

UNIVERSIDAD CARLOS III DE MADRID

Departamento de Ingeniería Eléctrica Electrónica y Automática
Área de Ingeniería de Sistemas y Automática



ROBÓTICA INDUSTRIAL

GUIONES DE PRÁCTICAS
2010-2011

ÍNDICE

Introuducción.

Lenguajes de Programación de Robots:
VAL II, ARLA y RAPID

Práctica 1

Los robots ABB IRB1600 y IRB2400:
Funcionamiento básico

Práctica 2

Programación en RAPID

Lenguajes de Programación de Robots: VAL II, ARLA y RAPID



1 INTRODUCCIÓN

Hoy por hoy, los robots industriales que se pueden encontrar en el mercado únicamente admiten la programación por guiado y la programación textual orientada a robot. Otros tipos de programación, como la programación orientada a tarea, aun no han salido de los laboratorios de investigación.

El lenguaje de programación de un robot industrial es un factor que influye directamente en el tipo de tareas que este puede llevar a cabo. En consecuencia, aspectos del lenguaje tales como el manejo de las entradas/salidas, la capacidad de comunicación con dispositivos externos, la disponibilidad de operadores aritméticos o posibilidades multitarea, determinan que un robot sea mas apropiado para cierta instalación industrial o para el desarrollo de una aplicación avanzada. En esta práctica se pretende ofrecer una visión general de los lenguajes de programación de robots industriales más importantes.

Un problema que encuentra el programador de robots es que existen casi tantos lenguajes como marcas de robots. Aunque no existen lenguajes de propósito general, si hubiese que seleccionar varios lenguajes representativos, estos serían **VAL II**, **ARLA** y **RAPID**. La potencia del lenguaje VAL II es la razón principal de que los robots STÄUBLI-UNIMATION (www.staublirobotics.com) sean los que tienen más implantación en los centros de investigación en robótica. En el mundo industrial europeo predominan el lenguaje ARLA y su sucesor, el RAPID, utilizados por los robots ABB (www.abb.es). Las principales virtudes del lenguaje ARLA son la simplicidad y la robustez, sin embargo tiene limitaciones importantes cuando se pretende afrontar aplicaciones avanzadas. Este hecho ha llevado a ABB a desarrollar el nuevo lenguaje RAPID cambiando la filosofía hacia un lenguaje estructurado de alto nivel, más abierto y potente.

2 VAL-II

VAL, que significa *Victor Assembly Language*, es un sistema de control y un lenguaje de programación diseñados expresamente para robots industriales. Fue concebido originalmente por Victor Scheinman en 1976 y adoptado por la compañía UNIMATION para sus robots de la serie PUMA.

En 1984, UNIMATION presentó el **VAL II**, una nueva versión que ampliaba las posibilidades de la anterior. La última versión hasta el momento del sistema VAL es el VAL+. Aunque aporta mejoras significativas, el juego de instrucciones básico del VAL II se conserva. En la actualidad, UNIMATION se ha transformado en STÄUBLI-UNIMATION, y los robots de la serie PUMA han sido sustituidos por la serie RX, que poseen un diseño mecánico mejorado y se programan en el lenguaje VAL+.



Figura 1.1: Robot RX170HSM lanzado por STÄUBLI-UNIMATION en 2008

2.1 El entorno VAL II

El sistema VAL tiene tres modos de funcionamiento:

- Modo comando.
- Modo editor.
- Modo ejecución.

Todo el sistema se maneja desde una consola. En **modo comando** se accede a un auténtico sistema operativo que permite configurar el robot, crear y guardar programas, y registrar posiciones del manipulador. Desde el modo comando se pasa al **modo editor** para editar textualmente, y corregir, los programas. Por último, también desde el modo comando es posible ordenar la ejecución de un programa, momento en el que el sistema pasa al **modo ejecución**.

2.2 El lenguaje VAL II

Repasemos las características más novedosas del lenguaje VAL:

- Con el programa principal del robot puede coexistir un **programa de control de proceso**. Ambos se ejecutan concurrentemente y comparten variables. La única limitación es que el programa de control de proceso no contenga instrucciones de movimiento del manipulador ni de los ejes externos. El programa de control de proceso se activa y desactiva desde el programa principal.
- El tratamiento con prioridad relativa de procesos asíncronos permite atender **interrupciones**, asociadas a determinados eventos tales como la activación de una señal externa, y **errores**.

- Existe la opción de **comunicación con un computador externo**, lo cual posibilita formas de control avanzadas: control en tiempo real de la trayectoria o cierre del bucle de control en el computador.
- El control del robot puede ser llevado a cabo desde un **fichero de comandos**.

Los movimientos efectuados por un robot bajo control del sistema VAL-II, son del tipo denominado “trayectoria continua”. Cuando se ejecuta una instrucción de movimiento, no se para la ejecución del programa, sino que esta continua paralelamente al movimiento del robot. Si mientras el robot completa un movimiento hacia un punto, llega una nueva instrucción de movimiento, el robot, en las inmediaciones del primer punto, cambiará su trayectoria con el fin de dirigirse a su nuevo destino. De este modo se consigue continuidad en la trayectoria y se evita que el robot se pare en aquellos puntos que solo sirven para prefijar el camino. Lógicamente, el robot sí deberá detenerse en aquellos puntos en los que deba de efectuar un determinado trabajo. La instrucción BREAK interrumpe la ejecución del programa hasta que el robot alcanza el último destino especificado.

Tipos de variables

El lenguaje VAL utiliza los siguientes tipos de variables:

- Variables asociadas a valores **lógicos** (ej. E/S).
- Variables **reales**, asociadas a números enteros, reales u octales.
- Variables **punto**, asociadas a posiciones del brazo. Una posición del brazo viene dada por un vector de seis componentes, que puede estar formado por las seis coordenadas articulares, o por tres coordenadas de posición mas tres ángulos de Euler del extremo del brazo.
- Variables **dimensionadas**. Son arrays de variables numéricas.

Instrucciones de posicionamiento

Hay dos formatos de variables punto. Si la posición viene dada por las coordenadas cartesianas mas la orientación, se trata de un punto normal. En cambio, si está especificada por las coordenadas articulares, es un **punto de precisión**. Un punto de precisión obliga a que la trayectoria pase por él, mientras que si el punto no es de precisión, la trayectoria pasa a una cierta distancia del mismo.

FRAME (<p1>, <p2>, <p3>, <p4>)

Define un sistema de referencia con origen en el punto $p4$, donde el eje X es paralelo a la línea que pasa por los puntos $p1$ y $p2$, y el eje Y es perpendicular al eje X y pasa por $p3$.

HERE <p> | HERE #<pp>

Asocia la posición actual del robot al punto p , o al punto de precisión pp .

SET <p>=<x>, <y>, <z>, <o>, <a>, <t> | SET <pp>=<pp1>

Asigna a la variable de posición p (o pp) los valores indicados, donde $pp1$ es un punto creado en modo comando.

SHIFT <p> BY <dx>, <dy>, <dz>

Suma a las coordenadas cartesianas de p , los desplazamientos indicados.

TOOL <trans>

Define una nueva posición y orientación para el TCP expresadas mediante una transformación.

Instrucciones de control del elemento terminal

CLOSE | CLOSEI

Cierra la garra durante un movimiento del brazo, o a la finalización del mismo.

GRASP <distancia>

Cierra una pinza servocontrolada hasta la distancia indicada.

OPEN | OPENI

Abre la pinza durante un movimiento del brazo, o a la finalización del mismo.

RELAX | RELAXI

Desactiva el accionamiento de la pinza.

HAND

Devuelve la distancia de apertura de la pinza (en mm).

PENDANT <expresión>

Devuelve el último valor de la unidad de programación.

Instrucciones de control del movimiento del brazo

ALIGN

Alinea el eje Z de la herramienta con el eje de coordenadas de la base del robot mas próximo.

APPRO <p>, <dist> | APPROX <p>, <dist>

Mueve el extremo del brazo hasta una posición situada a una distancia del punto indicado, según el eje Z de la herramienta, en coordenadas articulares o en coordenadas rectilíneas.

DEPART <distancia> | DEPARTS <distancia>

Mueve el brazo alejándose la distancia indicada de la posición en la que se encontraba, según el eje Z de la herramienta, en coordenadas articulares o en coordenadas rectilíneas.

DRIVE <articulación>, <incremento>, <velocidad>

Mueva la articulación seleccionada un incremento a una cierta velocidad.

MOVE <pos> | MOVES <pos>

Mueve el brazo hasta la posición indicada. El movimiento puede ser articular o rectilíneo.

MOVET <pos>, <dist> | MOVEST <pos>, <dist>

Similar a la instrucción anterior, pero en este caso además abre la pinza la distancia indicada.

NEST

Lleva al brazo a una posición indicada como posición de descanso.

READY

Lleva al brazo a una posición indicada como posición de trabajo.

WEAVE <distancia>, <ciclo>, <tiempo_parada>

Superpone al movimiento programado, uno ondulatorio con la amplitud y frecuencia indicadas, parando al final del movimiento durante un tiempo.

BREAK

Desactiva la opción de trayectoria continua antes de que el robot ejecute su próximo movimiento.

COARSE | FINE {ALWAYS}

Selecciona una mayor o menor tolerancia en los movimientos. Sólo afecta a la siguiente instrucción de movimiento, salvo que se especifique la opción ALWAYS.

SPEED <velocidad> {MMPS | IPS} {ALWAYS}

Indica la velocidad de movimiento del brazo. Sólo afecta a la siguiente instrucción de movimiento, salvo que se especifique la opción ALWAYS.

Instrucciones de control de flujo

GOSUB <subrutina>

Salto a subrutina.

GOTO <etiqueta>

Salto incondicional.

HALT <mensaje>

Detiene la ejecución del programa y muestra un mensaje. El programa continúa con EXECUTE.

PAUSE <mensaje>

Detiene la ejecución del programa y muestra un mensaje. El programa continúa con PROCEED.

PRIORITY

Devuelve la prioridad del programa ejecutado. Toma valores entre 1 y 127.

RETURN <n>

Devuelve el control a la línea *n* del programa que ejecutó la subrutina.

WAIT <canal>

Detiene el programa hasta que el canal indicado adopte el estado programado.

CASE <expresión> OF [VALUE <valor> | ANY] : <instrucción> END

DO <instrucción> UNTIL <expresión>

FOR <variable> = <valor> TO <expresión> <instrucción> END

IF <operación> THEN <instrucción> ELSE <instrucción> END

WHILE <expresión> DO <instrucción> END

Instrucciones de configuración

Una misma posición y orientación del extremo del brazo, puede ser alcanzada con distintas configuraciones del mismo. En el caso de un brazo como el PUMA, pueden darse hasta ocho configuraciones.

ABOVE | BELLOW

Situa el codo arriba o abajo.

FLIP | NOFLIP

Selecciona la quinta articulación con valores positivos o negativos.

LEFTY | RIGHTY

Selecciona una configuración con las tres primeras articulaciones hacia la izquierda o hacia la derecha.

Instrucciones de manejo de entradas/salidas digitales

SIG <variable> | SIG <-variable>

Pone en estado UNO o en estado CERO una salida digital.

RESET

Pone en estado CERO todas las salidas digitales.

SIG(<variable>)

Devuelve el estado de una entrada digital.

Instrucciones de activación y desactivación de programas de control de proceso

PCEXECUTE <subprograma>

Activa un programa de control de proceso.

PCEND

Detiene la ejecución del programa de control de proceso en curso.

STOP <mensaje>

Detiene la ejecución del programa de control de proceso en curso escribiendo un mensaje. Continúa con PCEXECUTE.

Interrupciones y errores

Existe la posibilidad de realizar una monitorización de algunas de las entradas digitales del robot. Solamente las 16 primeras entradas (de la 1 hasta la 16) pueden provocar

interrupciones. También hay que tener en cuenta que el sistema sólo reacciona ante cambios de nivel que se mantengan estables al menos 30 ms.

Para ello se cuenta con la instrucción REACT, en la que se indica la entrada a monitorizar, el nombre de la rutina que debe ser ejecutada en caso de producirse el evento y, opcionalmente, una prioridad. En el caso de producirse la condición monitorizada y de que la prioridad asignada a la monitorización sea mayor que la de la rutina o programa que se está ejecutando, se espera a concluir la operación en curso y se fuerza una llamada a la subrutina indicada. Si se pretende que la llamada a la subrutina se efectúe inmediatamente, deteniendo el movimiento del robot, deberá emplearse la instrucción REACTI.

Se permite así mismo realizar una monitorización de cualquier error durante la ejecución del programa, mediante la instrucción REACTE. La prioridad de esta monitorización es máxima.

Una vez lanzada una monitorización, esta continua hasta que la condición se produce, se ejecuta una nueva instrucción REACT o REACTI al mismo evento o se ejecuta una instrucción IGNORE que cancela la monitorización. La prioridad de una determinada subrutina o programa se fija mediante la sentencia LOCK.

REACT <canal>,<subprograma> | REACTI <canal>,<subprograma>

Monitoriza el estado de la entrada indicada, y en caso de estar en UNO, transfiere el control a un subprograma.

IGNORE <canal>

Inhibe las instrucciones REACT y REACTI asociadas a la entrada especificada.

REACTE <subrutina>

Ejecuta una subrutina en caso de producirse un error.

Línea Alter

Esta capacidad corresponde al control de la trayectoria en tiempo real. Consiste en que información procedente de un sensor exterior, o de un programa de control del proceso, sea usada para modificar la trayectoria del robot, mientras se ejecuta esta. Se inicializa con la instrucción ALTER. En el caso de un sensor exterior, conectado al controlador a través de una línea serie, cada 28 ms se demanda una nueva lectura. Si la información procede de un programa de control de proceso, este deberá incluir la instrucción ALTOUT para pasar al programa principal los valores de corrección de la trayectoria. Los datos de corrección pueden ser interpretados como absolutos o incrementales y referirse al sistema de coordenadas asociado a la base del robot o al asociado a la pinza. La instrucción ALTER sirve también para transmitir hacia el exterior datos relativos a la posición actual del robot o a la posición hacia la que se dirige el robot. Un dispositivo exterior puede usar esta información para generar la próxima orden de corrección de la trayectoria.

ALTER <canal>,<modo>,<programa>,<prioridad>

Inicializa el control en tiempo real de la corrección de la trayectoria del robot. El canal puede ser interno o externo y define el modo de funcionamiento, el programa a ejecutar en caso de enviar una señal de excepción, y a prioridad de ejecución.

ALTOUT <pp>

Información de posición desde un programa exterior.

NOALTER

Finalización del modo alter.

Operadores y funciones matemáticas

+ - * /

Operadores aritméticos básicos de adición, sustracción, multiplicación y división.

AND OR NOT

Operadores lógicos básicos.

ABS(<expresión>)

Devuelve el valor absoluto de la expresión.

SIGN(<expresión>)

Devuelve el signo de la expresión.

FRACT(<expresión>)

Devuelve la parte fraccionaria de la expresión.

INT(<expresión>)

Devuelve la parte entera de la expresión.

SQRT(<expresión>)

Devuelve la raíz cuadrada de la expresión.

SQR(<expresión>)

Devuelve el cuadrado de la expresión

SIN(<expresión>)

Devuelve el seno de la expresión.

COS(<expresión>)

Devuelve el coseno de la expresión.

ATAN2(<expresión>)

Devuelve el arcotangente de la expresión.

Otras instrucciones

BASE <x>, <y>, <z>, <o>, <a>, <t>

Define una nueva base de referencia para el robot.

DELAY <tiempo>

Detiene la ejecución del programa activo el tiempo indicado.

ENABLE <switch> | DISABLE <switch>

Habilita o inhabilita los interruptores lógicos del controlador del robot.

SIGNAL <canal>

Activa o desactiva el canal especificado.

PROMPT <mensaje>, <variable>

Escribe un mensaje y asigna el valor tecleado a la variable.

TYPE <variable>

Escribe en la salida estándar el valor de la variable que se indica.

2.3 Utilidades

Comunicaciones con computador externo

El VAL-II cuenta con la posibilidad de que un computador desempeñe funciones de supervisión, manteniendo con el controlador del robot un diálogo que permite el intercambio de información u órdenes. Este diálogo sigue el protocolo DDCMP de Digital.

2.4 Ejemplo de programación

El robot se encuentra esperando en un punto de reposo (punto A) la activación de la entrada digital 1. En cuanto la entrada digital 1 pasa al estado UNO, el robot se aproxima al punto B para coger la pieza. El robot lleva la pieza al punto C pasando por una zona segura. El ciclo finaliza en el punto de reposo.

```
PROGRAM EJEMPLO
10 OPENI
20 SPEED 100 MMPS ALWAYS
30 MOVE #A
40 WAIT SIG(1001)
50 SPEED 80 MMPS
60 APPRO B,50
70 MOVES #B
80 BREAK
90 CLOSE
100 SPEED 80 MMPS
110 DEPARTS 50
120 MOVE D
130 SPEED 80 MMPS
140 APPRO C,50
150 MOVES #C
160 OPENI
170 DEPARTS 50
END
```

3 ARLA Y EL SISTEMA 3

La compañía ABB (www.abb.es), antigua ASEA, líder en suministro de robots industriales y sistemas modulares de fabricación y servicio, adoptó como filosofía para la programación de sus robots de la serie IRB con controlador S3 (Sistema 3), un método por guiado extendido que ofrece facilidad de programación y un juego de instrucciones suficiente. Se trata del lenguaje **ARLA**.

El lenguaje ARLA ha sido durante muchos años el mas común en las factorías españolas dada la amplia implantación de los robots ABB. La aparición del lenguaje RAPID para el controlador S4 (Sistema 4) y posteriormente para los actuales controladores S5, supone un cambio radical en la programación de los nuevos robot ABB. Sin embargo, por su robustez, el lenguaje ARLA va a seguir estando presente durante mucho tiempo en nuestras plantas.

3.1 Edición

Los programas ARLA en el *Sistema 3* pueden ser editados de dos formas:

Edición asistida

La herramienta fundamental en esta forma de edición de un programa ARLA es la unidad de programación o *Teach-Pendant*. La unidad de programación dispone básicamente de una botonera, un display y un joystick. En el display aparecen, en cada momento, las instrucciones activas para la edición. Así, se puede hablar de un tipo de edición “asistida”. El joystick sirve para mover el manipulador; no olvidemos que la metodología de programación es “por guiado”.

Edición off-line

El programa es editado en un computador y volcado al robot a través de una línea serie (*Computer Link*).

3.2 Juego de instrucciones

Instrucciones de movimiento

```
POS {V = <vel> %} {FINE | LARGE ZERO ZONE | SMALL ZERO ZONE} {WEAVE
PROG <n> V = <vel> %} {REFPOINT [ON | OFF]} {LOCATION <nloc> {OFFSET X
= <offx> Y = <offy> Z = <offz>}} {POSITION <npos>}
```

La instrucción POS se edita en línea con el robot (*on-line*). Genera un movimiento a la posición en la que se encontraba el TCP del robot en el momento de la edición. Estas son las funciones de cada argumento opcional:

```
{V = <vel> %}
```

Especifica la velocidad en porcentaje de la velocidad base.

```
{FINE | LARGE ZERO ZONE | SMALL ZERO ZONE}
```

Establece la zona de precisión. Los puntos de parada tienen precisión FINE, el resto son puntos de paso.

```
{WEAVE PROG <n> V = <vel> %}
```

Superpone un movimiento oscilatorio definido en el programa *n* (programa de weaving). Se utiliza en aplicaciones de soldadura por arco.

```
{REFPOINT [ON | OFF]}
```

Activa o desactiva la utilización de un punto de referencia como nuevo origen de coordenadas. Este punto será el ocupado por el TCP en el momento de la edición de esta instrucción.

```
{LOCATION <nloc> {OFFSET X = <offx> Y = <offy> Z = <offz>}}
```

Establece el punto de destino del movimiento el guardado en el registro de tipo LOCATION *nloc*, con un offset opcional. Los registros de este tipo sólo contienen las coordenadas (*x* y *z*) del punto. El movimiento a este punto conserva la orientación.

```
{POSITION <npos>}
```

Establece el punto de destino del movimiento el guardado en el registro de tipo POSITION *npos*. Los registros de este tipo contienen las coordenadas del punto, la orientación, y la posición de los ejes externos.

Instrucciones de control del movimiento

```
[RECT | ROBOT] COORD
```

Establece el tipo de movimiento con el que se va a desplazar el TCP. Puede ser rectilíneo (RECT) o referido a los ejes del robot (ROBOT).

```
FRAME <f>
```

Establece el sistema de coordenadas cartesiano de referencia. La nueva base tiene sus ejes paralelos a la original, pero con el origen situado en la posición definida por el registro *f*.

```
TCP <ntcp>
```

Define el TCP (Punto central de la herramienta) para las siguientes instrucciones de posicionamiento.

```
V = <vel> MM/S MAX = <vel> MM/S
```

Define la velocidad base para las siguientes instrucciones de movimiento. También se define la máxima velocidad que se permitirá en los movimientos del manipulador.

Instrucciones de control de la periferia

Los robots de la serie IRB Sistema 3 disponen de los siguientes recursos para comunicarse con su entorno:

- Salidas digitales.
- Dos salidas digitales para el control de las garras (grippers).

GRIPPER [ON | OFF] {WAIT <t> S}

Cierra o abre las garras. Se puede configurar un retardo.

[SET | RESET | INVERT | PULSE] OUTP <nout>

Pone la salida digital número *nout* a 1 (SET), 0 (RESET), la invierte (INVERT), o la invierte temporalmente durante un pequeño intervalo (PULSE).

Instrucciones de control de flujo del programa

CALL PROG <n> {R <pat>} REP <nrep>

Efectúa *nrep* llamadas consecutivas a la subrutina *n* comenzando por la instrucción número *pat* (programa patrón).

[ENABLE | DISABLE] INTERRUPT

Autoriza o inhibe las interrupciones asociadas a entradas digitales. Cada entrada digital tiene una función distinta y configurable:

- Interrumpir la ejecución de la instrucción en curso y pasar a la siguiente.
- Detener el programa interrumpiendo la instrucción en curso.
- Detener el programa después de finalizar la ejecución de la instrucción en curso.
- Interrumpir la ejecución de la instrucción en curso y saltar a una subrutina.

JUMP TO <i> {[IF INP <nin> = [0 | 1]] | [IF OUTP <nout> = [0 | 1]] | [IF R <nr> [> | < | = | <>] <val>]}

Salta a la instrucción *i* incondicionalmente o condicionadamente al estado de una entrada o salida digital o al valor de un registro.

SET R <nr> TO <val>

Asigna un valor a un registro.

ADD <val> TO R <nr>

Suma un valor al contenido de un registro.

RETURN

En el final de una subrutina, devuelve el control al programa que la ha llamado. En el final del programa principal, devuelve el control a la primera instrucción.

WAIT <t> S | WAIT UNTIL INP <nin> = [0 | 1]

Demora la ejecución durante *t* segundos, o hasta que la entrada digital *nin* tome el estado indicado.

Operadores

+ - x ÷

Operadores aritméticos básicos de adición, sustracción, multiplicación y división.

Otros

COM

Introduce comentarios.

3.3 Utilidades

El controlador Sistema 3 dispone de paquetes de software opcionales para añadir nuevas prestaciones al robot en aplicaciones avanzadas. *CommTools* es una herramienta de comunicaciones con computador, *OffLine* es la herramienta que permite la edición de los programas desde un computador y *Adaptivity* es un paquete de control adicional para movimientos protegidos y acomodaticios.

CommTools

Es una colección de rutinas para comunicar el controlador del robot con un ordenador personal PC externo a través de una línea serie. Algunas de las prestaciones que ofrece esta herramienta son:

- cargar y borrar programas,
- arrancar y parar programas,
- leer y escribir en el controlador del robot datos de TCP, registros y E/S,
- interrogar sobre el estado actual del robot,
- ordenar un movimiento, etc.

OffLine

Es una herramienta software para desarrollar programas ARLA en un ordenador personal PC. Los programas así editados pueden ser fácilmente guardados en el computador y transferidos al robot en cualquier momento por una línea serie.

Adaptivity

Es un paquete software opcional del controlador del robot que hace posible movimientos protegidos y acomodaticios. Son movimientos controlados por información procedente de sensores externos. Así es posible detectar objetos, controlar la velocidad, y generar trayectorias complejas dependientes del entorno del robot.

3.4 Ejemplo de programación

El robot se encuentra esperando en un punto de reposo (punto A) la activación de la entrada digital 1. En cuanto la entrada digital 1 pasa al estado UNO, el robot se aproxima al punto B para coger la pieza. El robot lleva la pieza al punto C pasando por una zona segura. El ciclo finaliza en el punto de reposo.

```
10 FRAME 0
20 TCP 1
30 V = 100 MM/S MAX = 150 MM/S
40 ROBOT COORD
50 GRIPPER OFF
60 POS V = 100 % FINE
70 WAIT UNTIL INP 1 = 1
80 POS V = 100 % FINE
90 RECT COORD
100 POS V = 80 % FINE
110 GRIPPER ON WAIT 1 S
120 POS V = 80 % FINE
130 ROBOT COORD
140 POS V = 100 %
150 POS V = 100 % FINE
160 RECT COORD
```



```
170 POS V = 80 % FINE
180 GRIPPER OFF WAIT 1 S
190 JUMP TO 60
200 RETURN
```

4 RAPID Y LOS NUEVOS CONTROLADORES

Con la aparición del controlador S4 en 1994, ABB lanzó al mercado la cuarta generación de controladores. Este controlador aporta mejoras significativas respecto al anterior S3, principalmente en control dinámico y facilidad de uso del nuevo entorno de programación, aunque la mejora más sustancial fue la introducción del nuevo lenguaje de programación: el lenguaje RAPID. El controlador S4 incorpora las utilidades TrueMove y QuickMove que suponen un avance importante

TrueMove

Es un módulo del controlador que asegura un seguimiento fiel de la trayectoria programada, con prioridad sobre la velocidad. Si es necesario, la velocidad se autolimita.

QuickMove

Es una capacidad de auto-optimización del movimiento. Consiste en que, partiendo de datos de carga, configuración, velocidad y dirección del movimiento, el robot optimiza el servocontrol de las articulaciones para conseguir en todo momento la máxima aceleración. Si se están manipulando materiales frágiles, la aceleración se puede limitar por software.

Actualmente ABB ha desarrollado los controladores de quinta generación, la serie S5, denominados controladores IRC5. Incorporan nuevas características con su concepto modular, una unidad de interfaz portátil de diseño ergonómico totalmente nuevo, el FlexPendant y un control de robots múltiples (hasta cuatro) totalmente sincronizado por medio de la nueva función MultiMove.

- ◆ La modularidad del IRC5 supone un importante paso adelante en el control de robots con una división lógica de las funciones controladas, accionamiento de ejes y control del proceso. Esta flexibilidad hace posible optimizar la configuración de las celdas y actualizar o cambiar un módulo con la mínima repercusión sobre los otros.
- ◆ La tecnología de control de movimientos de ABB es la clave de la actuación del robot en términos de precisión, velocidad, duración del ciclo, programabilidad y sincronización con dispositivos externos.
- ◆ La precisión de recorrido de TrueMove y la brevedad del ciclo de QuickMove son aspectos fundamentales de las ventajas para el usuario. La incorporación de MultiMove refuerza el liderazgo de ABB en los sistemas de robots avanzados y su funcionalidad permite la acción coordinada de hasta cuatro robots en una celda.

MultiMove

Gracias a MultiMove es posible conectar hasta tres módulos de accionamiento adicionales, cada uno controlando un robot o un número de motores adicionales, a un controlador de armario sencillo o un módulo de control. Con un sistema MultiMove es posible usar los robots individualmente o de forma cooperativa. Algunos ejemplos de este último caso son:

- Dos robots soldando en objetos de trabajo girados por un posicionador
- Varios robots que elevan conjuntamente un objeto pesado
- Un robot que sostiene una pieza de trabajo mientras otro procesa la pieza de trabajo (normalmente, realizando soldaduras).

En general, la idea que subyace en estos cambios es la de presentar una arquitectura más abierta y utilizar hardware y software intercambiable.

RAPID

Es el lenguaje de programación del Sistema 4 y 5. Tiene las características de un lenguaje de alto nivel estructurado.

Otras novedades resaltables son:

- El entorno de programación se maneja con ventanas y menús desplegables.
- Los programas se guardan en código ASCII y en disquetes con formato DOS.
- El software del controlador se instala a partir de disquetes y se carga en RAM.

4.1 RoboStudio

RobotStudio es el software para simulación y programación off-line que la empresa ABB desarrolla para sus robots industriales. RobotStudio forma parte de una familia de productos de software que la empresa ofrece a sus clientes para mejorar su productividad y reducir costes, y que sirven de apoyo a la gestión del ciclo de vida de sus soluciones robóticas.

Con esta herramienta, ABB ofrece las ventajas generales de la programación off-line, entre las que se pueden destacar el permitir programar los robots en PCs sin parar la producción o la posibilidad de preparar los programas con antelación, para mejorar la productividad. RobotStudio pues, incorpora herramientas que repercuten en la mejora de la rentabilidad de los sistemas, permitiendo llevar a cabo planes de formación, programación y optimización de sistemas sin parar los robots.

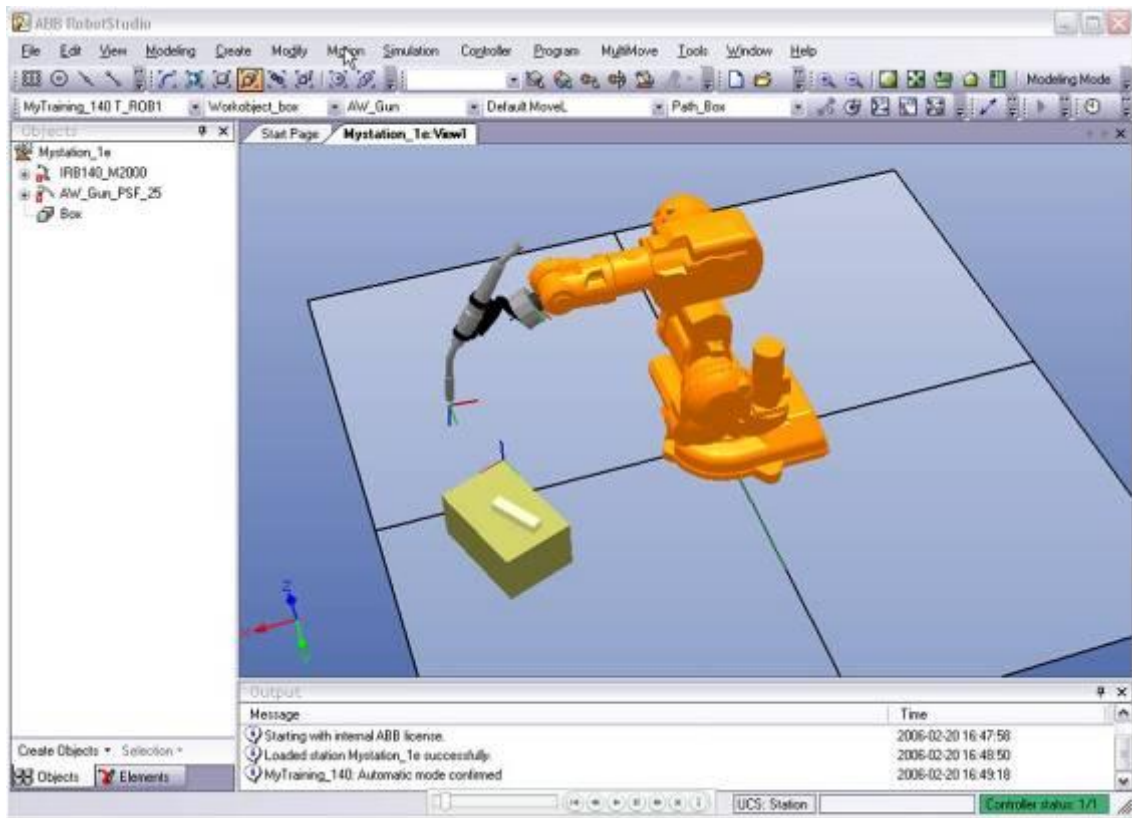


Figura 1.2: Imagen de RoboStudio

5 PROGRAMACIÓN EN RAPID

Es el lenguaje de programación utilizado en el IRC5. El nombre corresponde a *Robotics Application Programming Interactive Dialogue*. Es un lenguaje de programación textual de alto nivel y estructurado que dispone de procedimientos, funciones parametrizables, tratamiento de interrupciones, procesamiento multitarea, etc.

El RAPID es, fundamentalmente, una forma de programación textual, es decir, todo el programa y los datos se editan en formato de texto estando el robot fuera de línea (off-line). Sin embargo de este modo es muy difícil especificar las coordenadas de las posiciones que determinan la trayectoria deseada, a no ser que se calculen con la ayuda de un programa CAD. Por tanto, es habitual que las posiciones se editen “por guiado”, lo cual hace necesario que el programador mueva el robot manualmente. Así, coexisten dos modos de edición: desde una unidad de programación (*Teach-Pendant*), o bien simplemente tecleando texto en un ordenador personal.

Para completar la información que a continuación se expone del lenguaje RAPID, en Aula Global se puede descargar el manual de introducción a RAPID.

5.1 Estructura general del lenguaje

Un **programa** es un conjunto de **instrucciones** que describen la tarea del robot.

Existen instrucciones para las distintas acciones tales como movimientos, operaciones aritméticas, activación de salidas digitales, etc. Las instrucciones pueden tener una serie de **argumentos** para referirse a los objetos sobre los que actúan. Por ejemplo, en *Reset do5*, el argumento es la salida digital *do5*. Los argumentos pueden ser expresados de distintas formas:

- como un valor numérico, por ejemplo *0.5*,
- como una referencia a un dato, por ejemplo *reg1*,
- como una expresión aritmética o lógica, por ejemplo, $3*reg1+6.5$,
- como una llamada a una función, por ejemplo *Abs(reg1)*,
- como una cadena de caracteres, por ejemplo *"Pieza A terminada!"*.

Para asignar un valor a un dato se usa la instrucción “:=”

Al final de cada instrucción se escribe el caracter “;”

Con esto se delimita donde acaba cada instrucción.

Las rutinas se agrupan en **módulos**. Los módulos de un programa pueden estar distribuidos en varios ficheros, lo cual hace posible la reutilización de módulos en distintos programas.

La información se almacena en **datos**. Los datos se pueden clasificar atendiendo a varios criterios. Según el tipo de información que contienen hay varios tipos de datos:

- datos numéricos,
- datos de posición,
- datos de E/S,
- datos de herramienta,
- etc.

Según el alcance, o sea, desde qué partes del programa son accesibles,

- datos **globales** del programa,
- datos **locales** dentro de una rutina.

Según su variabilidad hay tres clases,

- datos **constantes**, que representan un valor que no varía,
- datos **variables**, a los que se les puede asignar un nuevo valor en la ejecución del programa,
- datos **persistentes**, que son variables pero su valor de inicialización se actualiza a medida que varía, incluso entre distintas ejecuciones del programa.

O según su estructura,

- datos **atómicos**, que contienen un solo dato,
- **registros**, que contienen una estructura de datos.

Además, el lenguaje incluye recursos para establecer comunicaciones de datos a través de canales serie, manejar archivos y tratar errores del sistema.

La unidad de longitud es el milímetro (mm), y la angular el grado sexagesimal (°).

5.2 Módulos

Una aplicación está dividida en un *programa* y *módulos del sistema*. El programa está también dividido a su vez en módulos (véase la figura 1.3).

Módulos del programa

Un módulo de programa puede estar formado de diferentes datos y rutinas. Cada módulo o el programa entero, podrá ser copiado en un disquete, a la memoria RAM, etc., y viceversa.

Uno de los módulos contiene el procedimiento de entrada, un procedimiento global llamado *main* (*principal*). Ejecutar el programa significa ejecutar el procedimiento principal. El programa puede incluir varios módulos, pero sólo uno de ellos contiene un procedimiento principal.

Un módulo puede, por ejemplo, definir el interfaz con el equipo externo o contener datos geométricos que han sido generados o bien por sistemas CAD o creados in situ mediante la unidad de programación. Mientras que las pequeñas instalaciones suelen estar contenidas en un módulo, las instalaciones más complejas pueden tener un módulo principal que hace referencia a rutinas y/o a datos contenidos en uno o varios otros módulos.

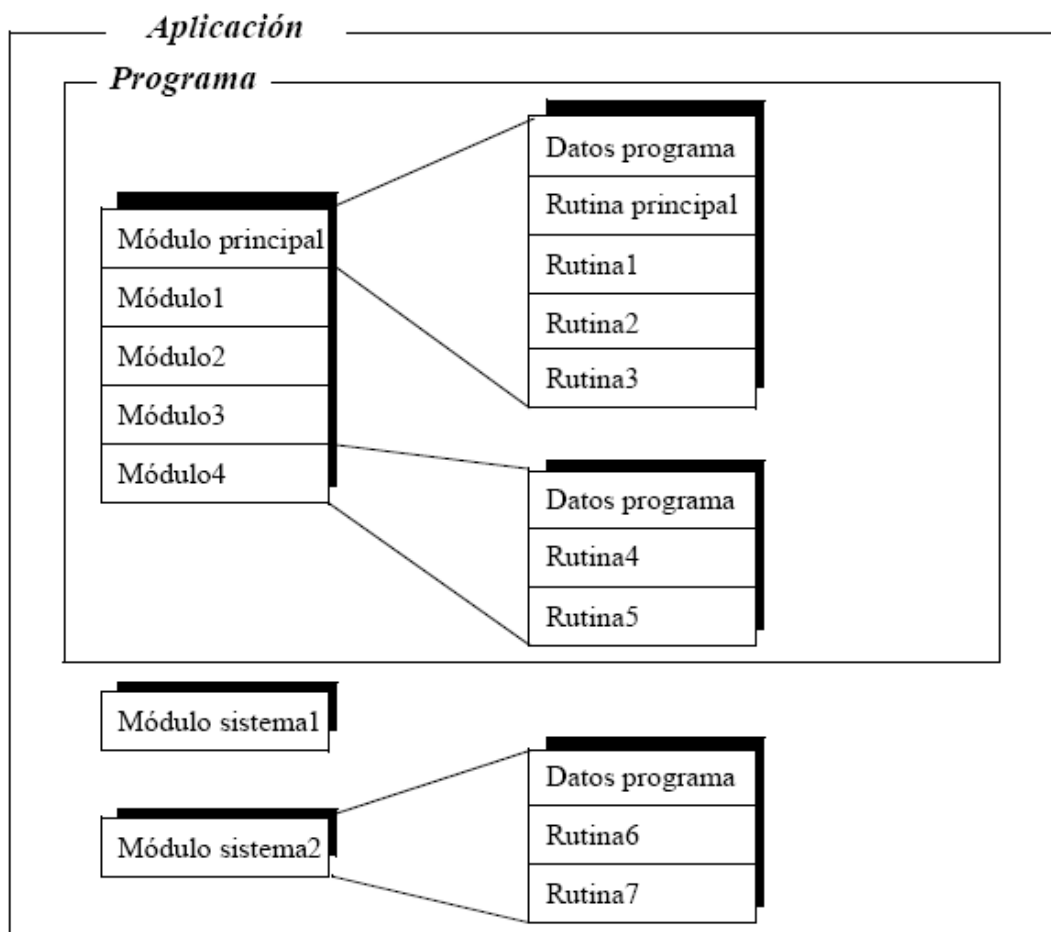


Figura1.3.- El programa está dividido en módulos

Módulos del sistema

Los módulos del sistema sirven para la definición de los datos y rutinas normales, específicos del sistema, como por ejemplo, las herramientas. No serán incluidos cuando se salva un programa, lo cual significa que cualquier actualización realizada en un módulo del sistema afectará a todos los programas que se encuentran en él, o que han sido cargados en una etapa ulterior en la memoria del programa.

Declaraciones de los módulos

Una declaración de módulo especifica el nombre y los atributos de este módulo. Dichos atributos sólo podrán añadirse de forma off-line, y no mediante la unidad de programación. A continuación se ofrecen algunos ejemplos de atributos de un módulo:

<u>Atributo</u>	<u>Si está especificado:</u>
SYSMODULE	el módulo será un módulo del sistema; de lo contrario, será un módulo del programa.
NOSTEPIN	no se podrá entrar en el módulo durante una ejecución paso a paso.
VIEWONLY	el módulo no podrá ser modificado
READONLY	el módulo no podrá ser modificado, pero sus atributos podrán ser eliminados
NOVIEW	el módulo no podrá ser visualizado, sólo ejecutado. Las rutinas globales podrán ser alcanzadas desde otros módulos y siempre son ejecutadas como NOSTEPIN. Los valores actuales de los datos globales podrán ser alcanzados desde otros módulos o desde la ventana de datos de la unidad de programación. Un módulo o un programa que contiene un módulo de programa NOVIEW no podrá ser salvado. Por esta razón, NOVIEW deberá utilizarse con prioridad para los módulos del sistema. NOVIEW sólo podrá ser definido off-line desde un PC.

por ejemplo:

```
MODULE nombre_módulo (SYSMODULE, VIEWONLY)
    !definición tipo datos
    !declaraciones datos
    !declaraciones rutinas
ENDMODULE
```

Un módulo no podrá tener el mismo nombre que otro módulo o que una rutina global o que un dato.

Sintaxis

Declaración de un módulo

```
<declaración módulo> ::=
    MODULE <nombre módulo> [ <lista atributos módulos> ]
```

```

    <lista definición tipo>
    <lista declaración datos>
    <lista declaración rutina>
ENDMODULE
<nombre módulo> ::= <identificador>
<lista atributos módulos> ::= '(' <atributos del módulo> { ',' <atributos del módulo>
} ')
<atributos del módulo> ::=
| SYSMODULE
| NOVIEW
| NOSTEPIN
| VIEWONLY
| READONLY

```

(*Nota* Si se utilizan dos o más atributos, deberán estar en el orden en que se indica anteriormente, el atributo **NOVIEW** sólo podrá ser especificado solo o junto con el atributo **SYSMODULE**.)

```

<lista definición tipo> ::= { <definición tipos> }
<lista declaración datos> ::= { <declaración de datos> }
<lista declaración rutina> ::= { <declaración de rutina> }

```

5.3 Rutinas

Existen tres tipos de rutinas (subprogramas): los procedimientos, las funciones y las rutinas de tratamiento de interrupciones.

- Los **procedimientos** no devuelven ningún valor y se utilizan en el contexto de las instrucciones.
- Las **funciones** devuelven un valor de un tipo específico y se utilizan en el contexto de las expresiones.
- Las rutinas de tratamiento de interrupciones (**Trap**) proporcionan una manera de procesar las interrupciones. Una rutina de tratamiento de interrupciones puede estar asociada a una interrupción específica y entonces, en el caso en que dicha interrupción ocurra en una etapa ulterior, se volverá a ejecutar automáticamente. No se podrá nunca llamar explícitamente una rutina de tratamiento de interrupciones desde el programa.

Siempre existirá al menos una rutina *main*, y por ella se inicia la ejecución del programa.

Alcance de las rutinas

El alcance de una rutina indica el área en la que la rutina está accesible. La directiva opcional de una declaración de una rutina, la clasifica como una rutina local (dentro del módulo), o como global.

Ejemplo: **LOCAL** PROC rutina_local (...
 PROC rutina_global (...

Las siguientes pautas referentes al alcance son aplicables a las rutinas (véase el ejemplo de la figura 1.4):

- El alcance de una rutina global puede incluir cualquier módulo.
- El alcance de una rutina local comprende el módulo en el que se encuentra.

- Dentro de su alcance, una rutina local oculta todas las rutinas globales o datos que tengan el mismo nombre.
- Dentro de su alcance, una rutina oculta las instrucciones y rutinas predefinidas o datos que tengan el mismo nombre.

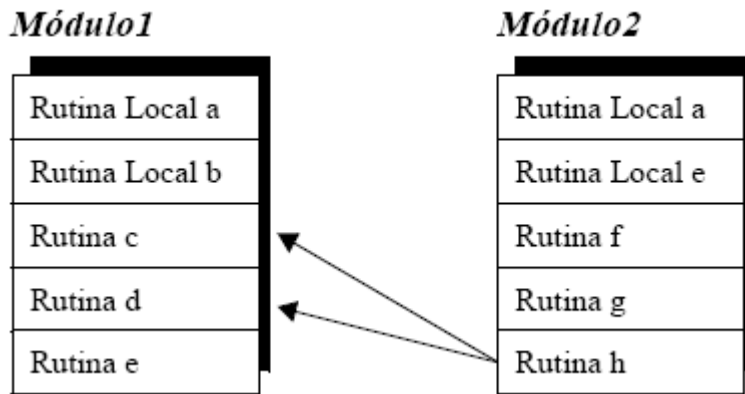


Figura 1.4.- Ejemplo: Las rutinas siguientes podrán ser llamadas desde la Rutina h: Módulo1 - Rutina c, d. Módulo2 - Todas las rutinas.

Una rutina no deberá tener el mismo nombre que otra rutina o datos dentro del mismo módulo. Una rutina global no deberá tener el mismo nombre que otra rutina global o que datos globales de otro módulo.

Parámetros

La lista de parámetros de una declaración de rutina especifica los argumentos (parámetros actuales) que deberán ser suministrados cuando se llama a una rutina.

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

- Normalmente, un parámetro suele ser utilizado únicamente como una entrada y es procesado como una variable de rutina. El cambio de esta variable no cambiará el argumento correspondiente.
- Un parámetro INOUT especifica que el argumento correspondiente debe ser una variable (entero, elemento o componente) o un entero persistente que puede ser cambiado por la rutina.
- Un parámetro VAR especifica que el argumento correspondiente debe ser una variable (entero, elemento o componente) que puede ser cambiada por la rutina.
- Un parámetro PERS especifica que el argumento correspondiente deberá ser un persistente entero que podrá ser cambiado por la rutina.

Si un parámetro INOUT, VAR o PERS es actualizado, ello significa que el argumento en sí será actualizado, es decir, que se posibilita la utilización de argumentos para devolver valores a la rutina que llama.

Ejemplo: PROC rutina1 (num par_in, INOUT num par_inout, VAR num par_var,PERS num par_pers)

Un parámetro puede ser opcional y puede ser omitido en la lista de argumentos de una llamada de rutina. Un parámetro opcional se reconoce por la barra invertida “\” que le precede.

Ejemplo: PROC rutina2 (num par_necesario \num par_opcional)

No se podrá hacer referencia a un valor de un parámetro opcional que es omitido en una llamada de rutina. Esto significa que las llamadas de rutina deberán ser comprobadas por si se encuentra algún parámetro opcional antes de poder utilizar un parámetro opcional.

Puede ocurrir que dos o más parámetros opcionales sean mutuamente exclusivos (es decir, declarados para excluirse mutuamente), y esto significa que sólo podrá haber uno de ellos presente en la llamada de rutina. Esto está indicado por una línea “|” situada entre los parámetros correspondientes.

Ejemplo: PROC rutina3 (\num excluye1 | num excluye2)

El tipo especial, *switch*, sólo podrá ser asignado a parámetros opcionales y proporciona una manera de utilizar los argumentos switch, es decir, argumentos que sólo están especificados por los nombres (y no por los valores). Un valor no podrá ser transferido a un parámetro switch. La única forma de utilizar un parámetro switch es comprobar su presencia mediante la función predefinida, *Present*.

Ejemplo: PROC rutina4 (\switch on | switch off)

```

...
IF present (off ) THEN
.
..
ENDPROC

```

Las matrices pueden utilizarse como argumentos. El grado de un argumento de matriz deberá corresponder con el grado del parámetro formal correspondiente. La dimensión de la matriz será definida por el parámetro (señalada con un “*”). Así, la dimensión actual dependerá de la dimensión del argumento correspondiente en una llamada de rutina. Una rutina podrá determinar las dimensiones actuales de un parámetro utilizando la función predefinida, *Dim*.

Ejemplo: PROC rutina5 (VAR num palet{*,*})

Final de una rutina

El final de la ejecución de un procedimiento está indicado explícitamente mediante una instrucción RETURN o bien puede estar indicado de forma implícita cuando se alcance el final (ENDPROC, BACWARD o ERROR) del procedimiento.

La evaluación de una función deberá terminarse con una instrucción RETURN.

El final de la ejecución de una rutina de tratamiento de interrupciones está indicado de forma explícita mediante la instrucción RETURN o bien, de forma implícita cuando se alcanza el final (ENDTRAP o ERROR) de esta rutina de tratamiento de interrupciones. La ejecución continuará a partir del punto en que ocurrió la interrupción.

Declaraciones de rutina

Una rutina puede comprender declaraciones de rutina (incluyendo parámetros), datos, un cuerpo de instrucciones, un gestor de ejecución hacia atrás (sólo para procedimientos) y un gestor de errores (véase la figura 1.5). Las declaraciones de rutina no podrán ser anidadas, es decir, que no se podrá declarar una rutina dentro de una rutina.

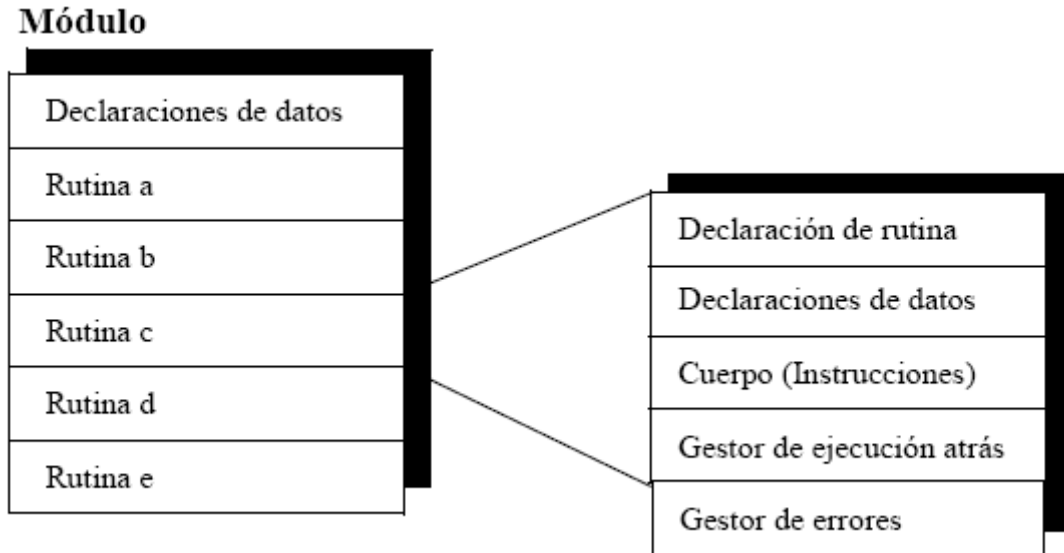


Figura 1.5 - Una rutina puede contener declaraciones, datos, un cuerpo de instrucciones, un gestor de ejecución hacia atrás y un gestor de errores.

Declaración de un procedimiento

Ejemplo: Multiplicación de todos los elementos de una matriz numérica por un factor;

```
PROC mulmatriz (VAR num matriz { * }, num factor)
  FOR índice FROM 1 TO dim( matriz, 1 ) DO
    matriz{índice } := matriz{índice } * factor;
  ENDFOR
ENDPROC
```

Declaración de una función

Una función puede devolver cualquier valor de tipo de dato, pero no un valor de matriz.

Ejemplo: Devolver la longitud de un vector;

```
FUNC num veclen (pos vector)
  RETURN Sqrt(Pow(vector.x,2)+Pow(vector.y,2)+Pow(vector. z,2));
ENDFUNC
```

Declaración de una rutina de tratamiento de interrupciones

Ejemplo: Respuesta a la interrupción de alimentador vacío;

```
TRAP alim_vacío
espera_alim;
RETURN;
ENDTRAP
```

Llamada de procedimiento

Cuando se llama un procedimiento, los argumentos que corresponden a los parámetros del procedimiento deberán utilizarse:

- Los parámetros de mandato deberán ser especificados. Además, deberán ser especificados siguiendo el orden correcto.
- Los argumentos opcionales podrán ser omitidos.
- Los argumentos condicionales podrán utilizarse para transferir los parámetros de una llamada de rutina a otra.

El nombre del procedimiento podrá ser especificado de forma fija utilizando un identificador (*early binding*) o bien podrá ser evaluado durante el tiempo de funcionamiento desde una expresión de tipo cadena (*late binding*). Incluso si el método *early binding* debe ser considerado como la forma de llamada de proceso ‘normal’, *late binding* con frecuencia proporciona un código compacto y muy eficaz. *Late binding* se define colocando el signo de porcentaje antes y después de la cadena que denota el nombre del proceso.

Ejemplo:

```

! early binding
TEST products_id
CASE 1:
proc1 x, y, z;
CASE 2:
proc2 x, y, z;
CASE 3:
....
! mismo ejemplo utilizando binding
% “proc” + NumToStr(product_id, 0) % x, y, z;
....
! mismo ejemplo otra vez utilizando otra variante de late binding
VAR string procname {3} :=[“proc1”, “proc2”, “proc3”];
....
% procname {product_id} % x, y, z;
....
```

Téngase en cuenta que *late binding* está disponible únicamente para llamadas de procedimientos, y no para llamadas de función. Si se hace referencia a un procedimiento desconocido utilizando *late binding*, la variable del sistema ERRNO se activa en ERR_REFUNKPRC. Si se hace referencia a un error de llamada de procedimiento (sintaxis, no procedimiento) utilizando *late binding*, la variable del sistema ERRNO se activa en ERR_CALLPROC.

Sintaxis

Declaración de una rutina

```

<declaración rutina> ::=
    [LOCAL] ( <declaración procedimiento>
              | <declaración función>
              | <declaración trap> )
    | <comentario>
    | <RDN>
```

Parámetros

```

<lista parámetros> ::=
    <primera declaración parámetro>{ <siguiente declaración parámetro> }
<primera declaración parámetro> ::=
    <declaración parámetro>
    | <declaración parámetro opcional>
    | <PAR>
<siguiente declaración parámetro> ::=
    ',' <declaración parámetro>
    | <declaración parámetro opcional>
    | ',' <PAR>
<declaración parámetro opcional> ::=
    '\ ' ( <declaración parámetro> | <ALT> )
    { ' ' ( <declaración parámetro> | <ALT> ) }
<declaración parámetro> ::=
    [ VAR | PERS | INOUT ] <tipo dato>
    <identificador> [ ' ' ( '*' { ' ' '*' } ) | <DIM> ] ' '
    | 'switch' <identificador>

```

Declaración de un procedimiento

```

<declaración procedimiento> ::=
    PROC <nombre procedimiento>
        '(' [ <lista parámetro> ] ')'
        <lista declaración datos>
        <lista instrucción>
        [ BACKWARD <lista instrucción> ]
        [ ERROR <lista instrucción> ]
    ENDPROC
<nombre procedimiento> ::= <identificador>
<lista declaración datos> ::= { <declaración datos> }

```

Declaración de una función

```

<declaración función> ::=
    FUNC <tipo valor dato>
        <nombre función>
        '(' [ <lista parámetro> ] ')'
        <lista declaración datos>
        <lista instrucción>
        [ ERROR <lista instrucción> ]
    ENDFUNC
<nombre función> ::= <identificador>

```

Declaración de una rutina de tratamiento de interrupciones

```

<declaración trap> ::=
    TRAP <nombre trap>

```

```

        <lista declaración datos>
        <lista instrucción>
        [ ERROR <lista instrucción> ]
ENDTRAP
<nombre trap> ::= <identificador>

```

Llamada de procedimiento

```

<llamada procedimiento> ::= <procedimiento> [ <lista argumento procedimiento> ] ';'
<procedimiento> ::
    = <identificador>
    | '%' <expresión> '%'
<lista argumento procedimiento> ::= <primer argumento procedimiento> { <argumento
procedimiento> }
    <primer argumento procedimiento> ::=
    <argumento procedimiento requerido>
    | <argumento procedimiento opcional>
    | <argumento procedimiento condicional>
    | <ARG>
<argumento procedimiento> ::=
    ',' <argumento procedimiento requerido>
    | <argumento procedimiento opcional>
    | <argumento procedimiento condicional>
    | ',' <ARG>
<argumento procedimiento requerido> ::= [ <identificador> ':' ] <expresión>
<argumento procedimiento opcional> ::= '\ ' <identificador> [ ':' <expresión> ]
<argumento procedimiento condicional> ::= '\ ' <identificador> '?' ( <parámetro> |
<VAR> )

```

5.4 Datos

Los datos se declaran del siguiente modo:

```
CLASE tipo nombre:=<valor>;
```

Ejemplo

```
VAR num reg1:=0.5;
```

Clases de datos

- **VAR**

Su valor puede variar a lo largo de la ejecución.

Ejemplo

```
VAR num total;
...
total:=total+1;
```

- **CONST**

Toman valores fijos que no varían a lo largo de la ejecución.

Ejemplo

```
CONST   robtarget      p1:=[ [888.7,778.06,1040.16],      [0.018434,-
0.07284,0.997163,0.004632],      [0,-1,0,0],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

- PERS

Su valor de inicialización se actualiza a medida que varía. No pueden ser datos de rutina. Tampoco pueden contener una expresión.

Ejemplo

```
PERS num reg1:=0;
...
reg1:=5;
(en la siguiente ejecución...)
PERS num reg1:=5,
...
reg1:=5;
```

Tipos de datos básicos

- num

Representa valores numéricos, ya sean enteros o reales.

Ejemplo

```
5  0.37  0.1E-5
```

- bool

Designa valores lógicos que pueden ser verdaderos o falsos (TRUE o FALSE).

Ejemplo

```
VAR bool abrir:=TRUE;
...
abrir:=reg1>1;
```

- string

Se usa para guardar cadenas de caracteres. El tamaño máximo es de 80 caracteres, incluidas las comillas “ ” que delimitan la cadena.

Ejemplo

```
CONST string mensaje:="Programa terminado!";
```

- dionum

Representa estados que pueden tomar las entradas y salidas digitales.

Ejemplo

```
CONST dionum cerrado:=1;
...
SetDO pinzal, cerrado;
```

- singaldi, signaldo, signalai, signalao

Sirven para referirse a cada canal de entrada/salida ya sea digital o analógico.

Otros tipos de datos

- `pos`
`<x num>`
`<y num>`
`<z num>`

Es un dato de tipo registro que sirve para guardar las coordenadas cartesianas de una posición X, Y, y Z.

- `orient`
`<q1 num>`
`<q2 num>`
`<q3 num>`
`<q3 num>`

Es un dato de tipo registro que guarda la orientación de algún elemento, por ejemplo la de una herramienta. RAPID usa la representación en cuaternios Q1, Q2, Q3 y Q4 para definir la orientación.

- `pose`
`<trans pos>`
`<rot orient>`

Se usa para cambiar de un sistema de coordenadas a otro. Está compuesto de una translación de tipo *pos* y una rotación, de tipo *orient*.

- `confdata`
`<cf1 num>`
`<cf4 num>`
`<cf6 num>`
`<cfx num>` (no se utiliza)

Sirve para definir la configuración del robot.

- `extjoint`
`<eax_a num>`
`<eax_b num>`
`<eax_c num>`
`<eax_d num>`
`<eax_e num>`
`<eax_f num>`

Se usa para definir las posiciones de los ejes externos al robot, si los hubiere.

- `robtarg`
`<trans pos>`
`<rot orient>`
`<robconf confdata>`
`<extax extjoint>`

Sirve para definir la posición del robot y de sus ejes externos.

- `speeddata`
`<v_tcp num>`
`<v_ori num>`
`<v_leax num>`
`<v_reax num>`

Especifica la velocidad a la que se moverán el robot y los ejes externos.

Datos predefinidos

v5 v10 v20 ... v60 v80 v100 v150 v200 v300 ... v600 v800 v1000 v1500
v2000 v2500 v3000 vmax

- zonedata
<finep bool>
<pzone_tcp num>
<pzone_ori num>
<pzone_eax num>
<zone_ori num>
<zone_leax num>
<zone_reax num>

Indica la zona de precisión por la que se va a pasar en los puntos intermedios de paso.

Datos predefinidos

fine z1 z5 z10 z15 z20 z30 ... z60 z80 z100 z150 z200

- loaddata
<mass num>
<cog pos>
<aom orient>
<ix num>
<iy num>
<iz num>

Describe las características de la carga que el robot pueda portar.

Datos predefinidos

load0 load1

- tooldata
<robothold bool>
<tframe pose>
<tload loaddata>

Se usa para definir las características de la herramienta final que lleva el robot.

Datos predefinidos

tool0 tool1

- jointtarget ejes_robot ejes_externos
<ejes_robot robaxes>
<ejes_externos extaxes>

Se utiliza para definir la posición a la que se moverán los ejes del robot y los ejes externos al ejecutar una instrucción MoveAbsJ.

- Progdisp desplazamiento_programa offset_externo
<desplazamiento_programa pdis>
<offset_externo eoffs>

Se utiliza para almacenar el desplazamiento de programa actual de los ejes del robot y los ejes externos. Sólo se utiliza para almacenar temporalmente el valor actual para un uso posterior.

5.5 Instrucciones de movimiento

- MoveJ Punto Velocidad Zona Herramienta


```
<Punto robtarget>
<Velocidad speeddata>
<Zona zonedata>
<Herramienta tooldata>
```

El robot se mueve hacia un punto siguiendo una trayectoria articular. Se emplea este tipo de movimiento cuando el robot no tiene que seguir ninguna trayectoria determinada, ya que es mas rápido y evita las singularidades.

Ejemplo

```
MoveJ p1,vmax,z30,pistola;
MoveJ *,v1000,fine,pistola;
```

- MoveL Punto Velocidad Zona Herramienta

```
<Punto robtarget>
<Velocidad speeddata>
<Zona zonedata>
<Herramienta tooldata>
```

El robot se mueve hacia un punto en línea recta.

- MoveC PuntoCírculo Punto Velocidad Zona Herramienta

```
<PuntoCírculo robtarget>
<Punto robtarget>
<Velocidad speeddata>
<Zona zonedata>
<Herramienta tooldata>
```

El robot se mueve hacia un punto trazando una circunferencia.

- MoveAbsJ Punto Velocidad Zona Herramienta

```
<Punto jointtarget>
<Velocidad speeddata>
<Zona zonedata>
<Herramienta tooldata>
```

Mueve el robot a una posición de ejes absoluta. Los ejes del robot y los ejes externos se desplazan hasta la posición de destino a lo largo de una trayectoria no lineal. Todos los ejes alcanzan la posición de destino al mismo tiempo. Esta instrucción sólo se puede usar si se cuenta con un sistema *MultiMove*.

Ejemplo

```
MoveAbsJ p50, v1000, z50, tool2;
MoveAbsJ *, v1000\T:=5, fine, grip3;
```

- PDispOn / PDispOff

Activa / desactiva el desplazamiento del programa. El desplazamiento de programa se activa cuando la instrucción PDispOn se ejecuta y permanece activa hasta que se activa otro desplazamiento de programa (la instrucción PDispSet o PDispOn) o hasta que se desactiva el desplazamiento de programa (la instrucción PDispOff). Esta instrucción sólo se puede usar si se cuenta con un sistema *MultiMove*.

Ejemplo

```
PROC draw_square()
PDispOn *, tool1;
MoveL *, v500, z10, tool1;
MoveL *, v500, z10, tool1;
MoveL *, v500, z10, tool1;
MoveL *, v500, z10, tool1;
PDispOff;
ENDPROC
```

- PDispSet Desplazamiento
<Desplazamiento pos>

Se usa para definir y activar un desplazamiento de programa usando la base de coordenadas conocida. Esta instrucción sólo se puede usar si se cuenta con un sistema *MultiMove*.

Ejemplo

```
VAR pose xp100 := [ [100, 0, 0], [1, 0, 0, 0] ];
...
PDispSet xp100;
```

5.6 Instrucciones de entrada/salida

- Set Señal
<Señal signaldo>

Sirve para poner una salida digital a uno.

Ejemplo

```
Set dol;
```

- Reset Señal
<Señal signaldo>

Sirve para poner una salida digital a cero.

Ejemplo

```
Reset dol;
```

- SetDO Señal Valor
<Señal signaldo>
<Valor dionum>

Sirve para cambiar el estado de una salida digital.

Ejemplo

```
SetDo dol,0;
```

5.7 Instrucciones de espera

- WaitTime Tiempo
<Tiempo num>

Detiene la ejecución del programa durante un tiempo dato en segundos.

Ejemplo

```
WaitTime 2;
```

- WaitUntil Condición [\MaxTime]
<Condición bool>
<[\MaxTime] num>

Detiene la ejecución del programa hasta que cierta condición lógica sea verdadera. Opcionalmente, se puede especificar un tiempo máximo.

Ejemplo

```
WaitUntil Dinput(di3)=1 OR Dinput(di4)=0\MaxTime:=10;
```

- WaitDI Señal Valor

<Señal signaldi>

<Valor dionum>

Detiene la ejecución del programa hasta que una entrada digital tome cierto valor.

Ejemplo

```
WaitDI di10,1;
```

5.8 Instrucciones de control de flujo

- ProcCall

Llamada a un procedimiento.

Ejemplo

```
...
cortar_barra;
...
```

- IF Condición THEN

```
...
ENDIF
```

```
IF Condición THEN
```

```
...
ELSE
```

```
...
ENDIF
```

```
IF Condición THEN
```

```
...
ELSEIF Condición THEN
```

```
...
ELSEIF Condición THEN
```

```
...
ELSE
```

```
...
ENDIF
```

Ejecución condicional.

- GOTO Etiqueta

Salto incondicional a una instrucción.

Ejemplo

```
GOTO cierre;
```

```
...
cierre:
```

Reset pinza;

- WHILE Condición DO

```
...
ENDWHILE
<Condición bool>
```

Ejecución de un bucle controlado por una condición. Permanece dentro del bucle mientras la condición sea verdadera.

Ejemplo

```
WHILE total<30 DO
  paletizar;
  Inc total,
ENDWHILE
```

- FOR Contador FROM Valor inicial TO Valor final [STEP Incremento] DO

```
...
ENDFOR
<Condición bool>
<Valor inicial num>
<Valor inicial num>
<Incremento num>
```

Ejecución de un bucle controlado por un contador.

Ejemplo

```
FOR reg1 FROM 1 TO total DO
  procesar;
ENDFOR
```

- RETURN [Valor devuelto]

Fin de ejecución de una rutina. Si la rutina es una función, devuelve el valor indicado.

Ejemplo

```
FUNC num absoluto (num valor)
  IF valor<0 THEN
    RETURN -valor;
  ELSE
    RETURN valor;
  ENDIF
ENDFUNC
```

- EXIT

Fin de ejecución del programa.

5.9 Ejemplo de programación

El robot se encuentra esperando en un punto de reposo (punto A) la activación de la entrada digital 1. En cuanto la entrada digital 1 pasa al estado UNO, el robot se aproxima al punto B para coger la pieza. El robot lleva la pieza al punto C pasando por una zona segura. El ciclo finaliza en el punto de reposo.

```
%%%
VERSION:1
```

LANGUAGE:ENGLISH

%%%

MODULE EJEMPLO

```

CONST   robtarget      A:=[[0,0,0],          [0,0,0,0],          [0,-1,0,0],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST   robtarget      B:=[[0,0,0],          [0,0,0,0],          [0,-1,0,0],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST   robtarget      B1:=[[0,0,0],         [0,0,0,0],          [0,-1,0,0],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST   robtarget      C:=[[0,0,0],          [0,0,0,0],          [0,-1,0,0],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST   robtarget      C1:=[[0,0,0],         [0,0,0,0],          [0,-1,0,0],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST   robtarget      D:=[[0,0,0],          [0,0,0,0],          [0,-1,0,0],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST   tooldata       pinza:=              [TRUE,          [[0,0,0],[1,0,0,0]],
[0,[0,0,0],[1,0,0,0],0,0,0]];

```

```

PROC cerrar_pinza()
  Set spinza;
ENDPROC

```

```

PROC abrir_pinza()
  Reset spinza;
ENDPROC

```

```

PROC coger_pieza()
  MoveJ B1,v100,z5,pinza;
  MoveL B,v80,fine,pinza;
  cerrar_pinza;
ENDPROC

```

```

PROC main()
  CONST dionum listo:=1;
  abrir_pinza;
  WHILE TRUE DO
    MoveJ A,v100,fine,pinza;
    WaitDI econtrol,listo;
    coger_pieza;
    MoveL B1,v80,z5,pinza;
    MoveJ D,v100,z100,pinza;
    MoveJ C1,v100,z5,pinza;
    MoveL C,v80,fine,pinza;
    abrir_pinza;
    MoveL C1,v80,z5,pinza;
  ENDWHILE
ENDPROC
ENDMODULE

```

6. Entradas/Salidas

Entre los parámetros del sistema del Sistema 5, están las entradas del sistema. Se han dispuesto para que un dispositivo de control externo, como puede ser un PLC, pueda tomar el mando del robot a través de señales digitales.

Los objetos de trabajo suelen asociarse a los útiles empleados para sujetar las piezas sobre las que trabaja el robot. Un cambio en la distribución en planta de la célula solamente implicará el ajuste de los objetos de trabajo, mientras que los puntos programados referidos a ellos seguirán siendo válidos. Por este motivo, el empleo habitual de los objetos de trabajo es una buena práctica de programación.

Los ejercicios de programación propuestos en esta práctica, requieren la utilización de entradas del sistema, objetos de trabajo y TCPs estacionarios.

7 Sistemas de coordenadas

Una trayectoria programada es una sucesión de posiciones que debe seguir el **punto central de la herramienta** (*Tool Center Point*). Cuando se programa un movimiento a una posición, es el TCP el que se mueve a esa posición. El TCP es el punto de la herramienta que interesa posicionar y orientar para realizar la tarea. Por ejemplo, en una pistola de aplicación de adhesivo, el TCP estará en la boca. Para facilitar la programación según la tarea que el robot debe realizar, el TCP puede estar referido a distintos sistemas de coordenadas

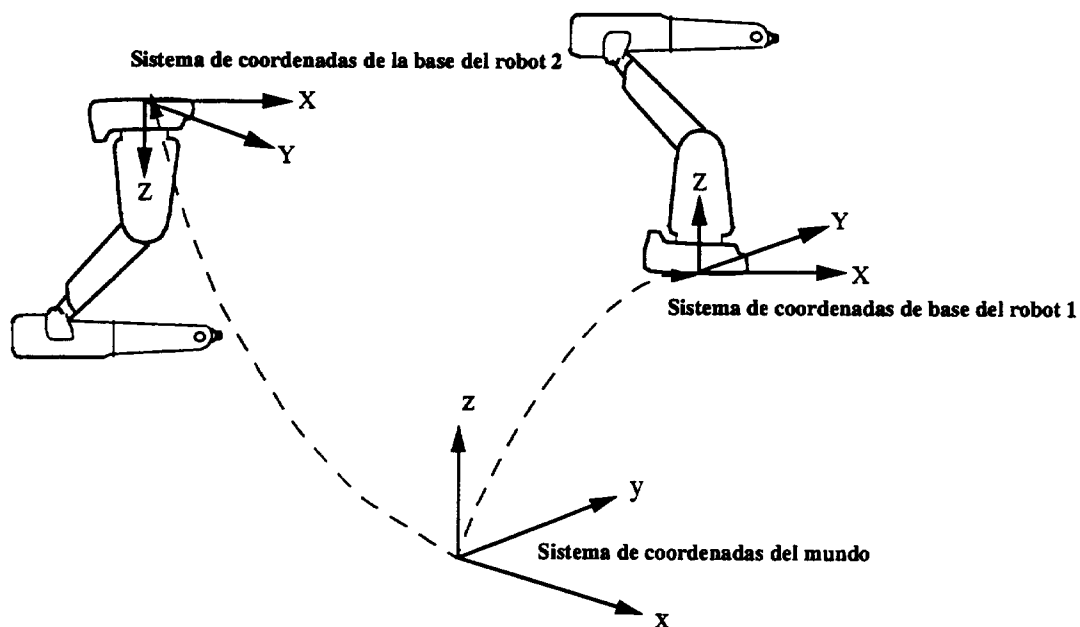


Figura 4.1: Sistemas de coordenadas de la base y del mundo

7.1 Sistemas de coordenadas para posicionar el TCP

Sistema de coordenadas de la base

Si no se define ningún sistema de coordenadas, las posiciones están referidas al sistema de coordenadas de la base.

Sistema de coordenadas del mundo

Si un robot no está montado en el suelo, o si varios robots están trabajando en la misma área, es útil disponer de un sistema de coordenadas del mundo.

Sistema de coordenadas del usuario

Este sistema se asocia a distintas áreas en las que trabaja un robot.

Sistema de coordenadas del objeto

Si un robot trabaja sobre varios objetos, cada uno puede tener su propio sistema de coordenadas asociado.

Sistema de coordenadas de desplazamiento

Cuando un robot debe repetir los mismos movimientos en áreas distintas, se puede aplicar un sistema de coordenadas de desplazamiento.

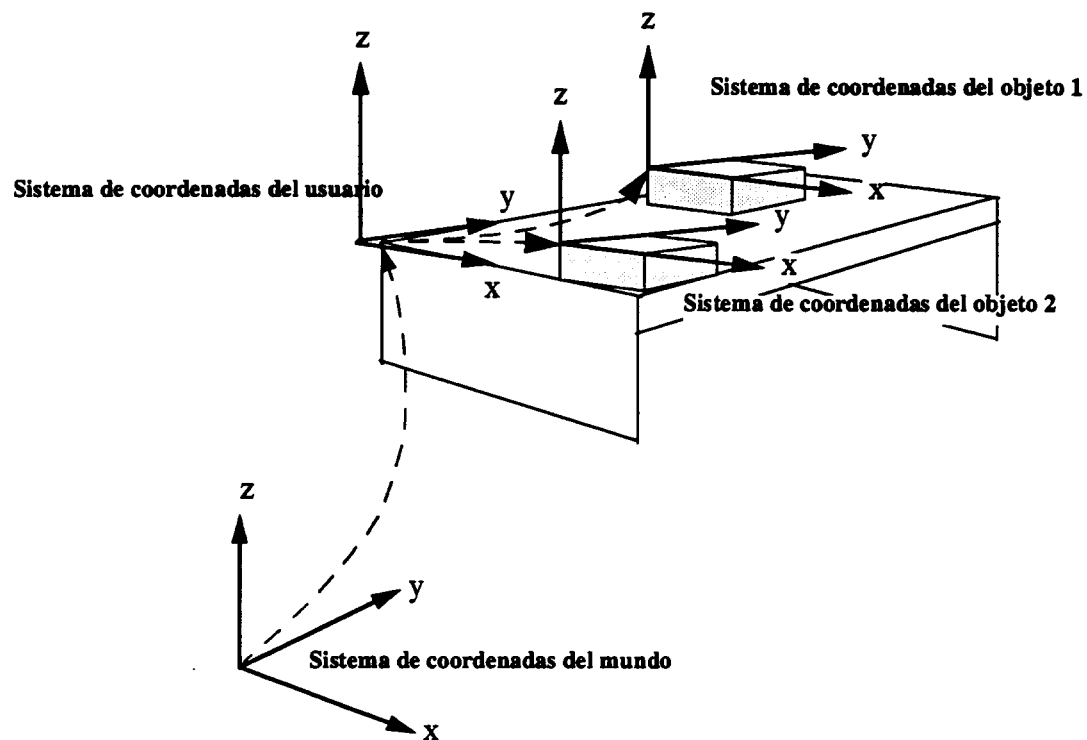


Figura 4.2: SDC del usuario y SDC del objeto

7.2 Sistemas de coordenadas para orientar el TCP

Sistema de coordenadas de la muñeca

Está asociado a la brida portaherramientas y no puede ser cambiado.

Sistema de coordenadas de la herramienta

Este sistema de coordenadas tiene su origen en el TCP y la orientación adecuada para conseguir las direcciones de movimiento necesarias para realizar la tarea.

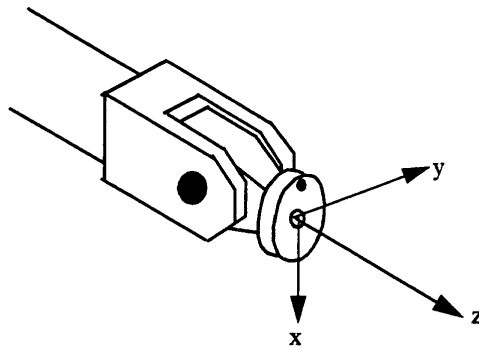


Figura 4.3: SDC de la muñeca

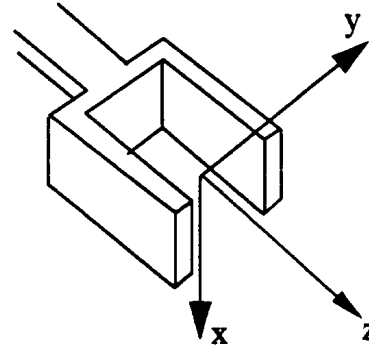


Figura 1.4: SDC de la herramienta

TCP estacionario

Si el robot sujeta la pieza y trabaja sobre una herramienta fija, se debe utilizar un TCP estacionario asociado a la herramienta.

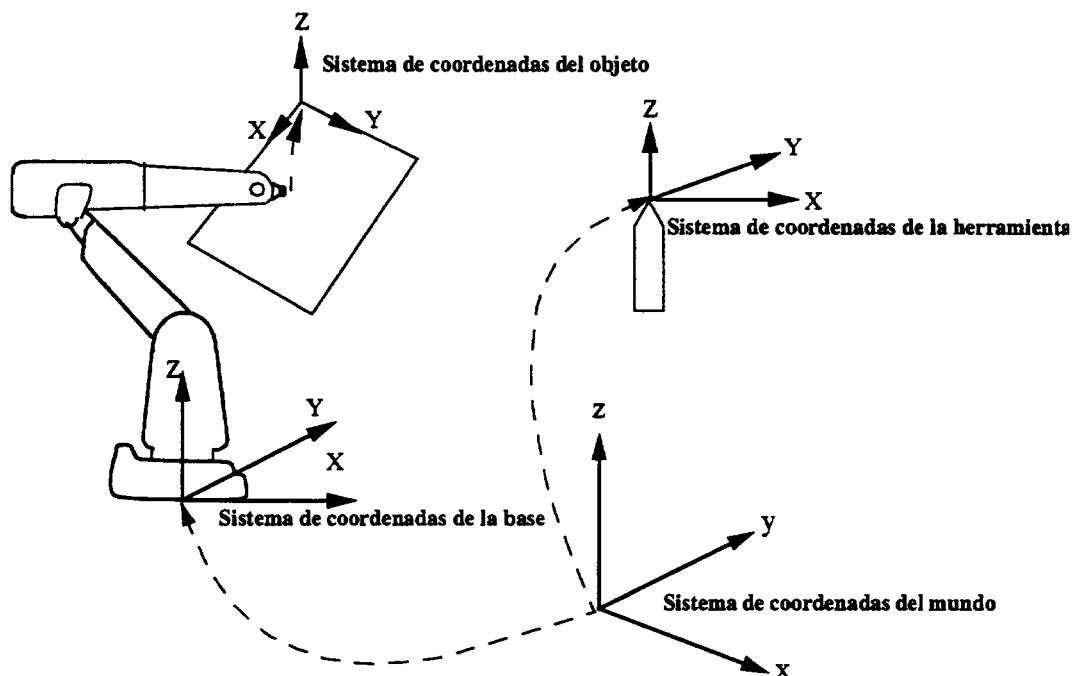


Figura 4.5: TCP estacionario

7.3 Definición de herramientas

Se conoce como herramienta, en el ámbito de los elementos terminales de robots, como aquel dispositivo que permite realizar una transformación sobre el elemento a fabricar no limitándose a un simple transporte como era el caso de las pinzas industriales.

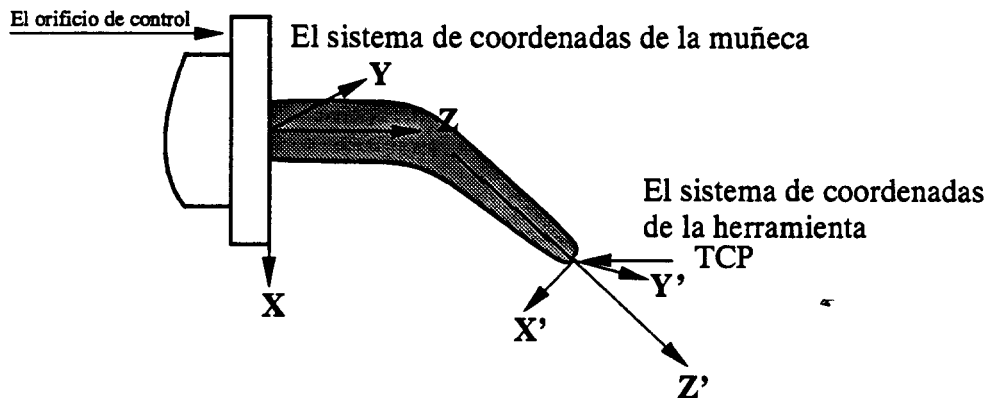


Figura 4.6: TCP

Es patente la importancia del TCP, ya que a él se refieren la mayoría de los movimientos. Pero además de las coordenadas del TCP en el sistema de coordenadas de la muñeca, el controlador del robot necesita conocer la orientación del SDC de la herramienta y algunos datos dinámicos de la misma, como son el centro de gravedad y el momento de inercia. Estos últimos son utilizados por el control dinámico del robot para asegurar un seguimiento óptimo de la trayectoria programada. Toda esta información debe agruparse, a la hora de definir una herramienta, en un dato de tipo `tooldata`.

Estructura del dato

```
tooldata
<robhold bool>
<tframe pose>
<tload loaddata>
  <mass num>
  <cog pos>
  <aom orient>
  <ix num>
  <iy num>
  <iz num>
```

`robhold` ¿Está el robot sujetando la herramienta?

`tframe` Sistema de coordenadas de la herramienta en el SDC de la muñeca.

`tload` Características dinámicas de la herramienta.

`mass` Peso en Kg.

`cog` Coordenadas del centro de gravedad en el SCD de la muñeca.

`aom` Orientación del SDC de momento en el SDC de la herramienta. El SDC de momento $X'Y'Z'$ tiene su origen en el centro de gravedad y es aquel con respecto al cual está expresado el momento de inercia.

i_x Momento de inercia alrededor del eje X' del SDC de momento, en $\text{Kg}\cdot\text{m}^2$. i_y Momento de inercia alrededor del eje Y' del SDC de momento, en $\text{Kg}\cdot\text{m}^2$. i_z Momento de inercia alrededor del eje Z' del SDC de momento, en $\text{Kg}\cdot\text{m}^2$.

Definición

Las herramientas deben definirse como variables persistentes (PERS) y nunca dentro de una rutina.

Sólo es necesario especificar el momento de inercia de la herramienta si su dimensión es mayor que la distancia de la brida portaherramientas a su centro de gravedad. En el resto de los casos puede suponerse una carga puntual, siendo los datos i_x , i_y , i_z iguales a 0 $\text{Kg}\cdot\text{m}^2$.

Ejemplo

```
PERS tooldata pinza := [TRUE, [[97.4,0,223.1],[0.924,0,0.383,0]],
[5,[23,0,75],[1,0,0,0],0,0,0];
```

El robot sujeta la herramienta, las coordenadas del TCP en el SDC de la muñeca son (97.4, 0, 223.1) (mm), los ejes X e Y del SDC de la herramienta están girados 45 grados respecto al SDC de la muñeca, el peso de la herramienta es de 5 Kg, las coordenadas del centro de gravedad son (23, 0, 75) (mm) en el SDC de la muñeca, la carga de la herramienta se supone puntual.

Programación

La especificación de la herramienta es un argumento obligatorio en las instrucciones de movimiento.

Ejemplo

```
MoveJ p1,v500,z50,pinza;
```

7.4 Definición de cargas

Cuando un robot está portando una carga agarrada con una herramienta, la carga influye en su comportamiento dinámico y, si no está definida en el software, puede perturbar la trayectoria programada. Las cargas se definen con el tipo de dato `loaddata`.

Estructura del dato

```
loaddata
<mass num>
<cog pos>
<aom orient>
<ix num>
```

```
<iy num>
<iz num>
```

La descripción de cada componente del dato ya ha sido vista en el apartado anterior.

Definición

Las cargas deben definirse como variables persistentes (PERS) y nunca dentro de una rutina.

Sólo es necesario especificar el momento de inercia de la carga si su dimensión es mayor que la distancia de la brida portaherramientas a su centro de gravedad. En el resto de los casos puede suponerse una carga puntual, siendo los datos i_x , i_y , i_z iguales a 0 Kg·m².

Ejemplo

```
PERS loaddata pieza := [5, [50, 0, 50], [1, 0, 0, 0], 0, 0, 0];
```

Carga de 5 Kg de peso, las coordenadas del centro de gravedad son (50,0,50) (mm) en el SDC de la herramienta, se supone carga puntual.

Programación

La instrucción `GripLoad` sirve para especificar en cada momento qué carga está portando el robot. La ausencia de carga equivale al dato predefinido `load0`.

Ejemplo

```
Set dpinza;
WaitTime 0.5;
GripLoad pieza;
MoveL p1, v100, fine, tpinza;
Reset dpinza;
WaitTime 0.5;
GripLoad load0;
```

7.5 Objetos de trabajo y TCPs estacionarios

Es de gran interés el caso de que los puntos programados estén referidos al sistema de coordenadas del objeto de trabajo. Así, ante un cambio en la localización del objeto sobre el que el robot va a trabajar, o del robot mismo, no será necesaria la reprogramación de estos puntos. Bastará con ajustar la nueva situación relativa del SDC del objeto de trabajo respecto del SDC del mundo.

En algunas aplicaciones el robot no porta la herramienta, sino el objeto de trabajo, mientras que la herramienta está fija. Entonces, se habla de un *TCP estacionario*. Este es el único caso en el que los movimientos programados no se refieren al TCP, que está inmóvil, sino al objeto de trabajo.

Objetos de trabajo y TCPs estacionarios se definen con el tipo de dato `wobjdata`.

Estructura del dato

```
wobjdata
<robhold bool>
<ufprog bool>
<ufmec string>
<uframe pose>
<oframe pose>
```

robhold

¿Está el robot sujetando el objeto de trabajo?. Es verdadero para herramientas estacionarias.

ufprog

¿Está fijo el sistema de coordenadas de usuario?. Es falso cuando se utilizan ejes externos coordinados.

ufmec

Unidad mecánica (motor) asociada al eje externo coordinado (sólo si ufprog es falso).

uframe

Sistema de coordenadas del usuario, normalmente asociado a la superficie de trabajo o al utillaje, respecto del SDC del mundo. En caso de herramienta estacionaria, se refiere al SDC de la muñeca.

oframe

Sistema de coordenadas del objeto de trabajo referido al SDC del usuario.

Definición

Los objetos de trabajo deben definirse como una variables persistente (PERS) y nunca dentro de una rutina.

Ejemplo

```
PERS wobjdata obj2 := [FALSE, TRUE, "", [[300,600,200],[1,0,0,0]],
[[0,200,30],[1,0,0,0]]];
```

El robot sujeta la herramienta. El SDC de usuario está fijo, no ha sido girado y su origen se encuentra en las coordenadas (300,600,200) (mm) respecto al SCD del mundo. El SDC del objeto de trabajo tampoco ha sido girado y las coordenadas de su origen están en (0,200,30) (mm) en el SDC del usuario.

Programación y ajustes

La especificación del objeto de trabajo es un argumento opcional en las instrucciones de movimiento.

Ejemplo

```
MoveL p1,v1000,z50,tool1\Wobj:=obj2;
```

Posibles ajustes en el layout de la instalación del robot pueden ser fácilmente corregidos dentro del programa.

Ejemplo

```
obj2.oframe.trans.z:=38.3;
```

7.6 Método para la definición del sistema de coordenadas de la herramienta.

Para definir el TCP de una herramienta, se necesita una punta de referencia fija en el espacio de trabajo del robot. Se debe realizar un desplazamiento a (al menos) cuatro posiciones del robot con orientaciones diferentes; lo más cerca posible de la punta de referencia fija (consulte la **¡Error! No se encuentra el origen de la referencia.**). Estas posiciones se conocen como puntos de aproximación.

Para definir la orientación completa de una herramienta, se debe mover cualquier posición del eje z deseado y cualquier posición del eje x deseado hacia la punta de referencia fija. Estas posiciones se conocen como puntos de elongación (consulte la **¡Error! No se encuentra el origen de la referencia.**). Esto puede hacerse montando un alargador en la herramienta para definir las direcciones z y x o mediante la alineación de la herramienta acorde con el sistema de coordenadas mundo y moviendo el robot en estas direcciones.

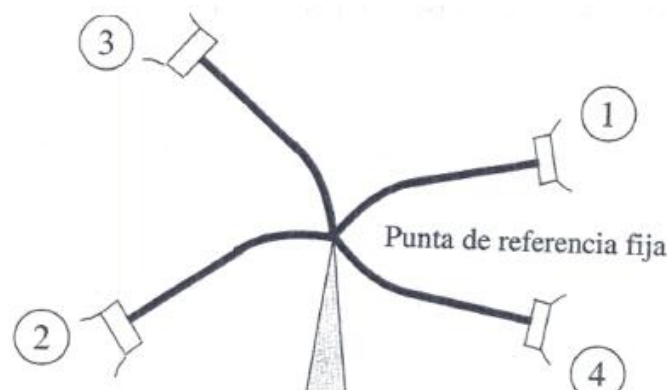


Figura 4.7: Puntos de aproximación para el TCP de una herramienta

Nota: Los puntos de elongación deben definirse en la misma orientación que el último punto de aproximación utilizado.

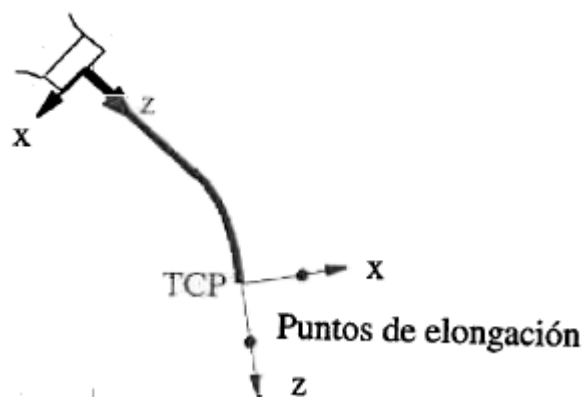


Figura 4.8: Puntos de elongación para la orientación de una herramienta.

Si sólo se desea definir el TCP, sólo se necesita la punta de referencia fija. Si sólo

necesita una definición de la orientación en la dirección z, el alargador sólo apuntará hacia z.

Se admiten los métodos siguientes:

- TCP de 4 puntos

Para definir el TCP se utilizan cuatro puntos de aproximación. La orientación se define acorde con el sistema de coordenadas de la herramienta (consulte la **¡Error! No se encuentra el origen de la referencia.**).

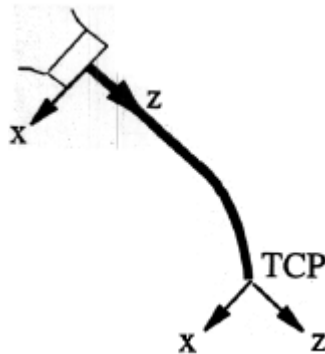


Figura 4.9: Con el método de 4 puntos. sólo se define el TCP. La dirección de la herramienta se corresponde con el sistema de coordenadas de la muñeca.

- ORIENTACIÓN DE TCP DE 4 PUNTOS NO DEFINIDA

Lo mismo que con el TCP de 4 puntos, pero la orientación no cambia.

- TCP y Z de 5 puntos

Se utilizan cuatro puntos de aproximación para definir el TCP y un punto de elongación para definir la dirección z de la herramienta. Las direcciones x e y estarán lo más cerca posible de los ejes correspondientes en el sistema de coordenadas de la 'muñeca' (consulte la **¡Error! No se encuentra el origen de la referencia.**).

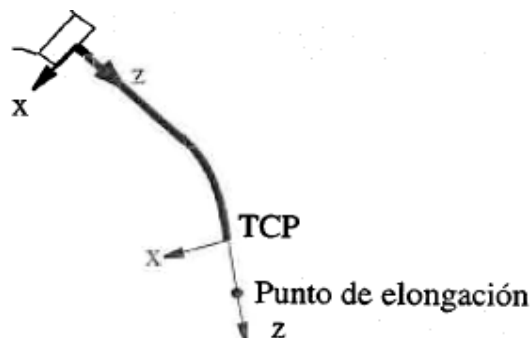


Figura 4.2: El método de 5 puntos permite definir el TCP y la dirección z de la herramienta. Las direcciones x e y son definidas automáticamente por el robot.

- **ORIENTACIÓN DE TCP DE 4 PUNTOS NO DEFINIDA**

Lo mismo que con el TCP de 4 puntos, pero la orientación no cambia.

- TCP y ZX de 6 puntos

Se utilizan cuatro puntos de aproximación para definir el TCP y un punto de elongación para definir la dirección z de la herramienta. Además, se usa un punto de elongación para definir la dirección x de la herramienta (consulte la **¡Error! No se encuentra el origen de la referencia.**).

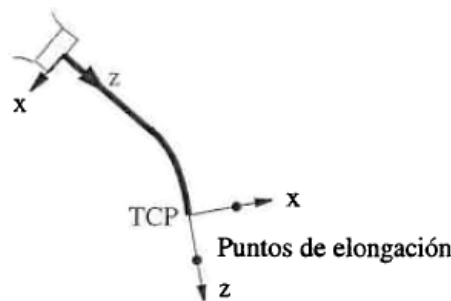


Figura 4.3: El método de 6 puntos permite definir el TCP y todas las direcciones de la herramienta.

8 Parámetros del sistema

Los **parámetros del sistema** son un conjunto de opciones de configuración del robot que sirven para adaptarlo a una determinada aplicación. Así, el robot puede estar preparado para operar en un área de aplicación u otra cambiando tan sólo los parámetros del sistema. Por ejemplo, los nombres de las entradas/salidas o las características de los ejes externos están incluidos aquí.

Los parámetros del sistema están agrupados en **temas**. Son los siguientes:

- Entradas/Salidas
- Manipulador
- Soldadura por arco

Dentro de un tema, los parámetros del sistema se dividen en **tipos**.

Siempre que se cambie algún parámetro, para que tenga efecto el cambio, es necesario reinicializar el sistema. Además, los parámetros del sistema se pueden guardar y restaurar ya sea por temas, o todos en conjunto.

Tema Entradas/Salidas

Tipos:

Tarjetas de E/S

Señales de usuario

Se puede cambiar el parámetro *Nombre Señal* para dar un nombre de usuario a una señal. Si se trata de una señal analógica, los parámetros *Max Lógico*, *Min Lógico*, *Max Físico* y *Min Físico* configuran el escalado entre el valor programado (lógico) y el físico (voltios o amperios).

Grupos de señales

Un **grupo de señales** es un conjunto de señales digitales que se manejan como una sola. El grupo toma valores numéricos según el código binario formado por las señales digitales integrantes.

Entradas del sistema

Controlar el robot desde un PLC mediante señales digitales puede hacerse definiendo **entradas del sistema**. Se trata de asociar entradas digitales a ciertas acciones o actividades.

Salidas del sistema

También las salidas digitales pueden ser configuradas para informar sobre el estado del robot. Son las **salidas del sistema**, asociadas a algunas actividades o estados del robot.

E/S enlazadas

Una entrada digital puede ser enlazada internamente a una salida, y viceversa.

Canales serie

Tema Manipulador**Tipos:**

Motor

Brazo

Carga del brazo

CheckPoint del brazo

Robot

Tema Soldadura por arco

Práctica 1

Los robots ABB IRB 1600 y IRB 2400: Funcionamiento básico



1 INTRODUCCIÓN

El objetivo de esta práctica es presentar los modos de operación de los robots ABB IRB 1600 y IRB 2400 con el controlador IRC5. Al mismo tiempo se iniciarán ejercicios sencillos de programación por guiado. Por último se establecerán las normas básicas de seguridad que deben ser observadas durante el manejo del robot.

2 FUNCIONAMIENTO BÁSICO

Los robots ABB de la serie IRB 1600 y IRB 2400 constan básicamente de un controlador, o armario de control, una unidad de programación, o *Flex Pendant*, y un manipulador, o brazo (figura 2.1).

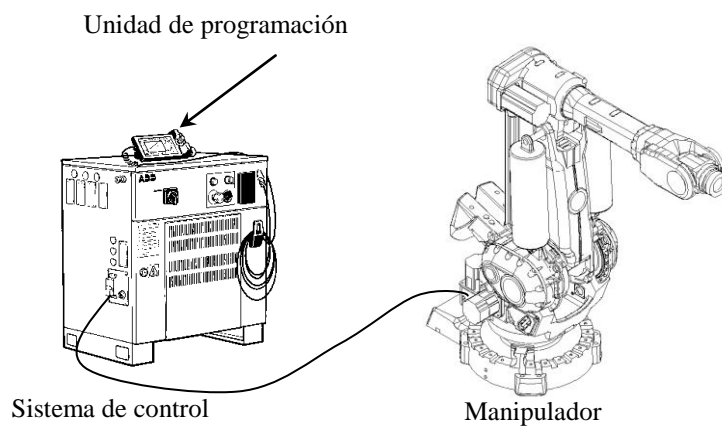
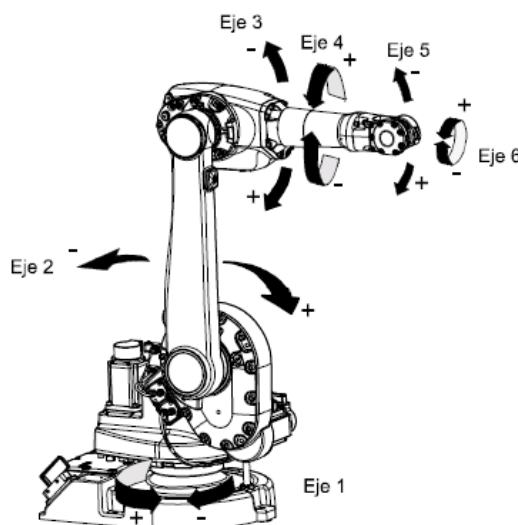


Figura 2.1: Conjunto de controlador, manipulador y unidad de programación.

2.1 Manipulador IRB 1600

El manipulador responde a una morfología angular, o antropomórfica, con 6 grados de libertad. La figura 2.2 muestra el manipulador del robot IRB 1600 y sus grados de libertad.



El robot IRB 1600 de ABB es un robot versátil de gran rendimiento, se ofrece en cuatro versiones; la mayor capacidad de carga del IRB 1600 lo convierte en el robot más fuerte del mercado. Además, está bien preparado para un trabajo flexible y de poco coste para prestar servicio en muchas aplicaciones.

El robot IRB 1600 que está en el laboratorio es el modelo IRB 1600-6/1,2 y posee

Figura 2.2: Robot IRB 1600

principalmente las siguientes características:

♦ **Capacidad de carga y alcance:**

La capacidad de carga es de 6 Kg y el alcance de 1,2 m. En la figura 2.3 se puede ver el rango de trabajo y el diagrama de carga para el mencionado modelo del robot.

♦ **Fiable: elevados tiempos útiles de producción:**

La combinación de una tecnología ya demostrada y de innovaciones bien probadas se traduce en un alto tiempo medio de vida entre fallos (MTBF), pocos requisitos de mantenimiento y unos tiempos de reparación reducidos.

♦ **Rápido: ciclos de corta duración**

Más rápido que cualquier otro robot de la competencia en su clase.

♦ **Preciso: calidad uniforme de las piezas**

Una repetibilidad extraordinaria ($\pm 0,05\text{mm}$) y una precisión muy buena en el recorrido.

♦ **Versátil: integración y fabricación flexibles**

Concepto de doblado hacia atrás y varias opciones de montaje (pared, suelo, invertido o inclinado)

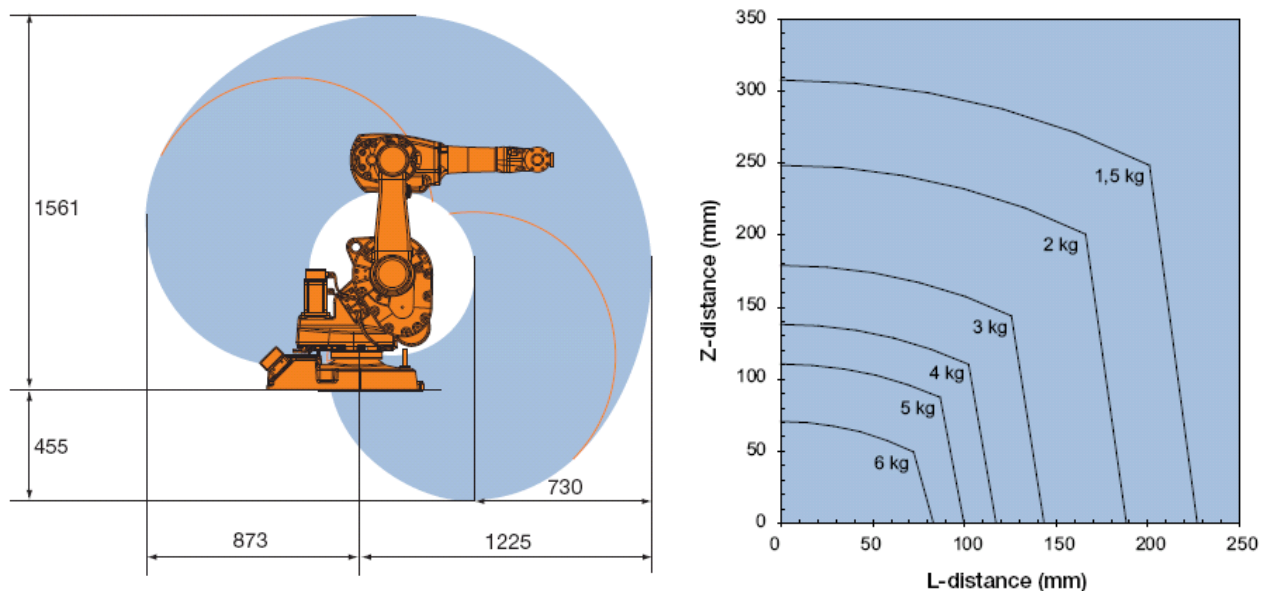


Figura 2.3: Rango de trabajo y diagrama de carga para el robot IRB 1600-6/1,2

En la tabla 2.1 se observa los rangos de movimiento de cada eje.

Ejes	Rango de movimiento
1	+180° a -180°
2	+136° a -63°
3	+55° a -235°
4	+200° a -200°
5	+115° a -115°
6	+400° a -400°

Tabla 2.1: Rango de los movimientos de cada eje.

2.2 Manipulador IRB 2400

El IRB 2400, proporciona unas prestaciones excelentes para la manipulación de materiales, la asistencia a la mecanización y las aplicaciones de procesos. Ofrece mayores volúmenes de producción, menores tiempos de entrega y entregas más rápidas para los productos que fabrica el cliente que en el caso del IRB 1600. Cuenta así mismo con 6 ejes (figura 2.4). El modelo existente en el laboratorio es el IRB 2400-16/1,5

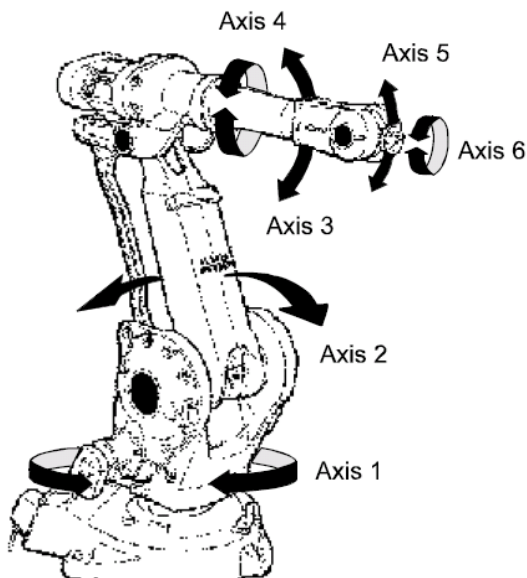


Figura 2.4: El robot IRB 2400

El robot IRB 2400 posee las siguientes características:

♦ **Capacidad de carga y alcance:**

Las opciones de capacidad de carga son de 20 kg, y el radio de acción máximo es de 1,5 m.

♦ **Fiable: elevados tiempos útiles de producción**

IRB 2400 es el robot industrial más popular del mundo. La robusta fabricación y el uso de un número mínimo de componentes contribuye a su alta fiabilidad y a los largos periodos entre operaciones de mantenimiento.

♦ **Rápido: ciclos de corta duración**

Gracias al control de movimiento, exclusivo de ABB, optimiza la aceleración y la deceleración, lo que se traduce en la menor duración posible de los ciclos.

♦ **Preciso: calidad uniforme de las piezas**

El mejor en su clase por lo que se refiere a la precisión del recorrido y la repetibilidad de la posición.

♦ **Versátil: integración y fabricación flexibles**

Todos los modelos se ofrecen con posibilidad de montaje invertido.

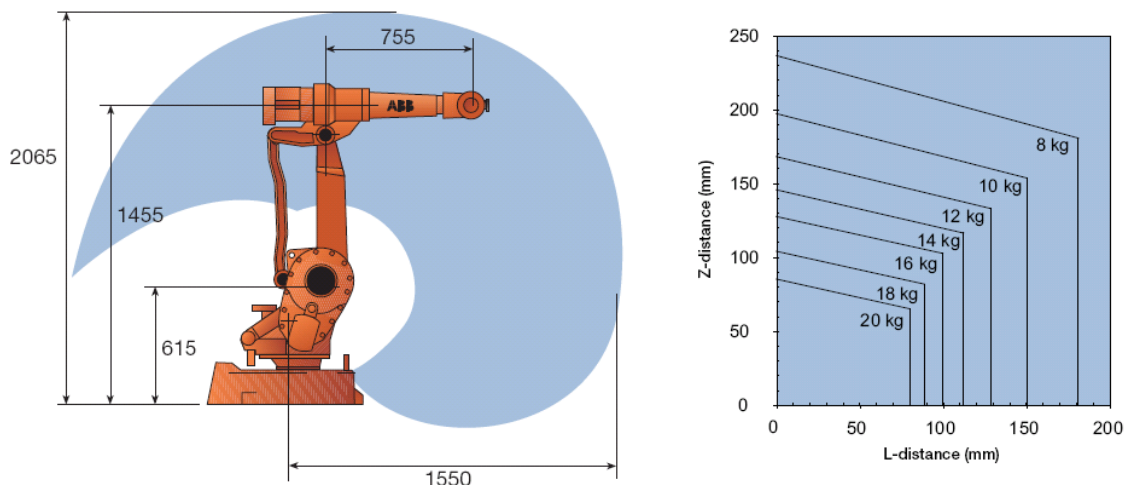


Figura 2.5: Rango de trabajo y diagrama de carga para el robot IRB 2400-16/1,5

En la tabla 2.2 se observa los rangos de movimiento de cada eje.

Ejes	Rango de movimiento
1	+180° a -180°
2	+110° a -100°
3	+65° a -60°
4	+185° a -185°
5	+115° a -115°
6	+400° a -400°

Tabla 2.2: Rango de los movimientos de cada eje.

2.3 Controlador

IRC5 es un controlador de robot de ABB de quinta generación. Incorpora nuevas características con su concepto modular, una unidad de interfaz portátil de diseño ergonómico totalmente nuevo, el FlexPendant y un control de robots múltiples (hasta cuatro) totalmente sincronizado por medio de la función MultiMove.

El controlador IRC 5 (figura 2.6) guarda en su interior el conjunto de la electrónica de control donde el programa es procesado. Además, en su frontal se encuentran el interruptor principal, la unidad de disco, y el panel de control, que reúne los principales mandos del robot.



Figura 2.6: Controlador IRC 5

El panel de control (figura 2.7) reúne el selector de modo de operación, los pulsadores de MOTORES ON y MOTORES OFF, y el pulsador de parada de emergencia. El selector de modo de operación conmuta entre los modos AUTO, MANUAL y MANUAL VELOCIDAD TOTAL.

- **Modo AUTO (modo de producción)**
El robot ejecuta el programa sin la presencia de un operador. No es necesario que el operador esté habilitando la unidad de programación.
- **Modo MANUAL (modo de programación)**
El operador está programando el robot a través de la unidad de programación. La velocidad está limitada a 250 mm/s.
- **Modo MANUAL VELOCIDAD TOTAL (modo de comprobación)**
El operador está comprobando el funcionamiento del programa sin limitación de velocidad.



Figura 2.7: Panel de control para el armario sencillo

Los indicadores de MOTORES ON y MOTORES OFF señalan si los motores están en tensión o no, respectivamente. Al mismo tiempo funcionan como pulsadores que sirven para pasar de un estado al otro.

En Aula global se puede descargar el data sheet del controlador IRC 5.

2.4 Unidad de programación

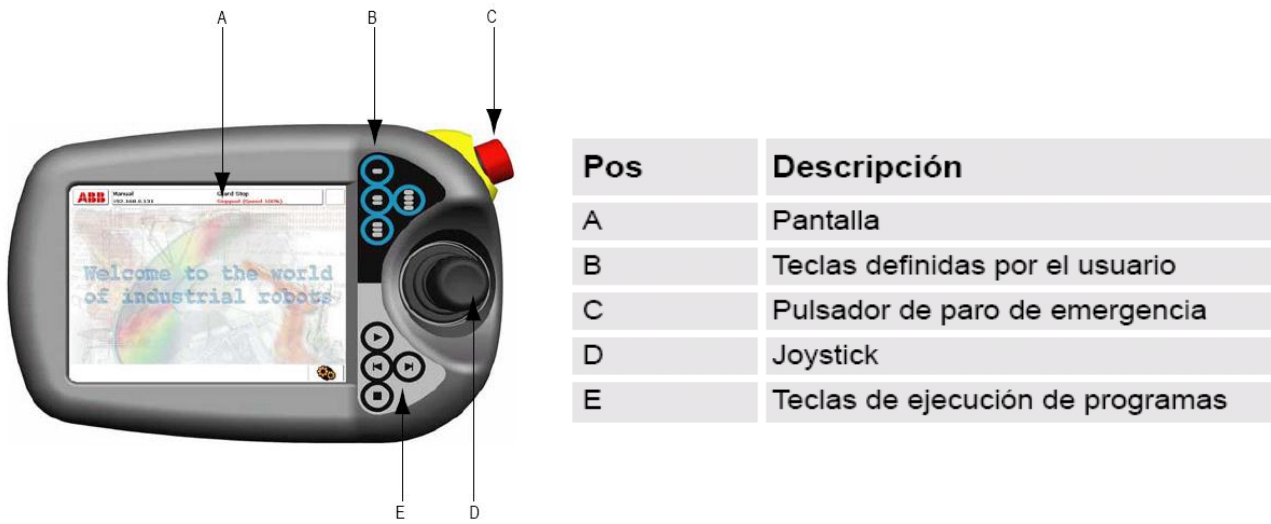


Figura 2.8: FlexPendant

La interacción entre el usuario y el robot tiene lugar a través de la unidad de programación (FlexPendant). Integrados en la unidad de programación se encuentran un

visualizador, una botonera, un pulsador de parada de emergencia y una palanca de mando o *joystick* (figura 2.8). El FlexPendant se usa para realizar muchas de las tareas implicadas en el manejo de un sistema de robot: ejecutar programas, mover el manipulador, modificar programas del robot, etc. En la figura 2.9 se muestra la pantalla principal del FlexPedant.

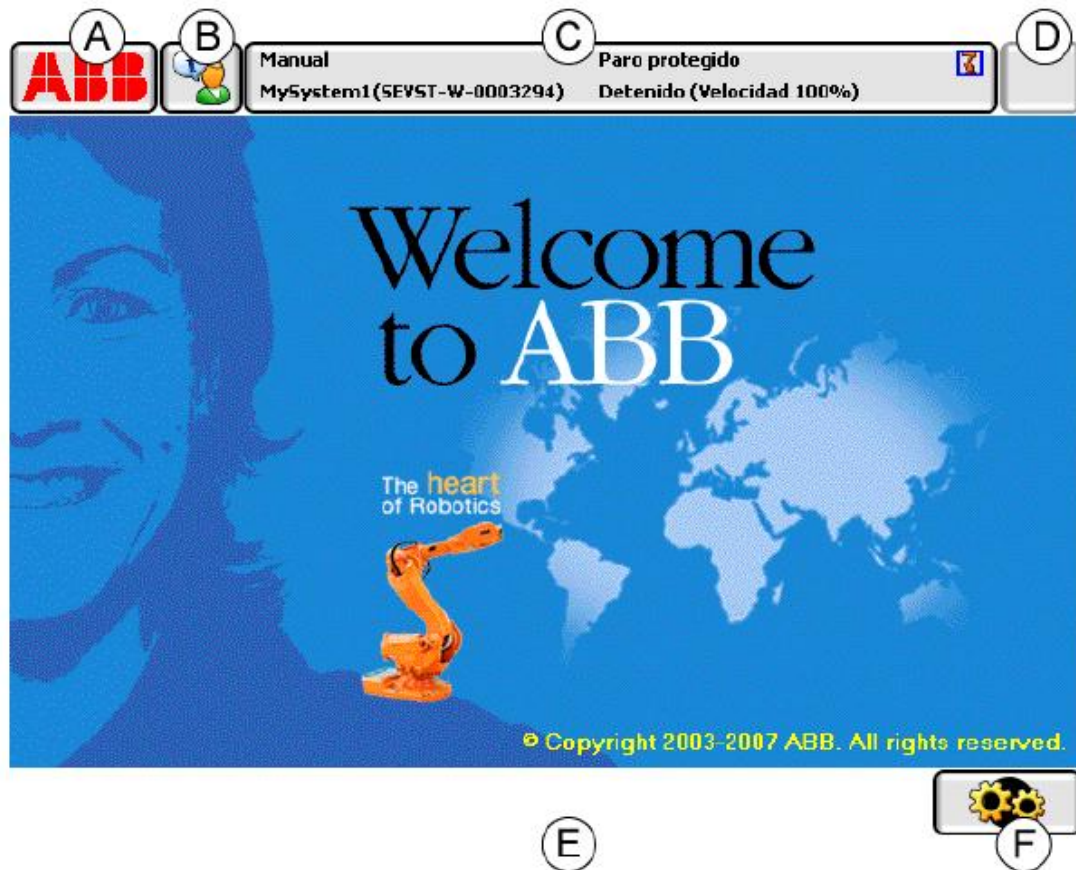


Figura 2.9: Pantalla del FlexPedant

A	Menú ABB
B	Ventana de operador
C	Barra de estado
D	Botón Cerrar
E	Barra de tareas
F	Menú de configuración rápida

Menú ABB

El menú ABB permite seleccionar los elementos siguientes:

- HotEdit
- Entradas y salidas
- Movimiento
- Ventana de producción
- Editor de programas
- Datos de programa
- Copia de seguridad y restauración

- Calibración
- Panel de control
- Registro de eventos
- FlexPendant Explorer
- Información del sistema
- Etc.

Ventana de operador

La ventana de operador muestra mensajes de los programas del robot. Suelen aparecer cuando el programa requiere algún tipo de respuesta del operador para poder continuar.

Barra de estado

La barra de estado muestra información importante acerca del estado del sistema, como por ejemplo el modo de funcionamiento, Motors ON/OFF, el estado del programa, etc..

Botón Cerrar

Al tocar el botón Cerrar se cierra la vista o aplicación que esté activa actualmente.

Barra de tareas

Puede abrir varias vistas desde el menú ABB, pero sólo trabajar con una cada vez. La barra de tareas muestra todas las vistas abiertas y se utiliza para cambiar entre ellas.

Menú de configuración rápida

El menú de configuración rápida contiene valores sobre el movimiento y la ejecución de programas.

2.5 Consideraciones de seguridad

Un robot industrial es una máquina potencialmente muy peligrosa para el operador. Incluso a velocidad reducida, el brazo es una masa con una enorme inercia. Por tanto, es imprescindible seguir rigurosamente un conjunto mínimo de normas de seguridad.

El operador debe ser consciente en todo momento de que un robot puede hacer movimientos inesperados. Una pausa o detención aparente puede ser seguida de un movimiento brusco a gran velocidad. Por otra parte, las entradas externas pueden influir en la trayectoria del robot sin previo aviso. Tampoco está descartado que un error del sistema o de tipo mecánico provoque un movimiento inesperado.

Elementos de seguridad

El robot incorpora algunos elementos de seguridad:

- **Paro de emergencia:** dispositivo que elimina la alimentación de los accionadores del robot y elimina el suministro de alimentación eléctrica de otras funciones peligrosas controladas por el robot.
- **Dispositivo de habilitación:** dispositivo que se usa manualmente y que permite funciones peligrosas, pero no las inicia.

- **Velocidad reducida:** velocidad restringida de operación del robot en modo manual para que dé tiempo suficiente al personal a alejarse de la zona peligrosa (velocidad limitada a 250 mm/s).

Instrucciones de seguridad para el manejo de los robots

- NO ENTRAR EN EL RECINTO DE SEGURIDAD DE LOS ROBOTS SI ESTAN TRABAJANDO EN MODO AUTOMÁTICO.
- ANTES DE ACTIVAR EL MODO TRABAJO AUTOMÁTICO DEL ROBOT, ASEGURARSE QUE NO HAY NADIE EN EL INTERIOR DEL RECINTO DE SEGURIDAD.
- CUANDO EL ROBOT TRABAJA EN MODO AUTOMÁTICO, ESTAR CERCA DE LA SETA DE SEGURIDAD PARA PULSARLA RÁPIDAMENTE EN EL CASO DE QUE SURGA ALGUN PROBLEMA.
- CUANDO SEA NECESARIO TRABAJAR DENTRO DEL RECINTO DE SEGURIDAD DEL ROBOT PARA REALIZAR OPERACIONES ESPECIALES (CALIBRACIÓN), EL SISTEMA DE SEGURIDAD QUEDA DESACTIVADO, CON LO QUE HAY QUE TENER ESPECIAL CUIDADO.
- PARA CAMBIAR LAS HERRAMIENTAS O EL ENTORNO DE TRABAJO DEL ROBOT ESTE DEBERÁ ESTAR EN MODO MANUAL Y CON LOS MOTORES APGADOS. EL SISTEMA DE SEGURIDAD QUEDARÁ DESACTIVADO POR LO QUE HAY QUE TENER ESPECIAL CUIDADO.

Código de colores de las luces de seguridad

VERDE: Funcionamiento normal en modo automático.

AMARILLO: Funcionamiento en modo manual. NO ACTUA EL SISTEMA DE SEGURIDAD.

ROJO: Robot bloqueado por el sistema de seguridad debido a una intrusión en el recinto de seguridad del robot. Será necesario su rearme.

3 ACTIVIDADES

3.1 Modos de movimiento

1) Mover el robot en modo de funcionamiento manual en cada uno de los siguientes modos de movimiento, teniendo seleccionado el SDC de la base:

- movimiento articular,
- movimiento rectilíneo,
- reorientación del TCP.

2) Realizar movimientos "en incrementos".

- 3) Cambiar al sistema de coordenadas de la herramienta (tool0) y mover según sus ejes.

3.2 Programa de dibujo

- 1) Crear un programa que dibuje sobre un papel una línea que una varios puntos.
- 2) Probar el programa en modo de funcionamiento manual.
- 3) Ejecutar el programa en modo automático.
- 4) Cambiar un punto.
- 5) Comprobar sobre la línea dibujada los efectos de cambios en la velocidad y en las zonas de precisión. Variar:

$$v_5 = v_{\max}$$

$$z_{fine} = z_{150}$$

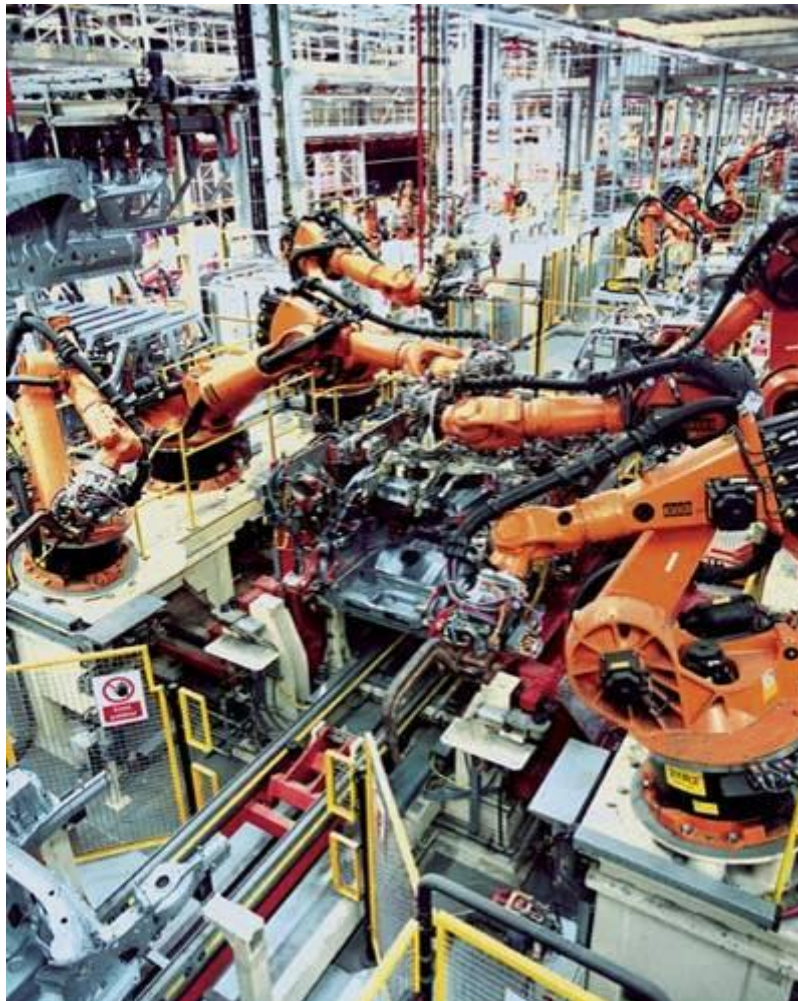
...

etc

- 6) Programar el robot para que dibuje un círculo.

Práctica 2

Programación en RAPID



1 INTRODUCCIÓN

La puesta en marcha de una instalación robotizada es un proceso laborioso y lento. Implica la definición de todos los elementos que forman parte del entorno del robot y la configuración del robot para que pueda interactuar con el mencionado entorno.

La herramienta es el principal instrumento a través del cual el robot actúa sobre su entorno de trabajo. La programación de una tarea incluye siempre la definición de la herramienta que el robot va a emplear.

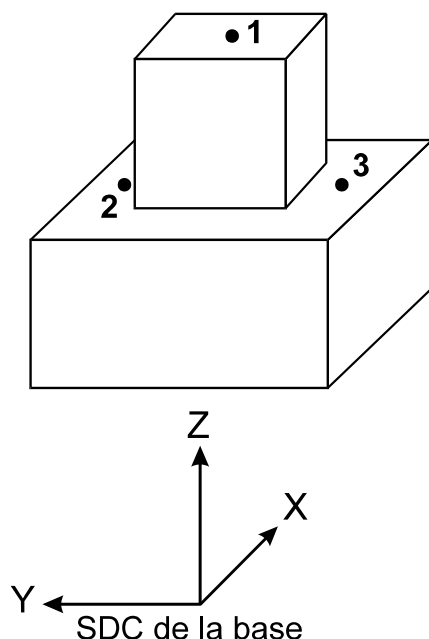
El robot intercambia datos con otros dispositivos de su entorno de aplicación para sincronizar sus movimientos con la evolución temporal de la tarea. Esta interacción tiene lugar a través de entradas/salidas. También hay que adaptar a la aplicación otros parámetros tales como los límites del área de trabajo, equipo adicional que porta el robot o el escalado de las señales analógicas. En el Sistema 4, se puede acceder a la configuración de todos estos parámetros en los parámetros del sistema.

En esta práctica se abordará un ejercicio de programación completo, en el que será necesario definir una herramienta y configurar los parámetros del sistema.

2 ACTIVIDADES

1) Definir una herramienta:

Utilizar la documentación técnica de las pinzas que se aporta en el guión para obtener los datos geométricos. Se puede suponer que el centro de gravedad está en el centro geométrico.



2) Crear un programa en RAPID para realizar la siguiente tarea:

Partiendo de una posición de reposo, el robot espera hasta que se active la entrada *di1* para ir a coger una pieza. La pieza se puede encontrar en dos posiciones posibles: el punto 2 o el punto 3. Entre estas dos posiciones, el robot debe buscar la pieza en la posición identificada por el valor del grupo de entradas digitales *gi1*, formado por *di2* y *di3*. Una vez cogida la pieza, la lleva al punto 1 sin colisionar con el entorno.

La pinza será la definida en el apartado anterior y se maneja a través de la salida digital *do1*