de plantilla.

La robot debe parar la ejecución de programa inmediatamente, debido a una condición de finalización del PLC, y se vuelva a su posición home a velocidad reducida ( $POV = 10$ ). Después de un mensaje de actualización, debe ser posible continuar el movimiento del robot mientras no hay otro defecto presente.

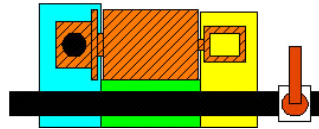


## Estrategia ante defectos – solución – preparación

Posible división del sistema en zonas:



## Estrategia ante defectos – solución – preparación

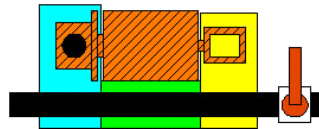


### Tareas en las zonas:

#### Zona 1:

- Opciones de movimiento del Robot
  - Moverse directamente al punto de retorno antes/después del mecanizado.
  - Durante el mecanizado: primero retracción del componente, y a continuación movimiento de retorno.
- Lógica
  - Take part counter into account
  - Desactivar la salida ADHESIVE\_ON en una manera apropiada

## Estrategia ante defectos – solución – preparación

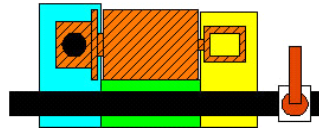


### Tareas en las zonas:

#### Zone 2:

- Opciones de movimiento del Robot
  - No es posible que el robot vuelva directamente a la posición home
  - El punto auxiliar delante del sistema puede ser útil
- Lógica
  - No hay lógica a tratar

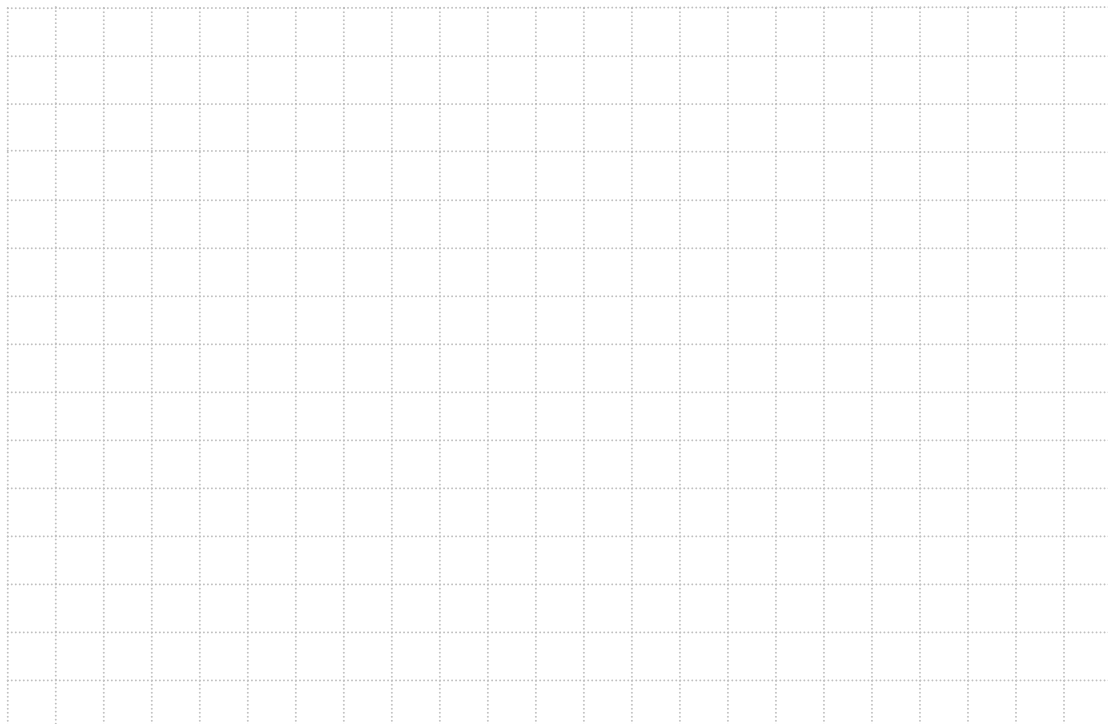
## Estrategia ante defectos – solución – preparación



### Tareas en las zonas:

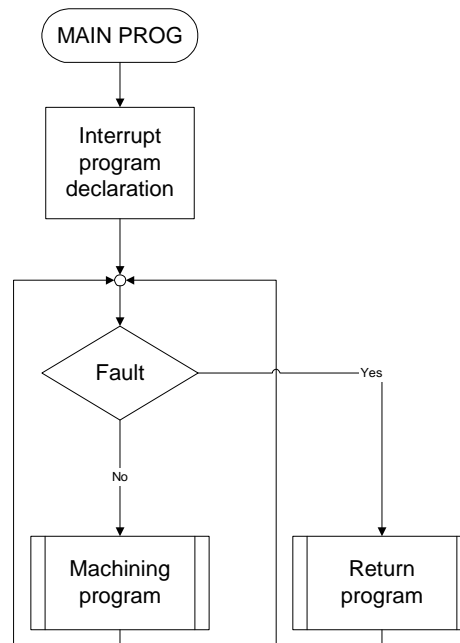
#### Zone 3:

- Opciones de movimiento del Robot
  - Moverse directamente al punto de retorno antes/después del mecanizado.
  - Durante el mecanizado: primero retracción del componente, y a continuación movimiento de retorno.
  - Observar la posición del eje 10 → Peligro de colisión
- *Lógica*
  - Descontar el contador
  - Desactivar la salida ADHESIVE\_ON en una manera apropiada



## Estrategia ante defectos – solución – flowchart del programa principal

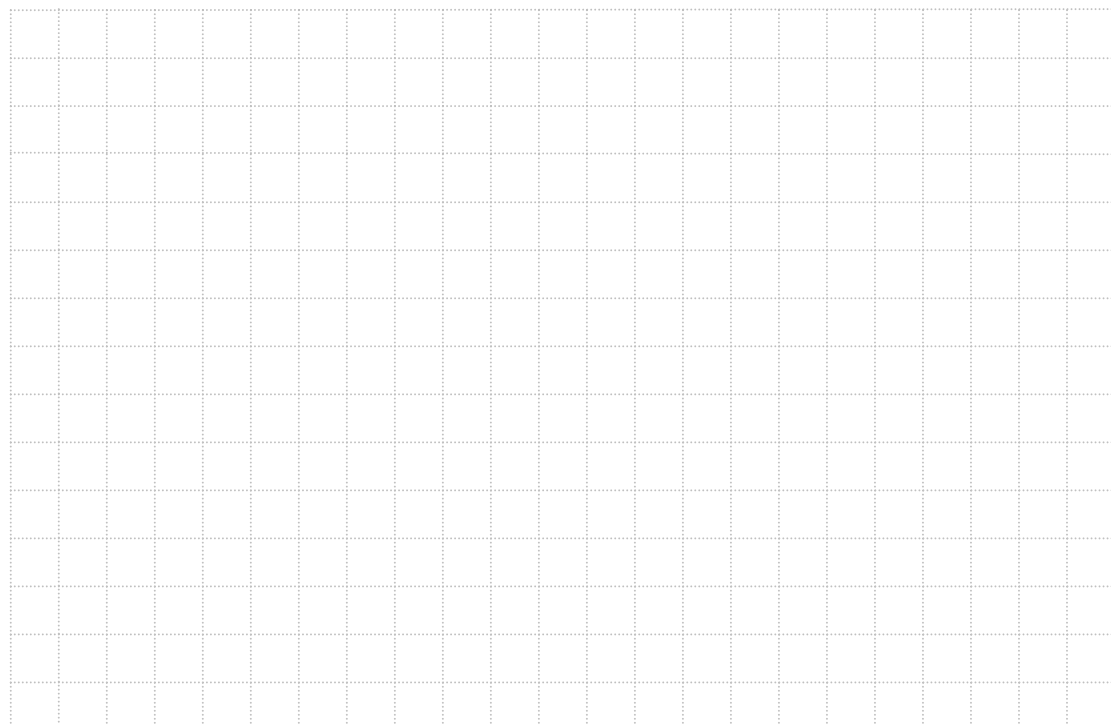
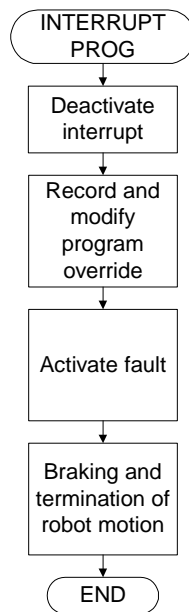
### PROGRAMA PRINCIPAL:





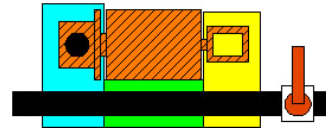
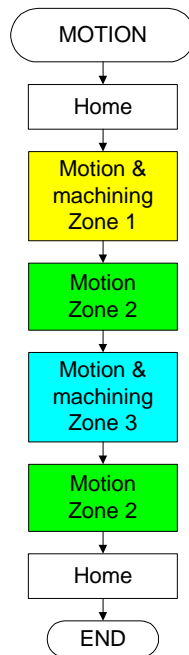
## Estrategia ante defectos – solución – flowchart del subprograma de interrupción

### Programa de interrupción



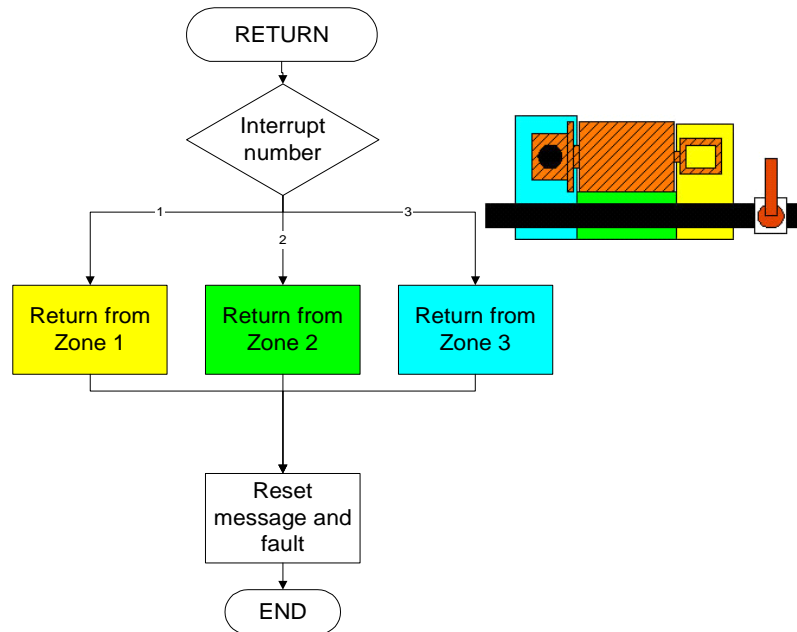
## Estrategia ante defectos – solución – flowchart del programa de mecanizado

Programa de  
mecanizado:



## Estrategia ante defectos – solución –flowchart del programa de retorno

Programa de  
retorno:



## **8. Trabajando con entradas / salidas**

### **8.1 Entradas / salidas digitales**

Entradas y salidas con el software de KUKA

0 I/O

1 Rob. Position

2 Variable

3 Diagnosis

4 Windows

0 Digital Inputs


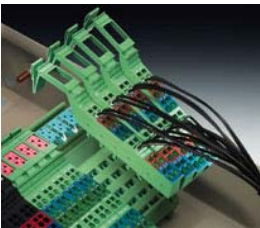
1 Digital Outputs

2 Analog Outputs

3 Automatic external

Inputs

	SYS	Inputs
1	Input	
2	Input	
3	Input	
4	Input	
5	Input	
6	Input	
7	Input	
8	Input	
9	Input	
10	Input	
11	Input	
12	Input	
13	Input	
14	Input	
15	Input	



0 I/O

1 Rob. Position

2 Variable

3 Diagnosis

4 Windows

0 Digital Inputs

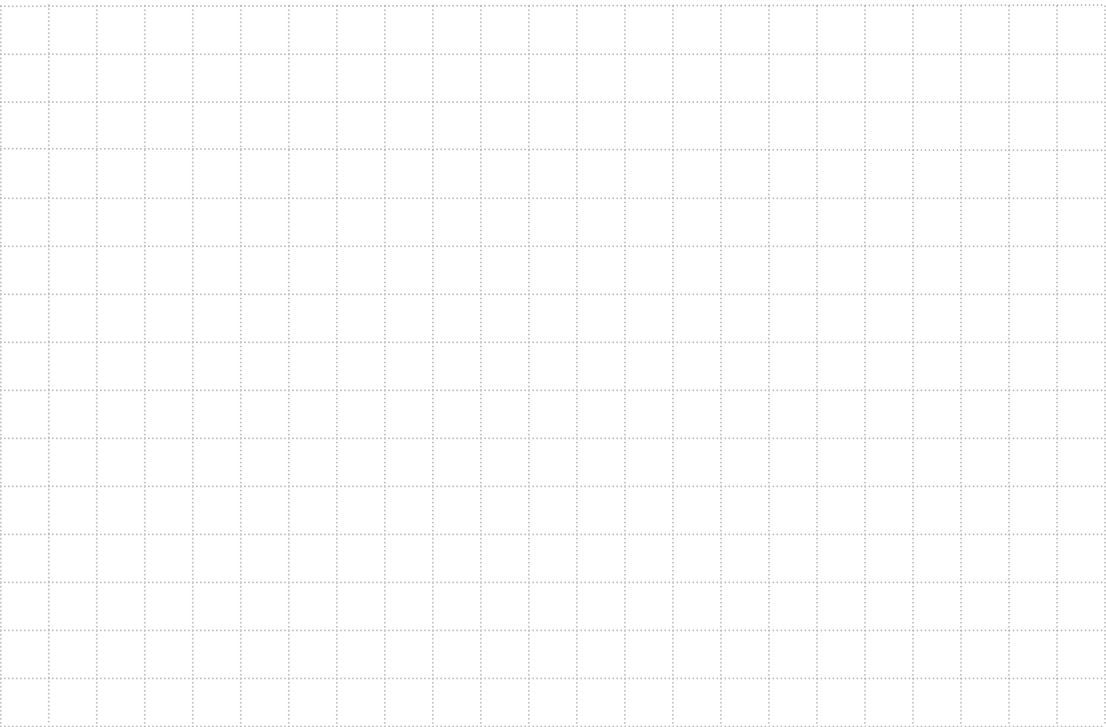
1 Digital Outputs

2 Analog Outputs

3 Automatic external

Outputs

	SYS	Outputs
1	Output	
2	Output	
3	Output	
4	Output	
5	Output	
6	Output	
7	Output	
8	Output	
9	Output	
10	Output	
11	Output	
12	Output	
13	Output	
14	Output	
15	Output	



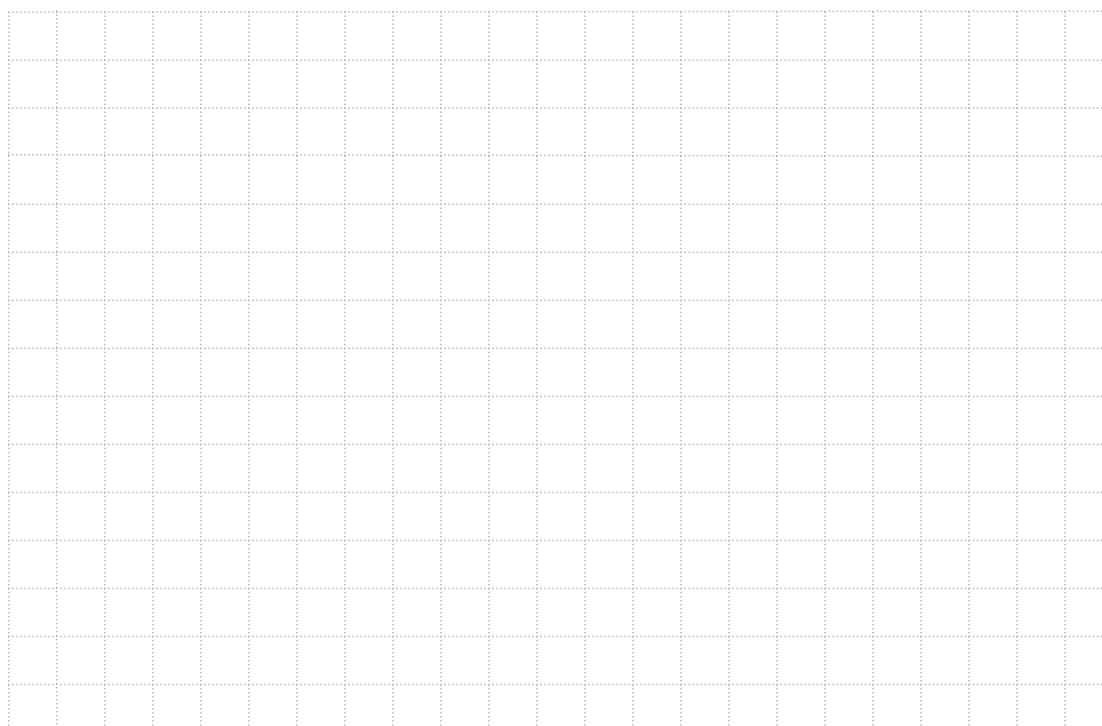
## Instrucciones entradas/salidas

- En el control de robot se puede utilizar un máximo de **4096 entradas y 4096 salidas**.
- Con el **KR C2**, se necesita un **módulo adicional de E/S**.
- Otras entradas/salidas se pueden configurar usando el **sistema bus de campo**.



La variable "\$SET\_IO\_SIZE" se utiliza para cambiar el número de entradas/salidas indicadas en la tabla. Este archivo se encuentra en Steu\Mada\\$option.dat

Valor de la variable	Numero de entradas/salidas preparadas
1 (default)	1024
2	2048
4	4096



## Instrucciones entradas/salidas

La entrada \$IN[1025] siempre está a TRUE; la entrada \$IN[1026] siempre está a FALSE. Estas variables se pueden utilizar tantas veces se desee.



**Por razones de seguridad, todas las instrucciones de entradas/salidas o utilizar las entradas/salidas mediante las variables de sistema provocan una parada del advance.**

Utilizar entradas/salidas mediante variables de sistema precedidas de la instrucción CONTINUE provoca que no se pare el advance.



## Entradas/salidas binarias

Si las entradas/salidas se direccionan individualmente, se refieren a entradas/salidas binarias. Las salidas binarias solo pueden tener 2 estados: Bajo o alto.

Sintaxis:

**\$OUT[No] = *Valor* or \$IN[No]**

Argumento	Tipo	Explicación
<i>Valor</i>	<b>BOOL</b>	FALSE: La salida se desactiva TRUE: La salida se activa

Ejemplo:

**\$OUT[1] = TRUE ;Salida 1 se activa**

**IF \$IN[3] == FALSE THEN ;Entrada 3 se consulta**



## Activar salidas en el punto final

Tan pronto como el robot ha llegado al punto final de una instrucción de movimiento, se pueden activar hasta **8 salidas**, y sin parar el advance.

Sintaxis:

**\$OUT\_C[No] = Valor**

Argumento	Tipo	Explicación
<i>Valor</i>	<b>BOOL</b>	FALSE: Salida se desactiva TRUE: Salida se activa



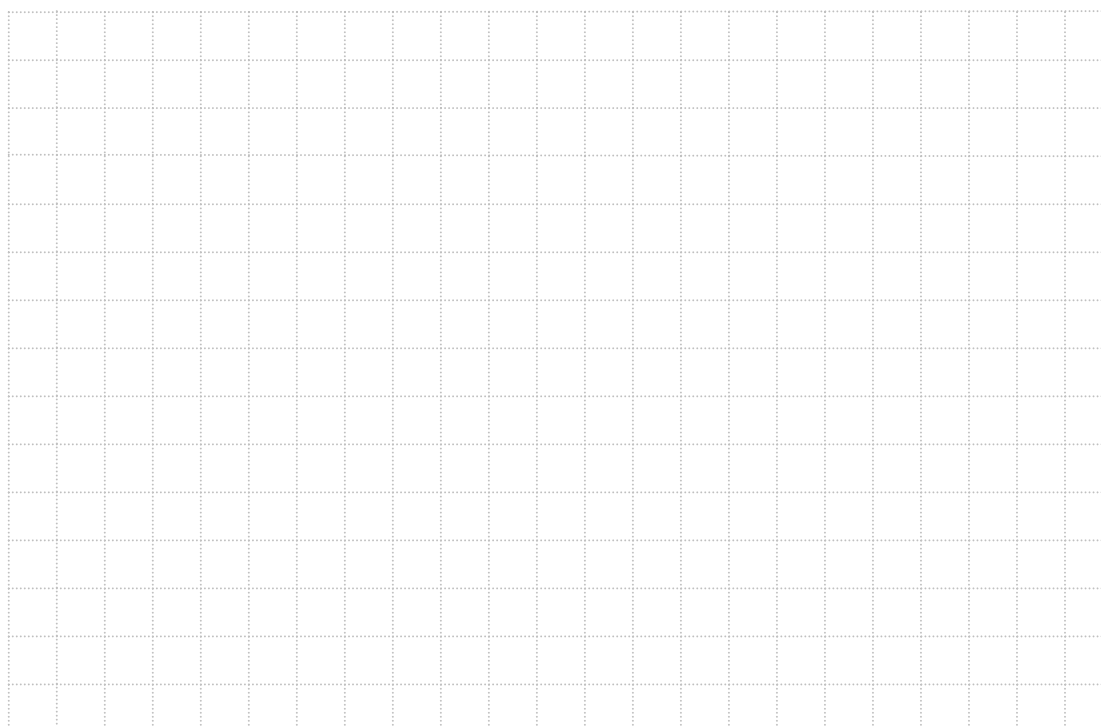
Cuando el cursor principal de funcionamiento alcanza el final del punto, la salida se se pone al valor booleano que era válido cuando se realizó la interpretación, incluso si el valor ha cambiado posteriormente.

01/2006

5

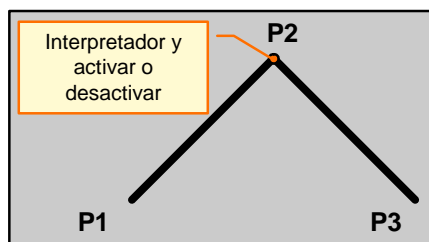
input\_output\_digital\_en.ppt

© Copyright by KUKA Roboter GmbH College

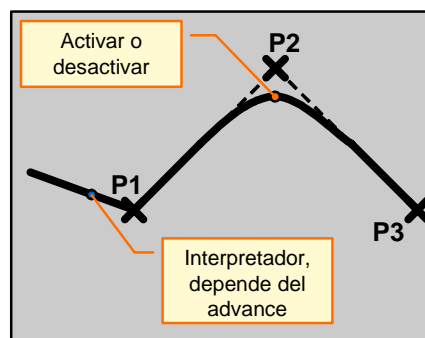


## Ejemplo: Comparación entre \$OUT y \$OUT\_C

```
LIN P2 C_DIS  
$OUT[10]=TRUE  
$OUT[11]=FALSE  
$OUT[12]=TRUE  
LIN P3
```



```
LIN P2 C_DIS  
$OUT_C[10]=TRUE  
$OUT_C[11]=FALSE  
$OUT_C[12]=TRUE  
LIN P3
```



## Declaración de una Signal

En el KRC es posible asignar un nombre individualmente a cada entrada o salida.

La **declaración signal**, como todas las declaraciones, debe estar situada en la sección de declaraciones del programa.

Sintaxis:

**SIGNAL** *Variable* \$OUT[No] or **SIGNAL** *Variable* \$IN[No]

Ejemplo:

```
SIGNAL BOTON $IN[6]  
INI  
...  
IF BOTON == FALSE THEN  
...  
ENDIF
```

o

```
IF $IN[6] == FALSE  
THEN  
...  
...  
ENDIF
```

## Entradas/salidas digitales



Se pueden agrupar entradas y salidas binarias usando la palabra clave **TO** (máximo 32).

### Sintaxis:

```
SIGNAL Variable $OUT[No] TO $OUT[No]
```

o

```
SIGNAL Variable $IN[No] TO $IN[No]
```

### Ejemplo:

```
SIGNAL MI_SALIDA_DIGITAL $OUT[10] TO $OUT[20]
```

```
...
```

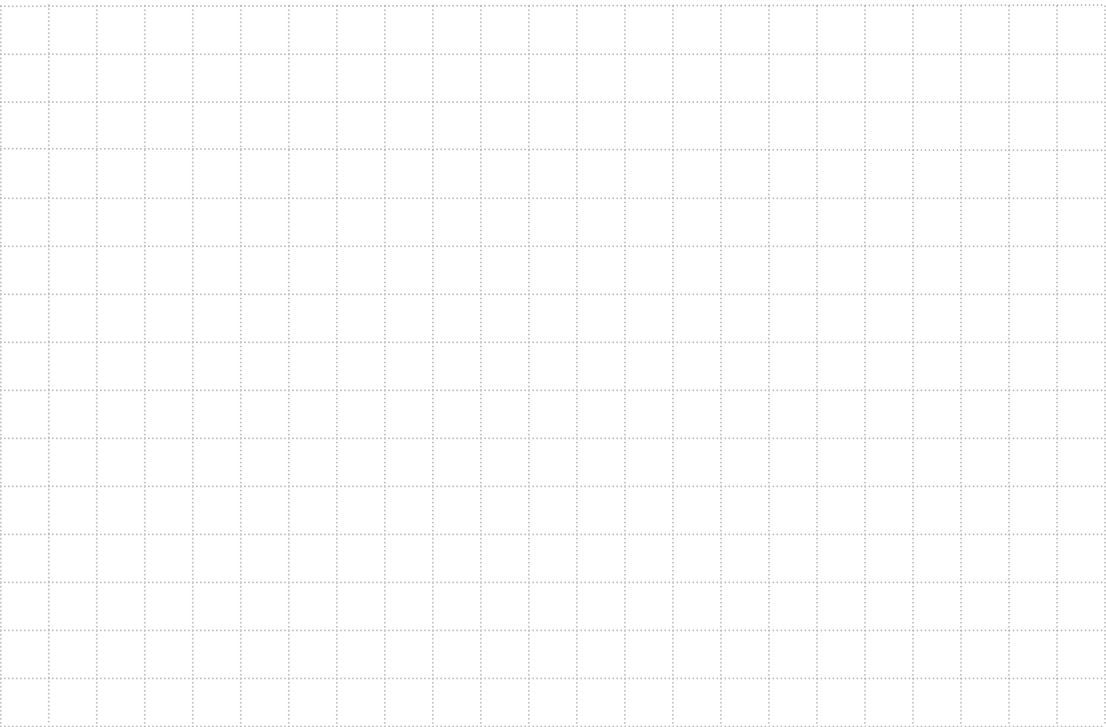
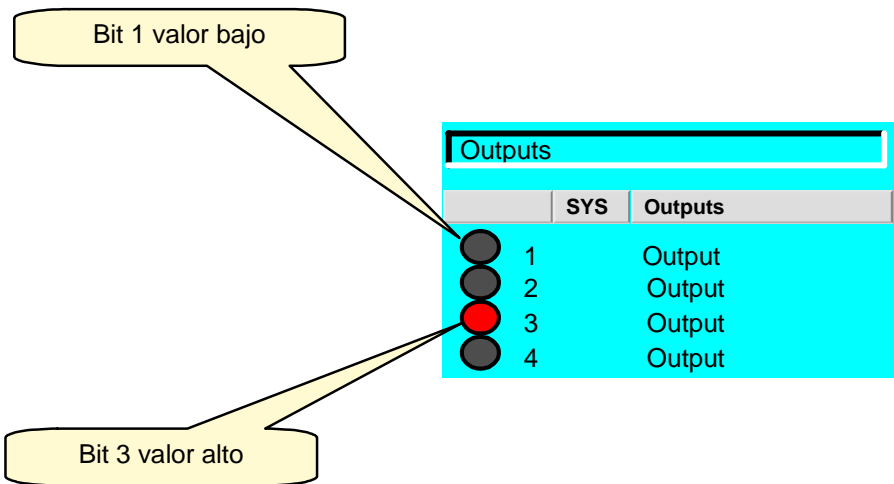
```
MI_SALIDA_DIGITAL = 35
```

```
...
```

```
MI_SALIDA_SALIDA = 'B100011' ; Decimal:  $2^5+2^1+2^0=35$ 
```



Visualización de las salidas



## Importante información sobre pulso en las salidas



- Un máximo de 16 salidas del pulso se puede programar simultáneamente.
- Se pueden programar pulsos a nivel alto como a nivel bajo.
- “Resetear Programa” y “Cancelar Programa” ambos terminal el pulso.
- Los pulsos pueden ser programados a nivel programador.
- Los PULSOS hacen parar el advance.



Un pulso **NO** se termina por:

- Una PARADA DE EMERGENCIA, stop por el operador o un error que induce un stop,
- Alcanzar el final de programa (instrucción END),
- pulsar la tecla start si el pulso se ha programado antes de la primera instrucción de movimiento y el robot todavía no ha alcanzado COI.



## Pulso en las salidas

Las salidas pueden ser activadas y desactivadas durante un periodo de tiempo específico usando la instrucción PULSE.

Sintaxis:

**PULSE ( \$OUT[*No*], *Valor*, *Tiempo* )**

Argumento	Tipo	Explicación
<i>Valor</i>	BOOL	FALSE: Entrada/salida se desactiva TRUE: Entrada/salida se activa
<i>Tiempo</i>	REAL	Rango de valores: 0.012 s...2 <sup>31</sup> s Incremento: 0.1 s; El control redondea los valores a la decima de un segundo.



## **8.2 Entradas predefinidas**



## Entradas digitales predefinidas

El control tiene 6 entradas digitales disponibles, las cuales se pueden leer usando el nombre de la variable \$DIGIN1...\$DIGIN6. Las entradas se incluyen en las entradas normales de usuario. Pueden ser de 32 bits de longitud y tener la correspondiente salida strobe.



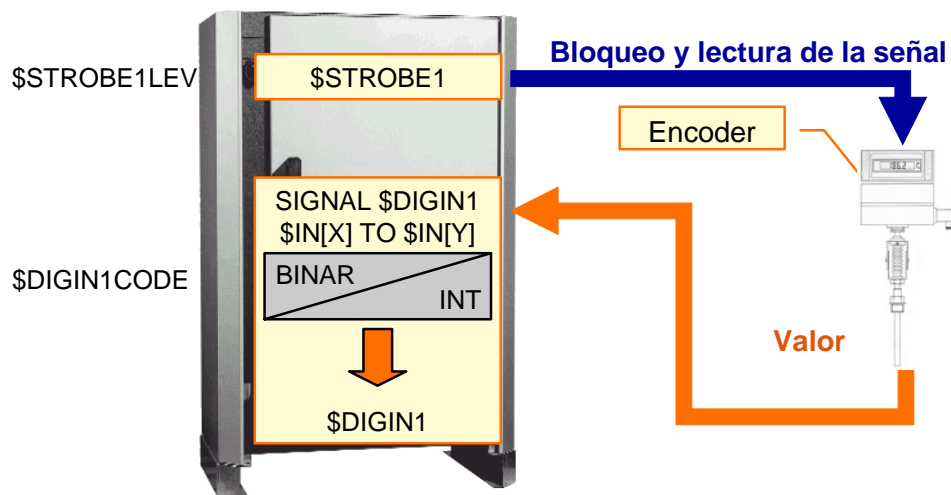
Las entradas se configuran en los datos de máquina:  
"/steu/mada/\$maschine.dat".

Las variables siguientes se deben definir aquí:

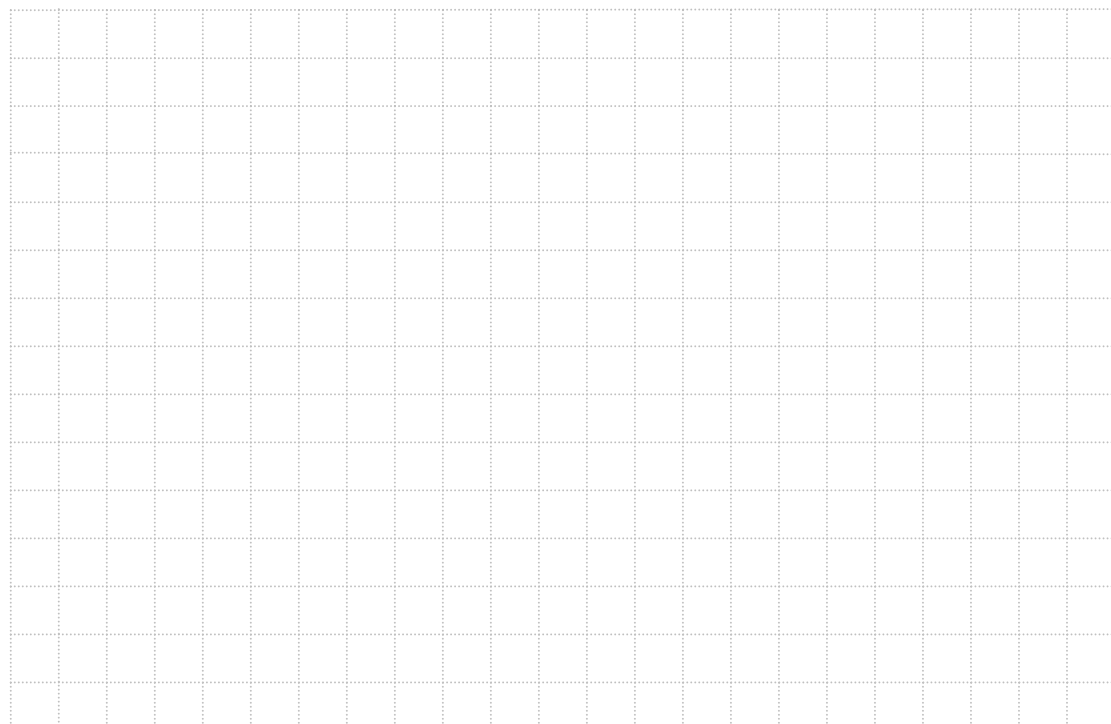
- \$DIGIN1      ...      \$DIGIN6
- \$DIGIN1CODE    ...    \$DIGIN6CODE
- \$STROBE1      ...      \$STROBE6
- \$STROBE1LEV    ...    \$STROBE6LEV



## Entradas digitales predefinidas



Aplicación: Registrar una lectura de la medida



## Entradas digitales predefinidas

Signal **\$DIGIN3** \$IN[1000] to \$IN[1011] *Rango y tamaño de la entrada digital*

DECL DIGINCODE **\$DIGIN3CODE** = #UNSIGNED or #SIGNED

#UNSIGNED: *aquí 12 bits sin signo (rango de valores 0 ao +4095)*

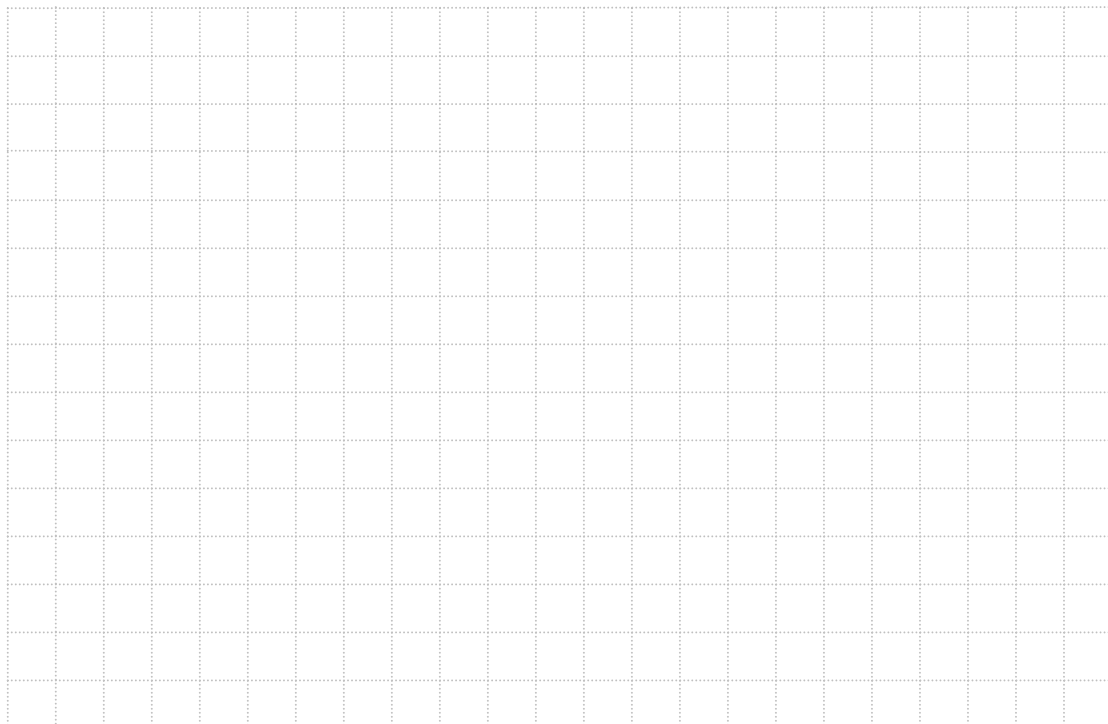
#SIGNED: *aquí 12 bits con signo (range de valores -2048 to +2047)*

Signal **\$STROBE3** \$OUT[1000]

*Salida que congela la señal del dispositivo externo para que éste pueda ser leído*

Bool **\$STROBE3LEV** =TRUE

*Strobe está en un pulso alto*



## Lectura cíclica de la entrada digital predefinida

Predefinición de la entrada:

**DIGIN ON** *Valor = Factor \* Nombre\_señal < +/- Offset>*

Finalización de la supervisión:

**DIGIN OFF** *Nombre\_señal*

Ejemplo:

```
...  
INT NUMERO  
...  
DIGIN ON NUMERO = FACTOR * $DIGIN2 + OFFSET  
...  
DIGIN OFF $DIGIN2  
...
```

## **8.3 Entradas / salidas analógicas**

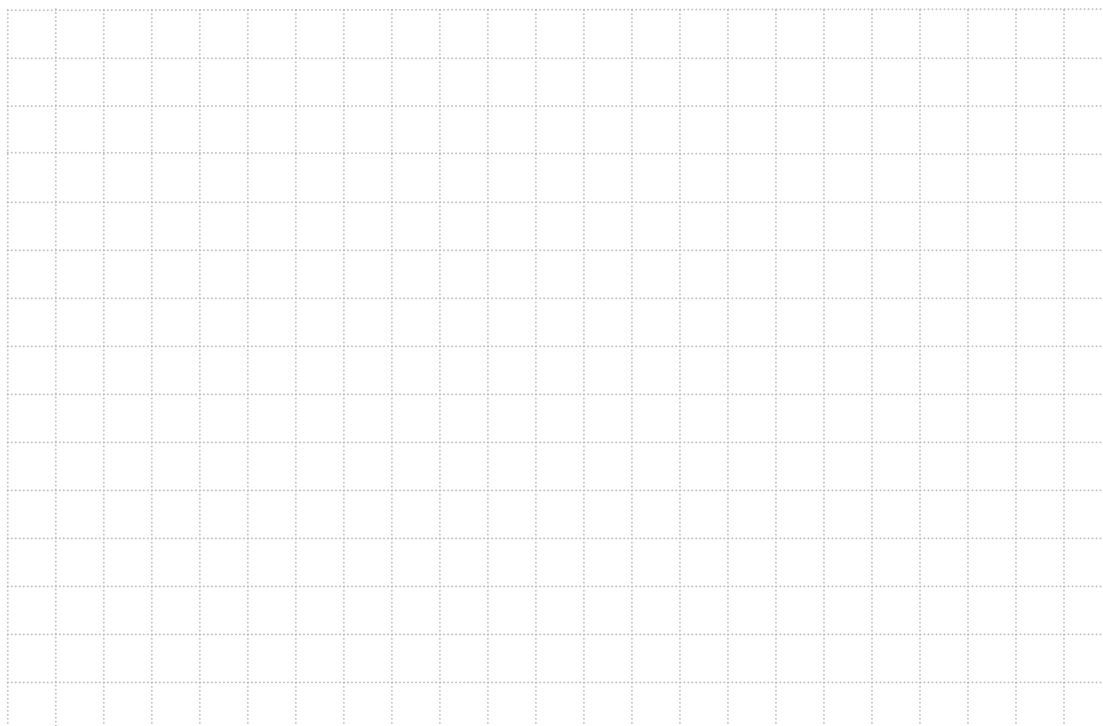
## Entradas/salidas analógicas

- El KRC tiene **32 entradas analógicas** y **32 salidas analógicas**.
- Las salidas analógicas se pueden leer o escribir usando las **variables de sistema \$ANOUT[1]...[32]**.
- Las entradas analógicas solo se pueden leer usando las **variables de sistema \$ANIN[1]...[32]**.
- Para la función de entradas/ salidas analógicas se requiere un **Bus de campo** opcional.



\* con software 4.x: solo **8** entradas analógicas / **16** salidas analógicas

01/2006 1  
input\_output\_predefined\_en.ppt  
© Copyright by KUKA Roboter GmbH College



## Salidas analógicas (estáticas)

Información general sobre salidas analógicas:

*Los valores de las 32 salidas analógicas del KRC están comprendidos entre los valores -1.0 ... +1.0 y son escalados a un voltage de salida comprendido entre  $\pm 10.0$  V.*

Ejemplo: Asignación de un valor estático

```
...  
$ANOUT[2] = 0.5 ;Canal de salida analógico 2 se pone a +5 V  
...  
Válvula = -0.9  
$ANOUT[5] = Válvula ;Canal de salidas analógico 5 se pone a -9 V  
...
```

Estás asignaciones son estáticas porque el valor de salida del canal correspondiente no cambiará hasta que se haga una asignación explícita a la variable de sistema \$ANOUT[No] correspondiente.



## Salidas analógicas (dinámicas)

Arranque de la salida analógica cíclica:

**ANOUT ON** *Nombre\_Señal* = *Factor* \* *Elemento\_Control* +/- *Offset*  
<Delay = t> <Minimum = U1> <Maximum = U2>

Final de la salida analógica cíclica:

**ANOUT OFF** *Nombre\_Señal*

Ejemplo:

```
...  
SIGNAL ADHESIVO $ANOUT[1]  
...  
ANOUT ON ADHESIVO = 0.5 * $VEL_ACT  
...  
ANOUT OFF ADHESIVO  
...
```



## Salidas analógicas (dinámicas)

Arranque de la salida analógica cíclica: con DELAY

**ANOUT ON** *Nombre\_Señal* = *Factor* \* *Elemento\_Control* +/- *Offset*  
<Delay = t> <Minimum = U1> <Maximum = U2>

### +/-Offset

*Un offset se puede programar opcionalmente como un elemento de control. El Offset puede ser una variable, una declaración signal, una señal analógica o una constante.*

Ejemplo:

```
...  
SIGNAL ADHESIVO $ANOUT[1]  
...  
ANOUT ON ADHESIVO = 0.5 * $VEL_ACT + 0.15  
...  
ANOUT OFF ADHESIVO  
...
```

01/2006

4

input\_output\_predefined\_en.ppt

© Copyright by KUKA Roboter GmbH College



## Salidas analógicas (dinámicas)



Se puede tener un máximo de 4 salidas analógicas activas.

Arranque de la salida analógica cíclica: con DELAY

**ANOUT ON** *Nombre\_señal* = *Factor* \* *Elemento\_Control* +/- *Offset*  
<Delay = t> <Minimum = U1> <Maximum = U2>

**<DELAY = t>** Usando la palabra clave *DELAY* e incorporar una cantidad de tiempo positiva o negativa en segundos, la señal de salida puede ser retrasada (+) o puede ser adelantada (-).

Ejemplo:

```
...  
SIGNAL ADHESIVO $ANOUT[1]  
...  
ANOUT ON ADHESIVO = 0.5 * $VEL_ACT DELAY = 0.75  
...  
ANOUT OFF ADHESIVO  
...
```

## Salidas analógicas (dinámicas)

Arranque de la salida analógica cíclica: con DELAY

**ANOUT ON** *Nombre\_Señal* = *Factor* \* *Elemento\_Control* +/- *Offset*  
<Delay = t> <Minimum = U1> <Maximum = U2>

**<Minimum U1>** La palabra clave *MINIMUM* define el mínimo voltage presente en al salida. Los valores permitidos son -1.0 ... +1.0 (= -10 V ... +10 V). El valor, tambien, puede ser una variable. (Condición  $U1 < U2$ )

Ejemplo:

```
...  
SIGNAL ADHESIVO $ANOUT[1]  
...  
ANOUT ON ADHESIVO = 0.5 * $VEL_ACT MINIMUM = 0.30  
...  
ANOUT OFF ADHESIVO  
...
```

## Salidas analógicas (dinámicas)

Arranque de la salida analógica cíclica: con DELAY

**ANOUT ON** *Nombre\_Señal* = *Factor* \* *Elemento\_Control* +/- *Offset*  
<Delay = t> <Minimum = U1> <Maximum = U2>

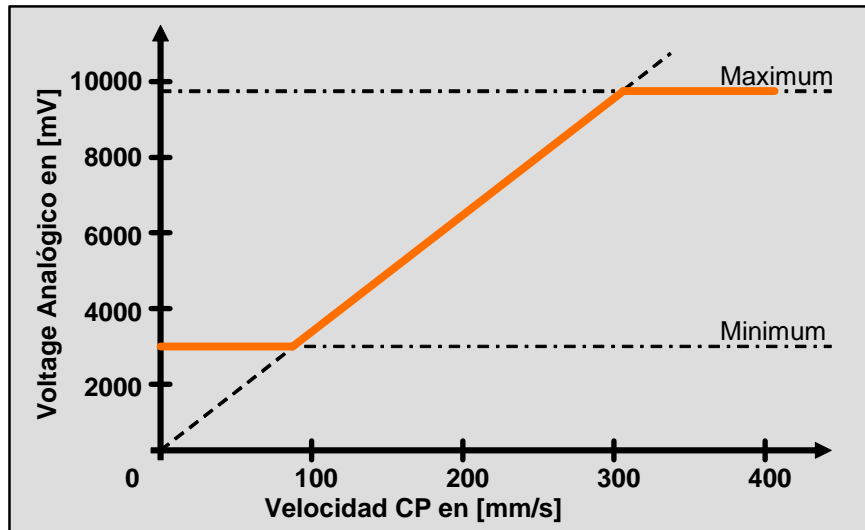
**<Maximum U2>** *La palabra clave MAXIMUM define el máximo voltage presente en la salida. Los valores permitidos son -1.0 ... +1.0 (= -10 V ... +10 V). El valor, tambien, puede ser una variable. (Condición U2>U1)*

Ejemplo:

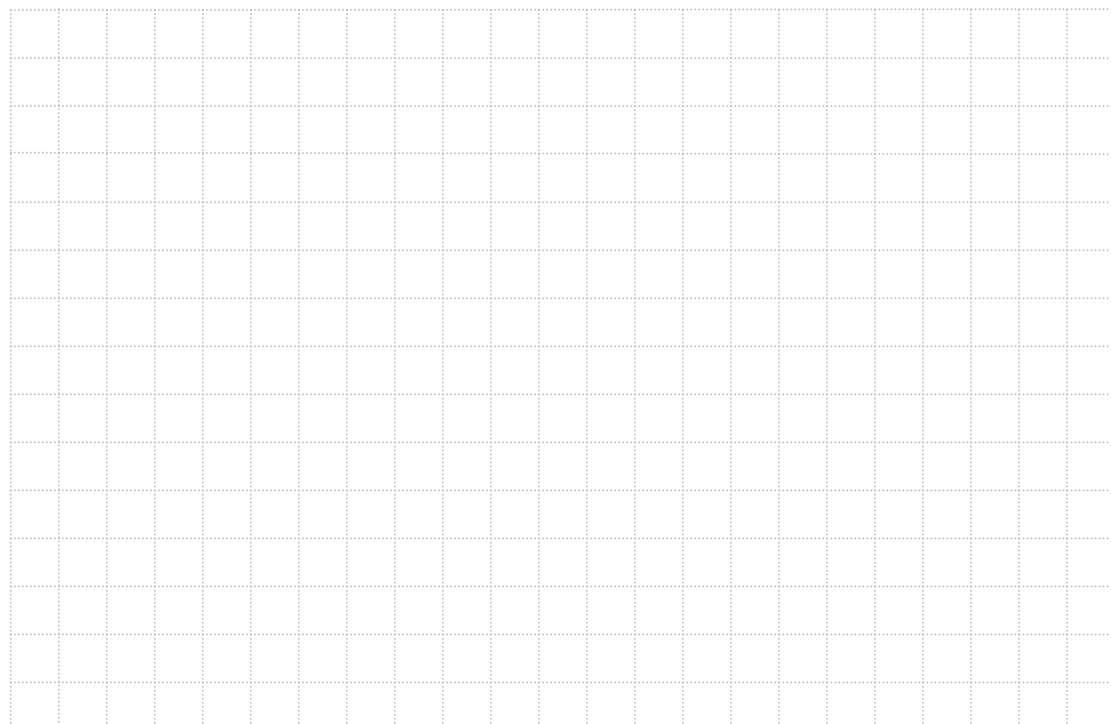
```
...  
SIGNAL ADHESIVO $ANOUT[1]  
...  
ANOUT ON ADHESIVO = 0.5 * $VEL_ACT MAXIMUM = 0.97  
...  
ANOUT OFF ADHESIVO  
...
```

Voltaje analógico dependiendo de la velocidad CP

Ejemplo



ANOUT ON ADHESIVO = 3.375 \* \$VEL\_ACT MINIMUM = 0.30 MAXIMUM = 0.97



## Entradas analógicas (Estáticas)

Información general sobre entradas analógicas:

*Las 32 entradas analógicas pueden ser leídas usando las variables \$ANIN[1] to \$ANIN[32] por la asignación de un valor a una variable REAL:*

Ejemplo: Asignación de un valor estático

```
SIGNAL SENSOR3 $ANIN[5]  
REAL Parte,Medicion  
...  
PARTE = $ANIN[2] ; Parte tiene el valor Real del canal analógico 2  
...  
O  
...  
Medicion = SENSOR3 ; Medicion tiene el valor Real del canal  
analógico 5
```

*Los rango de valores de \$ANIN[No] está comprendido entre +1.0 y -1.0 y representa un voltage de entrada de +10 V a -10 V.*



## Entradas analógicas (dinámicas)



Se puede tener un máximo de 3 entradas analógicas activas.

Lectura cíclica de la entrada analógica:

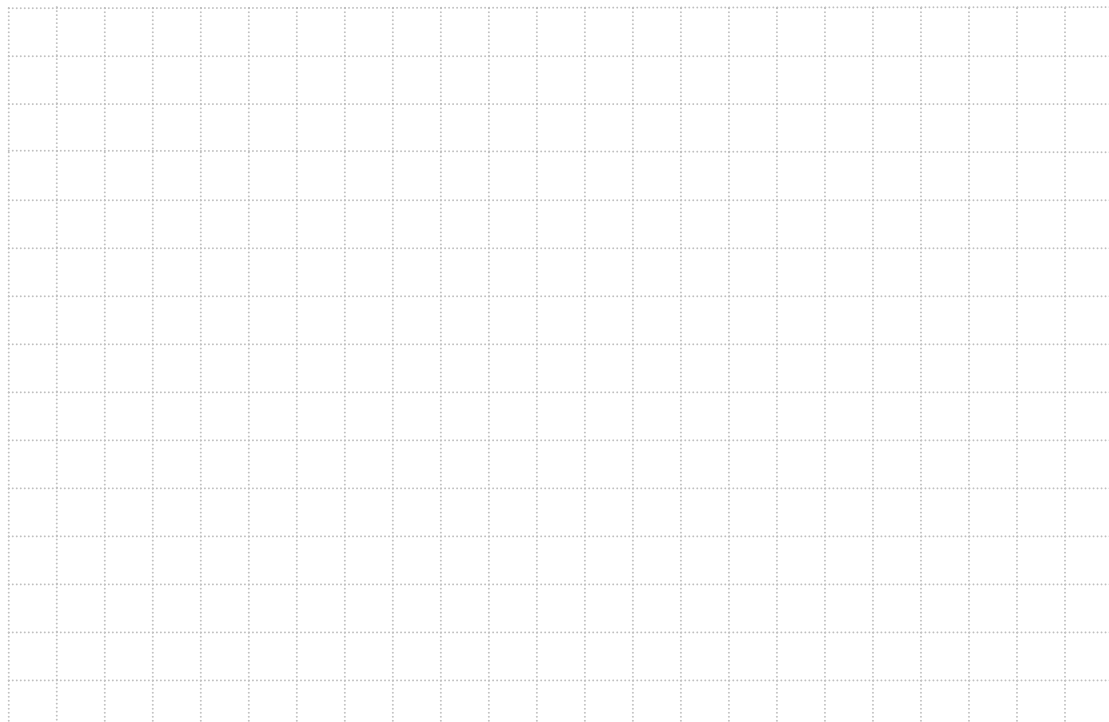
**ANIN ON** *Valor = Factor \* Nombre\_señal +/- Offset*

Terminar la supervisión cíclica:

**ANIN OFF** *Nombre\_señal*

Ejemplo:

```
...  
SIGNAL CORRECCION $ANIN[3]  
; REAL PARTE esta declaración se debe realizar en el  
    archivo DAT (local o $config.dat)  
...  
ANIN ON PARTE = 1.4 * CORRECCION  
...  
ANIN OFF CORRECCION  
...
```



## Entradas analógicas (dinámicas)

Lectura cíclica de una entrada analógica:

**ANIN ON** *Valor = Factor \* Nombre\_señal +/- Offset*

### **+/-Offset**

*Un offset se puede programar opcionalmente como un elemento de control. El Offset puede ser una variable, una declaración signal o una constante.*

Ejemplo:

```
...  
SIGNAL CORRECCION $ANIN[3]  
REAL PARTE  
...  
ANIN ON PARTE = 1.4 * CORRECCION -1.27  
...  
ANIN OFF CORRECCION  
...
```