

---

# **Grado de Electrónica Industrial y Automática**

## **Curso 2008-2009**

### **Robótica industrial**

## **8. Lenguaje RAPID (ABB)**

Departamento de Ingeniería de Sistemas y Automática  
UNIVERSIDAD CARLOS III DE MADRID

---



# Lenguaje RAPID

---

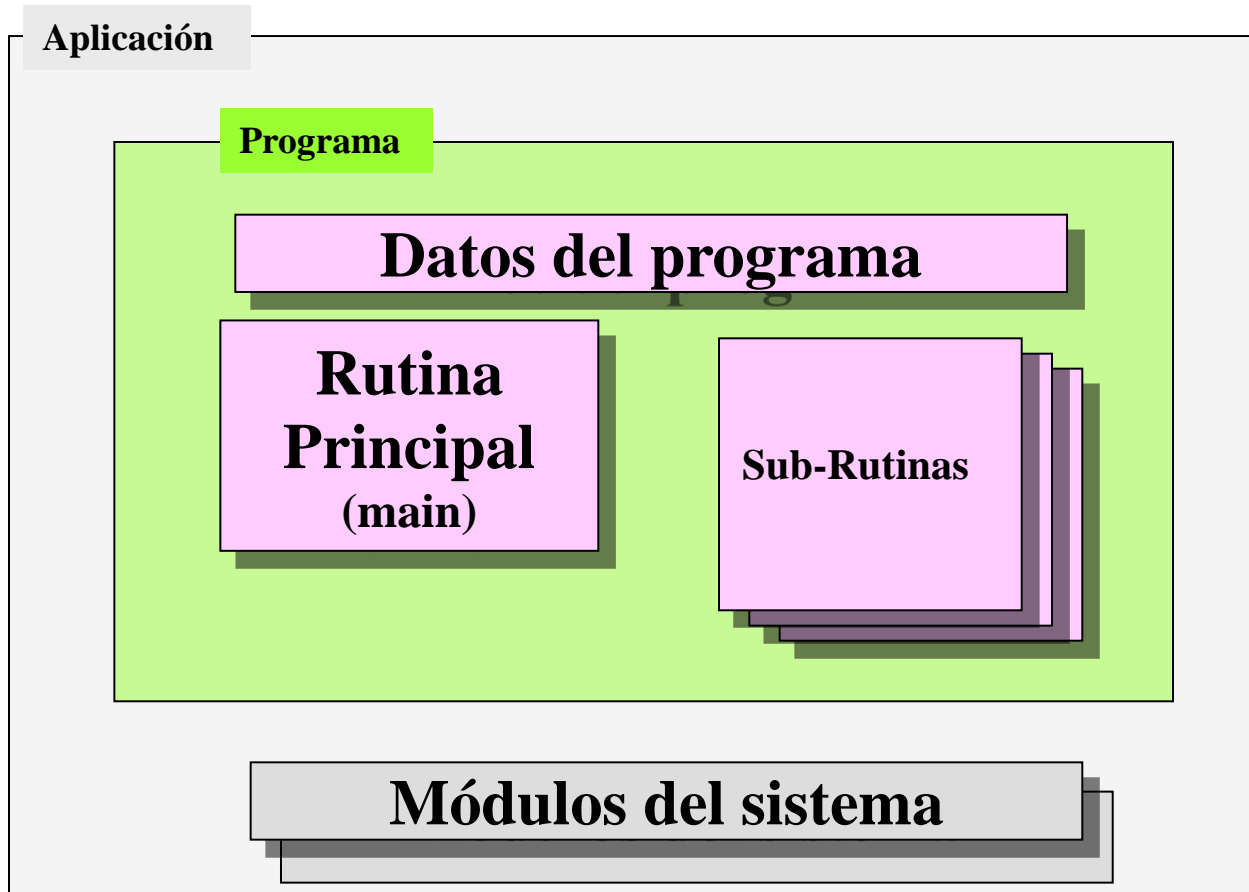
## Estructura del lenguaje

- Una aplicación RAPID consta de un programa y una serie de módulos del sistema.
- El programa es una secuencia de instrucciones que controlan el robot y en general consta de tres partes:
  - Una rutina principal (main):  
Rutina donde se inicia la ejecución.
  - Un conjunto de sub-rutinas:  
Sirven para dividir el programa en partes más pequeñas a fin de obtener un programa modular.
  - Los datos del programa  
Definen posiciones, valores numéricos, sistemas de coordenadas, etc.



# Lenguaje RAPID

## Estructura del lenguaje





# Lenguaje RAPID

---

## Elementos básicos

### ■ Identificadores

Permiten nombrar módulos, rutinas, datos y etiquetas.

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

El primer carácter es siempre una letra.

Longitud máxima 16.

Diferencia entre mayúsculas y minúsculas.

### Palabras reservadas

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



# Lenguaje RAPID

---

## Elementos básicos

### ■ Espacios y caracteres de fin de línea

RAPID es un lenguaje sin formatos, en consecuencia los espacios pueden utilizarse en cualquier parte excepto en: identificadores, palabras reservadas, valores numéricos.

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

### ■ Valores numéricos

Enteros	3,	-100	3E2
Decimal	3.5	-0.987	-245E-2

### ■ Valores lógicos

Se expresa como	<i>TRUE</i>	<i>FALSE</i>
-----------------	-------------	--------------

---



# Lenguaje RAPID

---

## Elementos básicos

### ■ Valores de cadena

Secuencia de caracteres entre comillas.

*“Esto es una cadena”*

### ■ Comentarios

Sirven para facilitar la comprensión del programa, ocupan una línea entera comenzando con el símbolo !, finaliza con un carácter de fin de línea

*! Esto es un comentario*

### ■ Comodines

Se utilizan para representar de forma temporal ciertas partes del programa que aún no han sido definidas

---

<DDN>	Declaración de datos
<RDN>	Declaración de una rutina.



# Lenguaje RAPID

---

## Elementos básicos

### ■ Encabezado de archivo

Un archivo de un programa siempre se inicia con:

```
%%%
```

```
VERSION:1
```

(Versión M94 del programa)

```
LANGUAGE:ENGLISH
```

(Cualquier idioma)

```
%%%
```



## Módulos

### ▪ Módulos del programa

Puede estar formado de diferentes daos y rutinas.

Uno de los módulos contiene el procedimiento de entrada, un procedimiento global de entrada llamado **main**.

### ▪ Módulos del sistema

Sirven para definir datos y rutinas normales del sistema, como por ejemplo las herramientas.

### ▪ Declaración

```
MODULE      <nombre_módulo> [<Lista de atributos>]
              <Lista declaración de datos>
              <Lista declaración rutina>

ENDMODULE
```

---





## Módulos

### ▪Declaración

[<Lista de atributos>]	:
SYSMODULE	: Módulo del sistema.
NOSTEPIN	: No se podrá entrar durante ejecución paso a paso.
VIEWONLY	: No podrá ser modificado.
READONLY	: No podrá ser modificado pero sí sus atributos.



# Lenguaje RAPID

---

## Rutinas

### ■ Tres tipos:

#### Procedimientos

```
PROC <nombre procedimiento> ( Lista de parámetros )  
    <Lista de declaraciones de datos>;  
    <Lista de instrucciones>;  
    ERROR <lista instrucciones>;  
ENDPROC
```

#### Funciones

```
FUNC <tipo valor dato> ( Lista de parámetros )  
    <Lista de declaraciones de datos>;  
    <Lista de instrucciones>;  
    RETURN dato;  
    ERROR <lista instrucciones>;  
ENDFUNC
```

#### Interrupciones

```
TRAP <nombre trap>  
    <Lista de declaraciones de datos>;  
    <Lista de instrucciones>;  
    ERROR <lista instrucciones>;  
ENDTRAP
```

# Lenguaje RAPID

# Tipos de Datos

- **bool**

**VAR bool** <identificador>:= <valor>  
                                   <valor>: **TRUE** / **FALSE**                   <expresión lógica>

### Ejemplo:

```
VAR bool flag;  
flag:=TRUE;  
flag:= valor1 > valor2;
```

- **clock**

**VAR klok** <identificador>;                      Tiempo máximo: 4.294.967 seg. ≈ 49,7 días  
**ClkReset, ClkStart, ClkStop, y ClkRead**

### Ejemplo:

**VAR klok** reloj;  
**ClkReset** reloj;

# Lenguaje RAPID

# Tipos de Datos

- **bool**

**VAR bool** <identificador>:= <valor>  
                                   <valor>: **TRUE** / **FALSE**                   <expresión lógica>

### Ejemplo:

```
VAR bool flag;  
flag:=TRUE;  
flag:= valor1 > valor2;
```

- **cl**ok

**VAR klok** <identificador>;                      Tiempo máximo: 4.294.967 seg. ≈ 49,7 días  
**ClkReset, ClkStart, ClkStop, y ClkRead**

### Ejemplo:

**VAR klok** reloj;  
**ClkReset** reloj;



# Lenguaje RAPID

---

## Tipos de Datos

### ■ **confdata** ( Estructura )

Permite definir las configuraciones de los ejes del robot.

Componentes:

**cf1:** Cuadrante utilizado del eje 1.

**cf4:** Cuadrante utilizado del eje 4.

**cf6:** Cuadrante utilizado del eje 6.

**cfx:** No utilizado.

**Ejemplo:**

**VAR confdata** conf15:=[1,-1,0,0]

### ■ **dionum**

Se utiliza en combinación con instrucciones y funciones que manipulan señales de entrada y salida digitales.

**Ejemplo:**

**CONST dionum** cerrado := 1;

**SetDO** pinza1, cerrado;

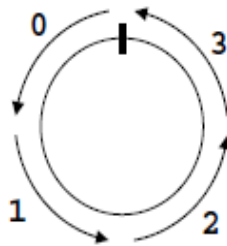


# Lenguaje RAPID

---

## Especificación de punto a alcanzar: definición de la configuración de los ejes (estructura confdata)

Para cada uno de los ejes 1, 4 y 6 se indica el cuadrante en el que se debe encontrar situado (ángulo girado desde 0°):

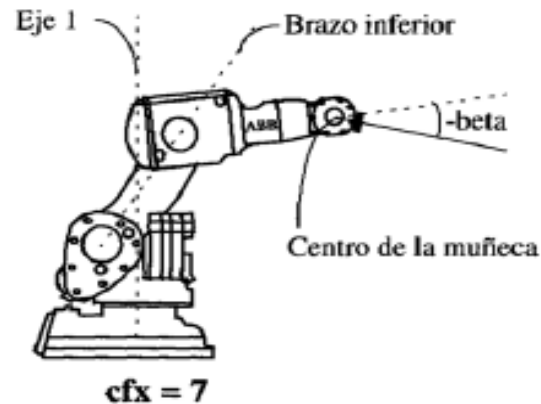
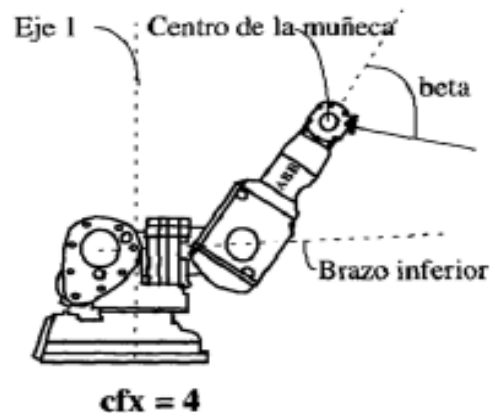
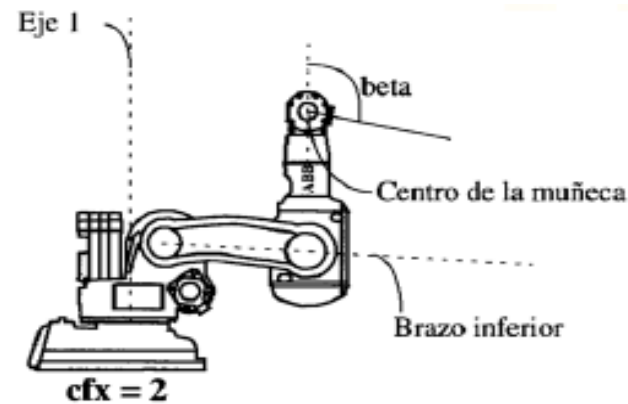
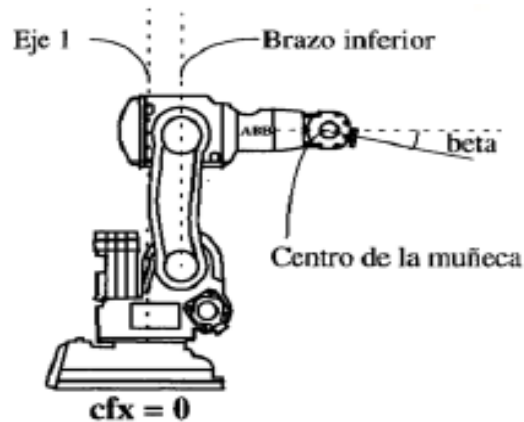


```
CONST confdata conf_1 := [0, 0, -1, 1];
```

La instrucción anterior indica una configuración 0 para el eje 1, 0 para el eje 4 y -1 para el eje 6. El último dato no se utiliza en el IRB-140.

---

## Especificación de punto a alcanzar: definición de la configuración de los ejes (estructura confdata)





# Lenguaje RAPID

---

## Tipos de Datos

### ■ **errnum**

Utilizada para describir errores recuperables durante la ejecución de un programa, para que pueda ser procesado por el gestor de errores.

**ERRNO:** Variable del sistema con valores del sistema.

**RAISE :** Instrucción que permite crear un error desde dentro del programa.

#### Ejemplo:

```
CONST errnum err_maq1:=1;  
IF Dinput(di1) = 0 RAISE err_maq1;
```

### ■ **extjoint**

Permite definir posiciones de los ejes externos, de los posicionadores o de los manipuladores de piezas de trabajo.

### ■ **Intnum**

Sirve para identificar una interrupción.

```
VAR error_transp;
```

```
CONNECT error_transp WITH correg_transp;
```

```
ISignalDI di1, 1, error_transp;
```





# Lenguaje RAPID

---

## Tipos de Datos

### ▪Iodev

Se utiliza para los canales serie, como impresores y archivos.

### ▪loaddata

Sirve para describir las cargas instaladas en la brida de montaje del robot.

### ▪mecunit

Sirve para definir las diferentes unidades mecánicas que pueden ser controladas y a las que se pueden acceder desde el robot y desde el programa.

### ▪num

Valores numéricos

**Ejemplo:**

```
VAR num a;  
a:=5;
```

### ▪orient

~~Define orientación de herramientas y rotaciones de ejes, en forma de cuaternios (q1, q2, q3, q4 ).~~



# Lenguaje RAPID

---

## Tipos de Datos

### ▪ **pos** ( Estructura )

Representar posiciones sólo x, y y z en milímetros.

**Ejemplo:**

```
VAR pos p1;  
p1 := [500,0,940];  
p1.y := p1.y + 50;
```

### ▪ **pose**

Se utiliza para cambiar de un sistema de coordenadas a otro.

Componentes

<b>trans</b>	Desplazamiento en la posición (x,y,z)
<b>rot</b>	Rotación del sistema de coordenadas (cuaternios).

**Ejemplo:**

```
VAR pose base1;  
base1.trans := [ 50, 0, 40 ];  
base1.rot   := [ 1, 0, 0, 0 ];
```



# Lenguaje RAPID

---

## Tipos de Datos

### ▪ **robtarget** ( Estructura )

Sirve para definir la posición del robot.

Componentes:

**trans** : Posiciones ( x, y, z )  
**rot** : Orientación de la herramienta.  
**robconf** : Configuración de los ejes.  
**extax** : posición de los ejes externos.

Ejemplo:

```
CONST robtarget p1;  
p1:=[100,0,200],[1,0,0,0],[1,1,0,0],[0,0,0,0,0,0];
```

### ▪ **string**

Ejemplo:

```
VAR string text;  
text:= "Arranque del sistema";
```



# Lenguaje RAPID

---

## Tipos de Datos

### ▪ **tooldata** ( Estructura )

Describe las características de una herramienta.

Componentes:

**robhold** : Tipo bool que define si el robot sujeta la herramienta o no.

**tframe** : Sistema de coordenadas de la herramienta

Posición del TCP (x,y,z)

Orinetación. (q1,q2,q3,q4)

**tload** : Carga de la herramienta

Peso

Centro de gravedad (x,y,z)

Ejes de momento de la herramienta

(q1,q2,q3,q4)

Momento de inercia de los ejes (x,y,z).



# Lenguaje RAPID

---

## Tipos de Datos

### ▪ **speeddata** ( Estructura )

Para especificar la velocidad a la que se moverán los ejes externos del robot.

Componentes

**v\_tcp** : Velocidad del TCP mm/s.

**v\_ori** : Velocidad de reorientación del TCP grados/s.

**v\_leax** : Velocidad de los ejes externos lineales en mm/s.

**v\_reax** : Velocidad de los ejes externos rotativos grados/s.

**Ejemplo:**

**VAR speeddata** vmedia:=[1000, 30, 200, 15];

Datos predefinidos:

v5 [5,5,50,5]

v10,v20,.....,v80

v100, v150, v200, v300,.....,v800

v1000, v1500, v2000, v2500, v3000

vmax [5000, 500, 5000, 500]



# Lenguaje RAPID

---

## Tipos de Datos

### ■ **zonedata** ( Estructura )

Para especificar como debe terminarse una posición.

Componentes

**finep** : Punto de paro o de paso bool  
**pzone\_tcp** : Radio de zona del TCP en mm.  
**pzone\_ori** : Tamaño de zona de reorientación mm.  
**pzone\_eax** : Zona de los ejes externos mm.  
**zone\_ori** : Tamaño de la zona de reorientación grados.  
**zone\_leax** : Tamaño de la zona ejes externos mm.  
**zone\_reax** : Tamaño zona ejes rotativos externos grados.

**Ejemplo:**

**VAR zonedata** trayec:=[FALSE, 25, 40, 50, 5, 35 10];

Datos predefinidos:

z1 [1, 1, 1, 0.1, 1, 0.1]  
z5, z10, z15, z20, .....,z100  
z150  
z200 [ 200, 300, 300, 30, 300, 30 ]



# Lenguaje RAPID

---

## Instrucciones

### ▪ Cambio de valor de una salida

**Reset** <salida digital>

Activación = 1

Desactivación = 0

**Set** <salida digital>

**SetDO** do1, 1

### ▪ Condición de espera

**WaitDI** di, 1

Esperar hasta que se active una señal digital

**WaitTime** 0.5

Esperar cierto tiempo

**WhileUntil**

Esperar hasta que se cumpla cierta condición



# Lenguaje RAPID

---

## Control de Flujo

### ▪ Compact IF

Ejecutar una instrucción sólo si se cumple una condición.

**IF** <condición> Instrucción;

### ▪ IF

Diferentes instrucciones se ejecutan si se cumple la condición

**IF** <condición> **THEN**  
Instrucciones;

**ELSE**  
Instrucciones;

**ENDIF**

### ▪ FOR

**FOR** <contador> **FROM** VI **TO** VF [**STEP** Incremento] **DO**  
Instrucciones;  
**ENDFOR**

*Ejemplo:*      **FOR** reg1 **FROM** 1 **TO** total **DO**  
                                 **MoveJ** p2,v100,z10,tool0;

. . .

**ENDFOR**





# Lenguaje RAPID

---

## Control de Flujo

### ▪WHILE

```
WHILE <condición>      DO
    Instrucciones;
ENDWHILE
```

### ▪TEST

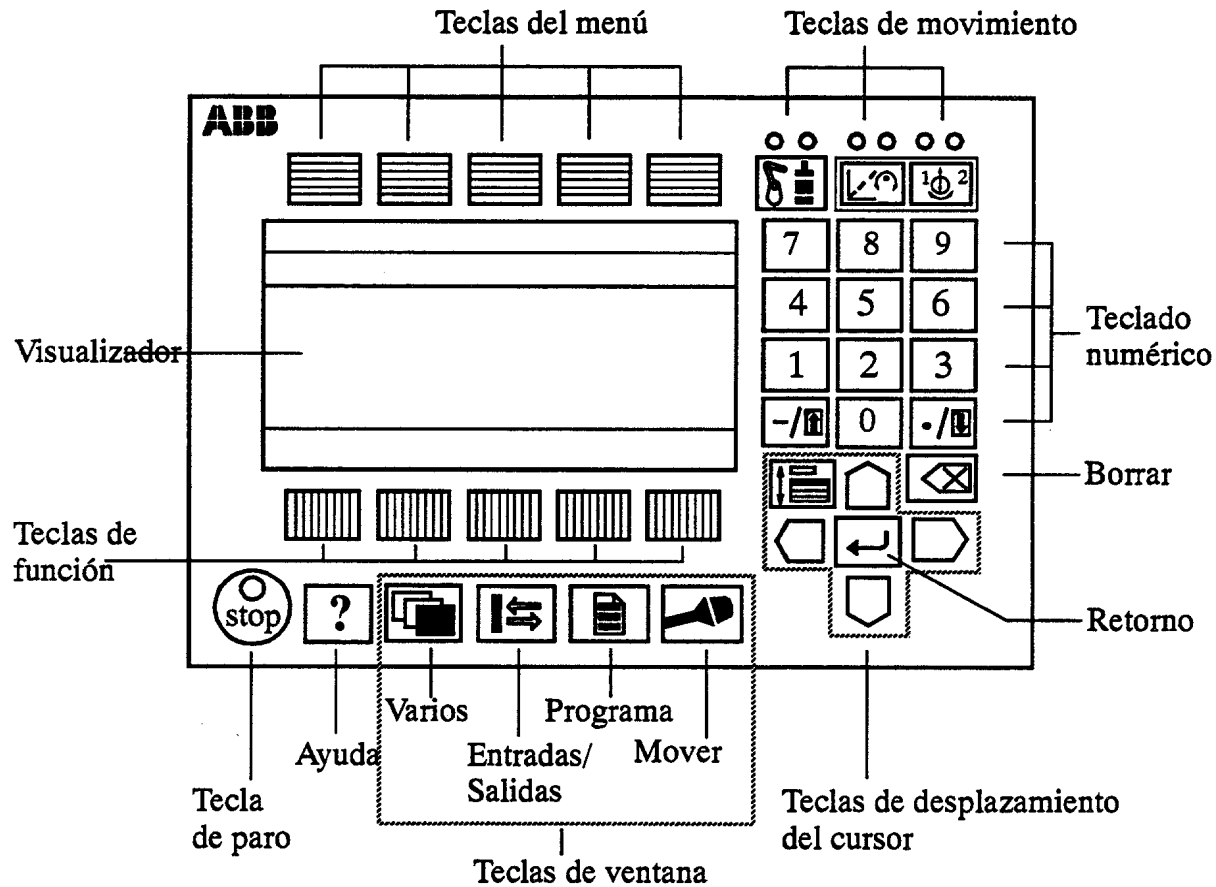
```
TEST <dato>
    CASE valor1, valor2,..., valor(n-1):
        rutina1;
    CASE valor n:
        rutinax;
    DEFAULT
        instrucciones;
ENDTEST
```

### ▪GOTO

```
GOTO Etiqueta
```

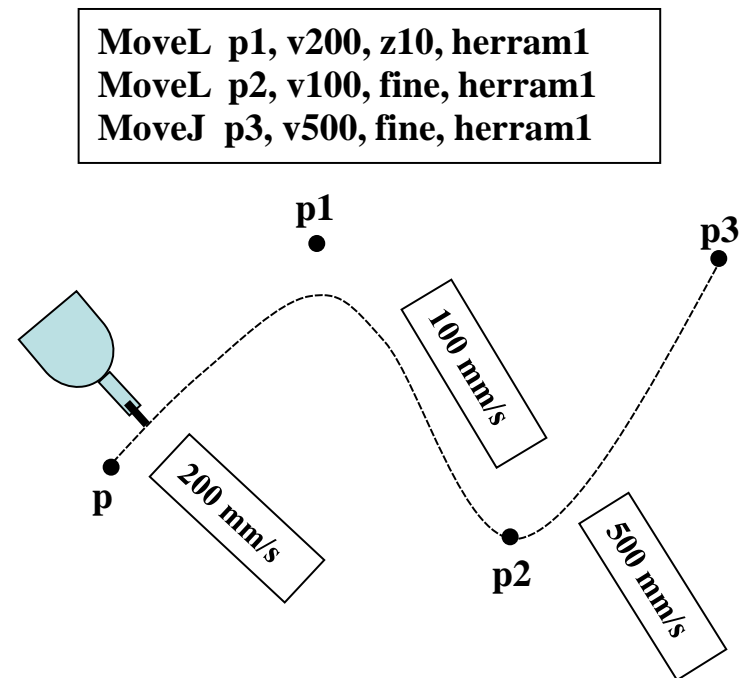
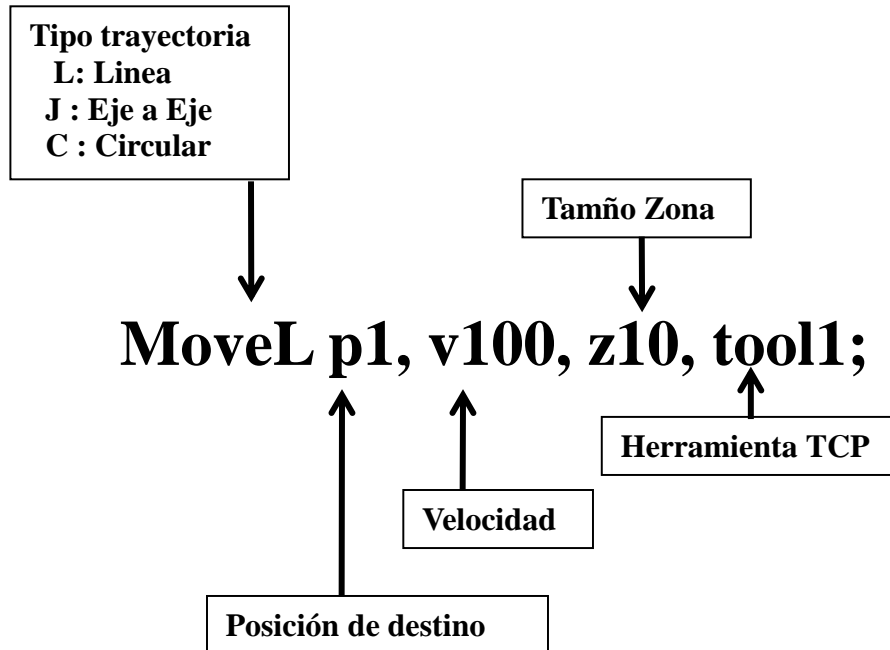
# Lenguaje RAPID

## Tabla de programación



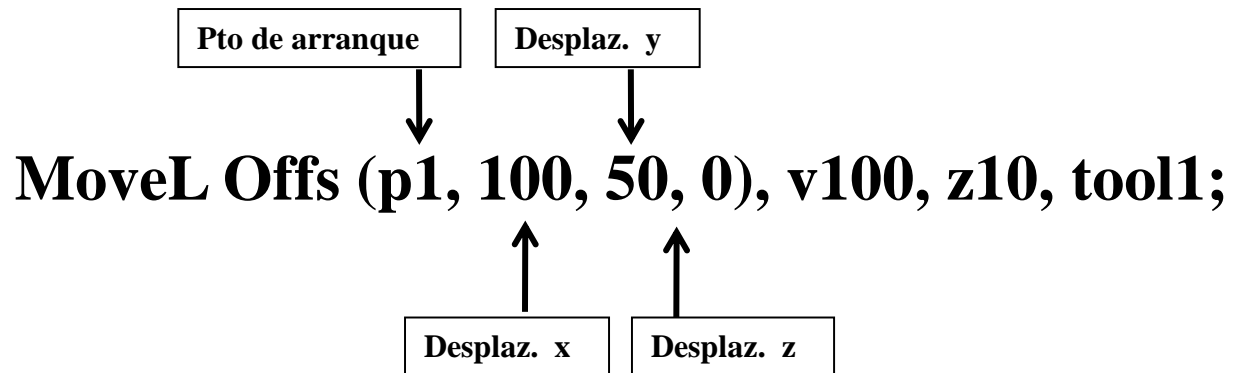
## Instrucciones

### ■ Posicionamiento

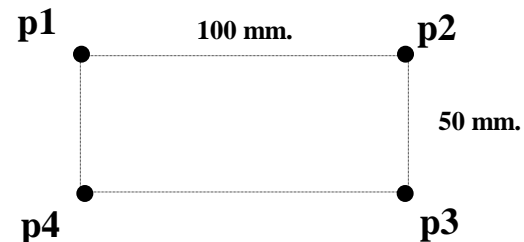


## Instrucciones

### ■ Posicionamiento (Programación con desplazamiento)



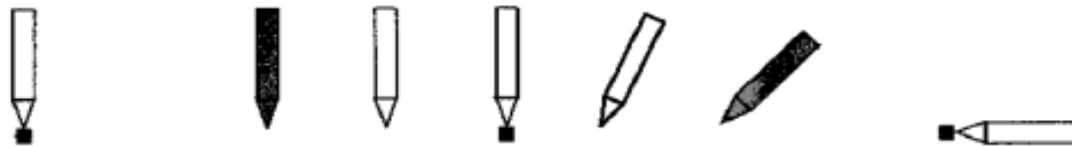
```
MoveL p1, v200, fine, herram1;  
MoveL Offs (p1, 100, 0, 0 ), v100, fine, herram1;  
MoveL Offs (p1, 100, 50, 0 ), v100, fine, herram1;  
MoveL Offs (p1, 0, 50, 0 ), v100, fine, herram1;  
MoveL p1, v100, fine, herram1;
```



## Ejecución de una trayectoria: diferencia entre puntos destino y de paso



*Figura 2 Tres posiciones han sido programadas; la última con una orientación de herramienta diferente.*

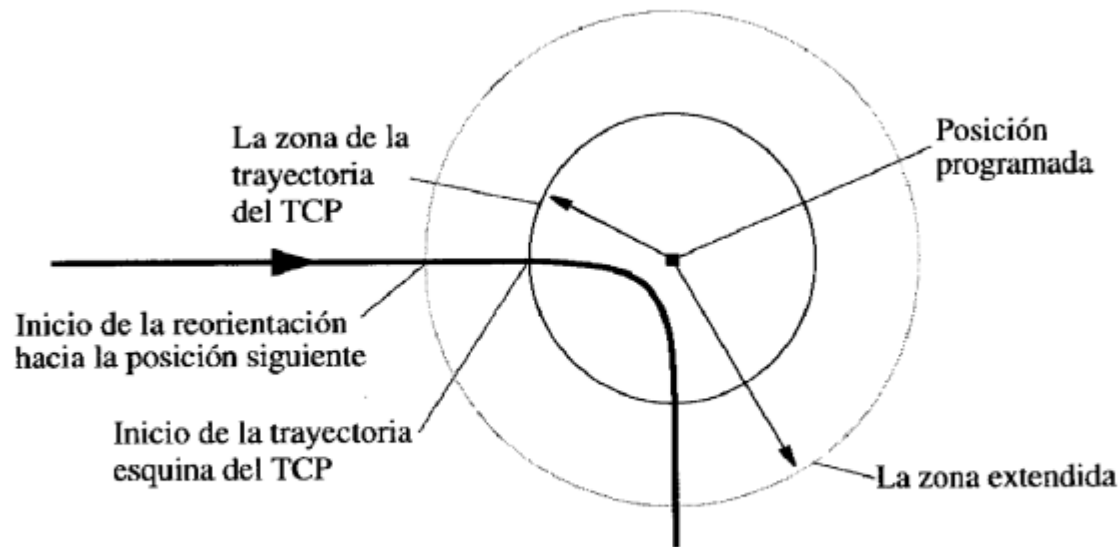


*Figura 3 Si todas las posiciones fueran puntos de paro, la ejecución del programa tendría este aspecto.*



*Figura 4 Si la posición del medio fuera un punto de paso, la ejecución del programa tendría este aspecto*

## Ejecución de una trayectoria: definición de la zona de precisión para un punto de paso





# Lenguaje RAPID

## Instrucciones de control del movimiento

- **Velocidad:** Cambia las velocidades programadas en todas las instrucciones de posicionamiento que vienen a continuación hasta que aparezca un nuevo VelSet.

### VelSet Ajuste, Máximo;

Ajuste(num): Porcentaje de la velocidad programada a la que se ejecutará la instrucción de movimiento.

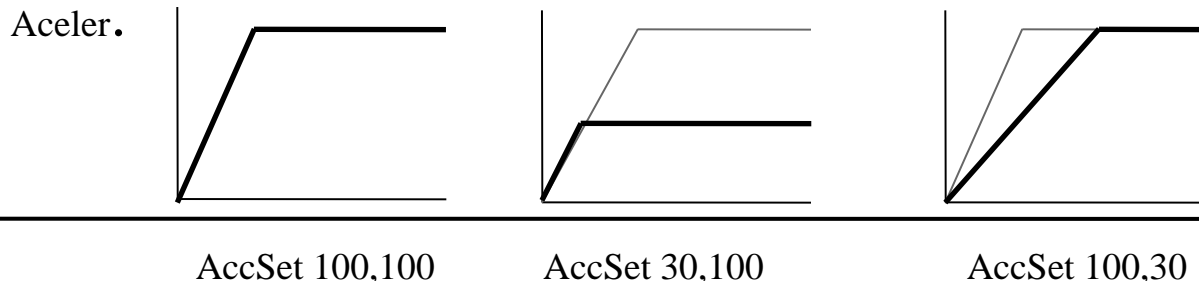
Máximo(num): Velocidad máxima del TCP en mm/s.

- **Aceleración:** Permite trabajar con aceleraciones y deceleraciones más lentas, para la realización de movimientos suaves con el robot. El robot por defecto se mueve con la máxima aceleración posible.

### AccSet Ajuste, Rampa;

Ajuste(num): Porcentaje de la aceleración máxima para todas las instrucciones de movimiento que vienen a continuación hasta un nuevo AccSet.

Rampa(num): Porcentaje de la rampa de aceleración máxima para todas las instrucciones de movimiento que vienen a continuación hasta un nuevo AccSet.





# Lenguaje RAPID

---

## Funciones de interés:

### Offs

Sirve para añadir un offset a una posición determinada del robot.

```
Offs (Punto, offsetX, offsetY, offsetZ)
```

```
Punto      robtarget
```

```
offsetX    num
```

```
offsetY    num
```

```
offsetZ    num
```

Ej.

```
MoveL p1,v100,fine,tool1;
```

```
MoveL offs(p1,100,0,0),v100,fine,tool1;
```

```
MoveJ offs(p1,100,100,0,0),v100,fine,tool1;
```

```
MoveL offs(p1,0,100,0),v100,fine,tool1;
```

```
p1:=offs(p1,100,0,0);
```

Para introducir un desplazamiento, relativo a un punto ya definido en el prog.





# Lenguaje RAPID

---

## Instrucciones

### Instrucciones de entrada/salida

- **Set señal;**  
Sirve para colocar el valor de la señal de la salida digital a uno.  
Ej. Set do1;
- **Reset señal;**  
Sirve para poner una señal de salida digital a cero.  
Ej. Reset do1;
- **SetDO señal, valor;**  
Sirve para cambiar el valor de una señal de salida digital.  
Ej: SetDO do1,1;



## Instrucciones

### Instrucciones de espera

A la hora de programar movimientos con el robot en muchos casos hay que combinarlos con esperas para conseguir una sincronización, por ejemplo en tareas de ensamblado. Las instrucciones de espera más habituales son:

**1.WaitTime [NPos] Tiempo;**

**Ej.: WaitTime 2;**

**1.WaitUntil [NPos] condición o expresión lógica [MaxTime] [TimeFlag]**

**1.WaitDI señal digital, valor;**



# Lenguaje RAPID

---

## Instrucciones

### Instrucciones de búsqueda

#### **SearchL Señal,PuntBusc,AlPunto,Veloc,Herram**

Busca una posición moviendo el TCP de forma lineal. Un cambio en la señal guarda la posición actual en puntbusc, que es un dato tipo robtarget. Se recomienda usar sen1, sen2, sen3 que son entradas de usuario a la tarjeta del sistema.

#### **SearchC Señal,PuntBusc,Puntcirc,AlPunto,Veloc,Herram**

Busca una posición moviendo el TCP de forma circular.

Ej.

```
SearchL sen1,pb,p10,p20,v100,tool1;  
MoveL pb,v100,fine,tool1;
```



# Lenguaje RAPID

---

**Funciones** para consultar el estado de entradas/salidas digitales

## **DInput**

Lectura del valor de una señal de entrada digital. El valor que devuelve es 0 ó 1 (el dato es de tipo `dionum`).

Ej.

```
IF DInput(di1) = 1 THEN  
    . . .
```

## **DOutput**

Lectura del valor de una señal de salida digital. El valor que devuelve es 0 ó 1. Este dato que devuelve es de tipo `dionum`.



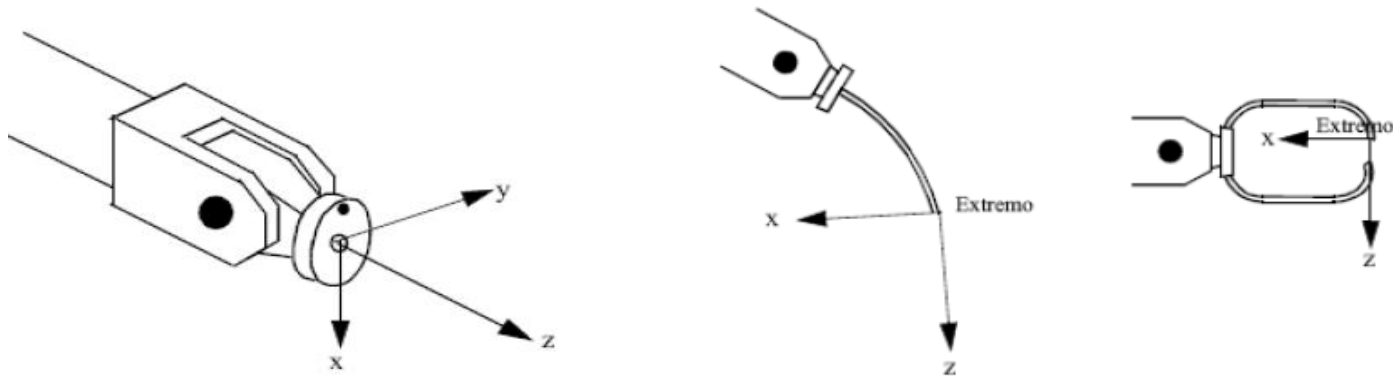
# Lenguaje RAPID

---

```
%%%  
  VERSION:1  
  LANGUAGE:ENGLISH  
%%%  
MODULE EJEMPLO1  
  PERS tooldata pinza:=[TRUE,[[0,0,225],[1,0,0,0]],[2,[0,0,-  
110],[1,0,0,0],0,0,0]];  
  CONST robtarget p2:=[[855.14,-146.00999,1509.56],  
[0.421773,-0.475939,0.382936,-0.670037],[-1,-1,2,0],  
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];  
  CONST robtarget p1:=[[849.61,175.55,1510.07],  
[0.577642,-0.494359,0.355695,-0.543526],[0,0,2,0],  
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];  
  PROC main()  
    MoveJ p1,v2500,fine,tool0;  
    WaitTime 2;  
    Set dpinza;  
    MoveL p2,v1000,fine,pinza;  
    WaitTime 3;  
    Reset dpinza;  
  ENDPROC  
ENDMODULE
```

## Definición de una herramienta

Se puede definir manualmente creando una nueva variable del tipo **tooldata**, tecleando los valores de cada uno de sus componentes, o declarando una variable de ese tipo en el programa y asignándole valores. Tener en cuenta que la herramienta se define respecto al sistema de coordenadas de muñeca (tool0), que se indica a continuación:



Se puede definir utilizando el robot para cambiar el TCP y la orientación de una herramienta. En vez de entrar a teclear los valores de las componentes, como en el caso anterior, lo que se hace es seleccionar la función Especial: DefinirCoord.

Podremos definir el nuevo TCP situando el extremo de la herramienta entre 4 y 6 veces en un mismo punto del espacio, tal como se indica en el capítulo 7 de la Guía del Usuario.



## Actividades prácticas : Repaso conceptos

---

**Definir** como **herramienta** el soporte para el bolígrafo. Tomar su longitud como 20.7 cm y su peso como 400 gramos. Suponer que el centro de gravedad esta en el centro.

- Probar a reorientar tanto la herramienta tool0 (muñeca del robot) como la nueva herramienta. Comprobar que punto permanece fijo en el espacio en cada caso (el TCP).

EJEMPLO:

```
PERS tooldata herramienta := [FALSE, [[97,0,223],[0,924,0,0,383,0]], [5,  
[-23,0,75],[1,0,0,0],0,0,0]];  
PERS loaddata carga := [5,[50,0,50],[1,0,0,0],0,0,0];
```

- Realizar un programa que se mueva hasta un punto cualquiera P1. Moverse a P1 utilizando tool0 e inmediatamente después moverse al mismo punto utilizando la nueva herramienta. ¿Qué ocurre?
- **Definir** como **herramienta** la punta del bolígrafo, utilizando el método de los 4 ptos.
- Instalar las pinzas neumáticas. Realizar un programa que espere la señal de inicio mediante la entrada di1, recoja una pieza y la lleve a una posición separada cualquiera. Esperar allí 3 segundos, devolver la pieza a su sitio y regresar a la posición de inicio. Cuando alcance el punto final que activar una salida digital durante 5 segundos.

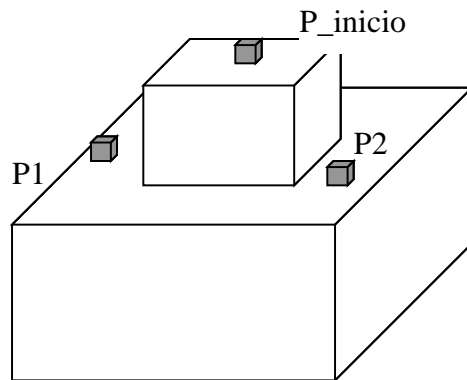
# Actividades a realizar: Programación condicional.

Se propone el siguiente ejercicio;

El robot comienza a moverse cuando se activa la entrada digital número 1.

En función de los valores de las entradas digitales 2 y 3 realizará cada una de las siguientes tareas:

di2	di3	tarea
0	0	ir a la posición de reposo
1	0	coger la pieza de la posición de reposo y dejarla en la posición 1
0	1	coger la pieza de la posición de reposo y dejarla en la posición 2
1	1	coger la pieza de la última posición en que la dejó y llevarla a la posición de inicio



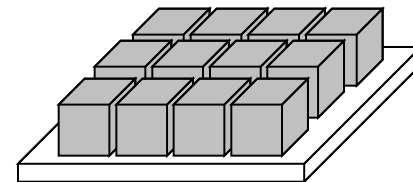


## EJERCICIOS DE PROGRAMACIÓN

- FLUJO EJECUCIÓN

En el primer ejercicio el robot cogerá la pieza y la depositará en el mismo sitio pero girada 180 grados sobre el eje z. Repetirá este proceso 10 veces o hasta que se active la entrada digital 1.

En el segundo ejemplo se simulará una tarea de paletizado, colocando las piezas en una matriz de 3 por 4 elementos como se muestra en la figura.



Para ello se definirá un punto y se calcularán los demás con desplazamientos. Para completar la tarea se emplearán bucles `for` anidados.