PRÁCTICA 4:

Iniciación a RAPID. Montaje de torres.

El objetivo de esta práctica consiste en avanzar en la programación con RAPID, familiarizándose con la estructura del programa, empleando nuevas funciones y definiendo procedimientos.

PARTE 1. ALGUNOS CONCEPTOS SOBRE RAPID

El lenguaje de programación de los manipuladores ABB es un lenguaje de alto nivel que se denomina RAPID. En esta sección se va a comentar brevemente la estructura, tipos de datos y algunas funciones básicas de este lenguaje, necesarias para el desarrollo de la práctica. Se ha empleado, como punto de partida para la elaboración de esta descripción, el documento: *Manual del operador - Introducción a RAPID* de ABB.

Estructura:

Los programas RAPID constan de uno o varios módulos, cada uno de los cuales puede contener uno o varios procedimientos. En los programas sencillos tan sólo suele haber un módulo, mientras que en los más complejos se suele disponer de más de uno. Estos módulos pueden ser de dos tipos:

- Módulos de programa. Tienen extensión *.mod*. Para el controlador no supone ninguna diferencia que el programa esté escrito en varios módulos, pero simplifica a los programadores la comprensión del programa y la reutilización del código. En el caso de prácticas anteriores, se disponía de un módulo llamado *CalibData* que contenía información sobre la herramienta y los *WorkObjects* creados, así como un módulo *Module1* con el procedimiento *Main* y otros procedimientos adicionales de las trayectorias creadas.
- Módulos de sistema. Poseen extensión *.sys*. Los datos y procedimientos que deban permanecer en el sistema incluso al cambiar de programa. Por ejemplo, en este módulo estará la definición del objeto de trabajo *wobj0*.

La definición de los módulos, tal y como muestra la Figura 1, está compuesto por una primera parte de declaración de datos y una segunda parte de definición de rutinas.

```
MODULE Module1
        Declaraciones de datos
        PROC main ()
          …
        ENPROC
        PROC my_procedure (…)
          ...
        ENDPROC
ENDMODULE
```

Figura 1. Declaración de módulo.

Los procedimientos son uno de los tipos de rutina que se pueden emplear en RAPID. De forma habitual, la ejecución de una aplicación comenzará en el procedimiento *main*, aunque se puede especificar otro punto de inicio. Este procedimiento en concreto no dispone de ningún argumento de entrada, pero se pueden declarar procedimientos con parámetros. En la Figura 2 se muestra un ejemplo en el que desde el procedimiento *main* se llama a *My_procedure* (no es preciso usar paréntesis), que posee tres parámetros de entrada de tipo *num*, *string* y *bool* respectivamente. Al declarar un procedimiento, todos los argumentos se declaran entre paréntesis a continuación del nombre del procedimiento. Esta declaración contiene el tipo de dato y el nombre de cada argumento. El argumento

recibe su valor de la llamada al procedimiento y actúa como una variable dentro del procedimiento (el argumento no puede usarse fuera de su procedimiento). Las variables declaradas dentro de un procedimiento sólo existen dentro del procedimiento.

```
PROC main ()
  My_procedure 2,"Hello", TRUE;
ENPROC
PROC my_procedure (num nbr_times, string text, bool flag)
  ...
ENDPROC
```

Figura 2. Ejemplo de procedimiento main con llamada a my_procedure.

Otros tipos de rutina que se pueden crear con RAPID son las funciones que devuelven un parámetro como salida, y las interrupciones, que se ejecutan cuando una condición se vuelve verdadera.

Tipos de dato:

Dentro de un programa RAPID los datos se pueden definir de tres formas diferentes:

- Constantes (CONST) que representan valores fijos a los que no se les puede reasignar un nuevo valor.
- Variables (VAR) a los que se les puede asignar un nuevo valor durante la ejecución de un programa.
- Persistentes (PERS), se trata de variables en las que cada vez que se cambia su valor durante la ejecución del programa, también se cambia el valor de su inicialización.

Existen muchos tipos de datos en RAPID, pero en esta práctica, además de los tipos *num*, *string* y *bool* presentes en prácticamente todos los lenguajes de programación, sólo se prestará atención a los siguientes:

- *pos* Es un tipo de dato que contiene tres valores numéricos de posición *(X,Y,Z)*.
  Ejemplo de definición: *VAR pos pos1:=[600, 100, 800]*
  Ejemplo de acceso a su componente *Z*: *pos1.z*
- *orient* Este dato contiene la orientación de la herramienta expresada en forma de cuaternión *(q1,q2,q3,q4)*.
- *pose* El tipo de dato *pose* está compuesto de un dato *pos* y otro *orient*.
  Ejemplo de definición: *VAR pose pose1:= [[600, 100, 800], [1, 0, 0, 0]];*
- *robtarget* Este tipo de dato representa una localización objetivo del efector final y se compone de cuatro partes:
  - un dato tipo *pos*, de nombre *trans*, que son las coordenadas del punto *X,Y,Z*.
  - un dato tipo *orient*, de nombre *rot*, que define la orientación del punto.
  - un dato tipo *confdata*, de nombre *robconf*, que define la configuración del manipulador para alcanzar el punto.
  - un dato tipo *extjoint*, de nombre *extax*, que especifica posiciones de hasta 6 ejes adicionales. Se usa el valor 9E9 si no hay ningún eje adicional.

Control de flujo del programa:

Los programas no siempre se deben ejecutar secuencialmente, sino que es posible controlar qué código se ejecuta, en qué orden y cuántas veces. Para ello se dispone de distintas estructuras de control de flujo similares a las que poseen otros lenguajes de programación. Se pueden destacar:

- IF *condición* THEN
  .....
  ENDIF
- FOR *variable* FROM *valor_inicial* TO *valor_final* DO
  .....
  ENDFOR
- WHILE *condición* DO
  .....
  ENDWHILE

Algunas funciones básicas:

Existen muchas funciones predefinidas en RAPID, algunas de propósito general, como pueden ser funciones matemáticas, y otras relacionadas directamente con el control de manipuladores o manejo de señales. Aquí se van a destacar algunas de utilidad para la presente práctica:

- *MoveL*: Instrucción que mueve el robot **linealmente** desde su posición actual hasta la posición especificada. Su sintaxis es: *MoveL ToPoint, Speed, Zone, Tool \WObj;* Los parámetros indican lo siguiente:
  - *ToPoint*. Punto de destino de tipo *robtarget*.
  - *Speed*. Velocidad de movimiento, definida por un dato tipo *speeddata*. Existen un buen número de valores predefinidos como *v5* (5 mm/s), *v100* (100 mm/s), *vmax* (velocidad máxima del robot), …
  - *Zone*. Especifica una zona de esquina definida por una constante del tipo de dato *zonedata*. Existen muchos valores predefinidos como, por ejemplo, *fine* (el manipulador se sitúa en la posición especificada de forma exacta), *z5* (pasará a menos de 5mm del punto especificado, ver Figura 3), *z10* (pasará a menos de 10mm), …
  - *Tool*. Especifica la herramienta utilizada por el robot, definida por una variable del tipo de dato *tooldata*.
  - *\WObj*. El sistema de coordenadas que la instrucción de movimiento debe utilizar, se especifica con el argumento opcional *\WObj*.

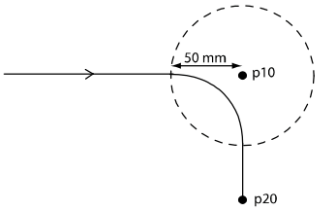


Figura 3. Parámetro *zone*. Fuente: *Manual del operador - Introducción a RAPID* de ABB.

- *MoveJ*: Instrucción que mueve el robot rápidamente de forma articular (sin seguir línea recta) desde su posición actual hasta la posición especificada. Su sintaxis es: *MoveJ ToPoint, Speed, Zone, Tool \WObj;*
- *Offs*: Función que añade un offset a una posición del robot, expresada en relación con el objeto de trabajo en que está expresado el punto. Su sintaxis: *Point2 := Offs (Point, x, y, z)*. El efecto

que tiene esta función es realizar un desplazamiento *(x,y,z)* sobre el punto *Point* para obtener un segundo punto *Point2*.

- *SetDO*: Asigna un valor determinado a una señal digital de salida. Su sintaxis es *SetDO signal, valor;*
- *TPReadNum*: Pide un valor numérico por pantalla del FlexPendant y lo asigna a una variable.

PARTE 2. EJERCICIO DE LA PRÁCTICA

En esta práctica se pretende construir una estación de trabajo que realice el montaje de 3 torres. El robot añadirá una pieza a la torre que se le indique introduciendo un número por pantalla. Esta estación de trabajo estará compuesta por un manipulador y una herramienta tipo ventosa, situados sobre una mesa, dos bases sobre las que se recogerán y se situarán las piezas que se irán generando y colocando de forma organizada.

La Figura 4 muestra la estación de trabajo que se debe construir y programar para que se comporte tal y como se puede visualizar en el vídeo disponible en el campus virtual de la asignatura. A continuación, se definen los pasos que deben realizarse para completar la aplicación.

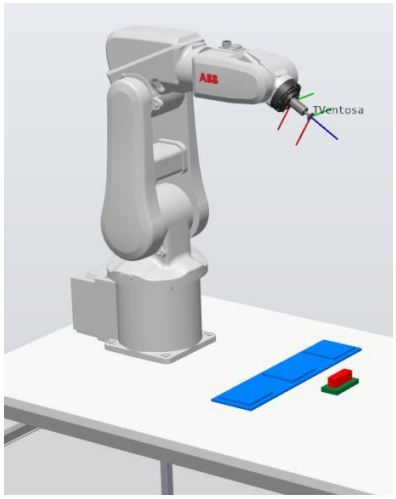


Figura 4. Montaje de tres Torres.

1. Creación de los elementos de la estación:
   - Descargar del campus virtual la estación de trabajo con el nombre *Practica4_inicio*.
   - Crear *Base0*, tetraedro con dimensiones *(40,80,10) mm*, de color azul. Su posición respecto al mundo será *(450, -40, h)* siendo *h* la altura de la mesa. Esta Base deberá estar conectada a la mesa, de manera que, al desplazarse, lo haga también la *Base0*.
   - Crear Base, tetraedro con dimensiones *(100,400,5) mm*, de color azul. Su posición respecto al mundo será *(290, -200, h)*.
   - Crear *Base1*, tetraedro con dimensiones *(80,80,5) mm*, de color azul. Su posición respecto al mundo será *(300, -190, h+5)*. Esta *Base1* se deberá conectar a *Base*.
   - Crear *Base2*, tetraedro con dimensiones *(80,80,5) mm*, de color azul. Su posición respecto al mundo será *(300, -40, h+5)*. Esta *Base2* se deberá conectar a *Base*.
   - Crear *Base3*, tetraedro con dimensiones *(80,80,5) mm*, de color azul. Su posición respecto al mundo será *(300, 110, h+5)*. Esta *Base3* se deberá conectar a *Base*.
   - Crear *Pieza*, tetraedro con dimensiones *(60, 20, hPieza) mm*, de color rojo y se ubicará en la posición central de la *Base0*. La *Pieza* deberá conectarse a *Base0*. El sistema de referencia local de la *Pieza* se situará en centro de su cara superior, con el eje *Z* hacia abajo, puesto que será la

orientación que tomará la herramienta del manipulador cuando la coja. La dimensión hPieza habrá que medirla.

- Crear un objeto de trabajo para la *Pieza* y otro para la *Base* con nombre *WO_Pieza* y *WO_Base respectivamente, después* conectarlos a su objeto correspondiente. Los sistemas de coordenadas de usuario y objeto de cada uno de los objetos de trabajo se dispondrán tal y como se muestra en la Figura 5.

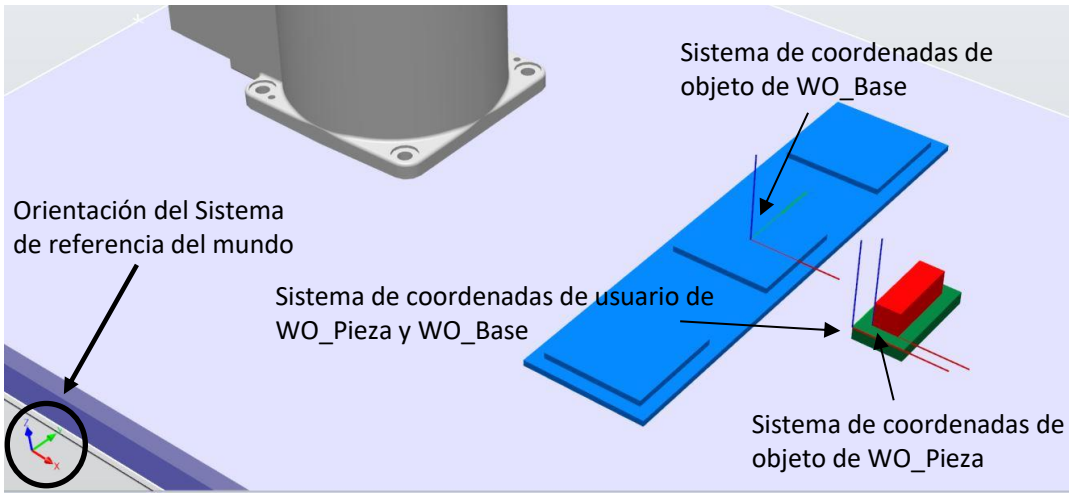


Figura 5. Ubicación de los Objetos de trabajo.

2. Definición de puntos. Con el objetivo de disponer de los puntos de referencia necesarios para poder diseñar la aplicación RAPID, se procederá de la siguiente manera:
   - Crear un punto Jhome. Posición Inicial → Punto → Crear posición de ejes. Editar ejes del robot (0,0,-90,0,0,0). Esta es en la posición que deberá acabar el robot cuando termine el programa.
   - Crear un punto en la posición de reposo del manipulador (cercano al punto de recogida), otro en el punto de recogida de la pieza y otros tres más, situados donde se depositarán la primera pieza de cada una de las torres (sobre el centro de *Base1, Base2 y Base3*), girado 90º respecto el eje z. Cada uno de estos puntos se definirá respecto al *WorkObject* más conveniente.

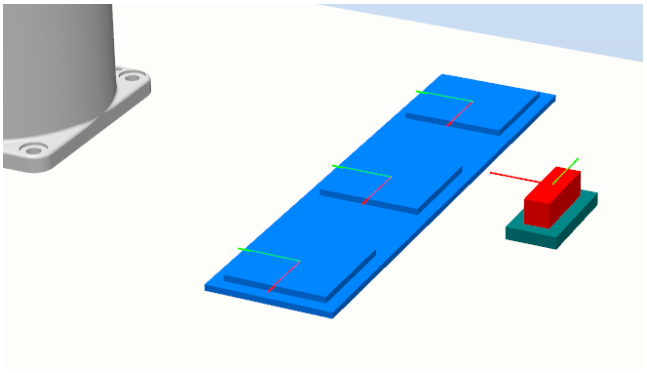


Figura 6. Ubicación de los puntos.

   - Crear una trayectoria que contenga estos seis puntos para que, posteriormente, aparezcan definidos en RAPID. Si no se crea ninguna trayectoria, al sincronizar, no aparecerán definidos.
   - Sincronizar con RAPID.
   - Guardar el estado inicial de la simulación para poder volver a él: *Simulación* → *Restablecer* → *Guardar estado actual.*

3. Creación del programa RAPID. El programa generado por la sincronización deberá ser modificado para que realice la tarea. Para ello, se deberán realizar las siguientes tareas:
   - ❑ Creación de las siguientes señales de entrada y salida:
     - Señal digital de salida *DTool*, que será la encargada de activar y desactivar la ventosa para coger y soltar la pieza.
   - ❑ Se crearán los siguientes procedimientos:
     - Procedimiento *Initialize*: en él se inicializan las variables que se creen (si es necesario).
     - Procedimiento *Pick*: debe tener un parámetro de entrada que indique el punto al que se debe desplazar el robot para coger la pieza, pasando por un punto de aproximación. El movimiento deberá ser, tal y como indica la Figura 7: Punto de aproximación → *Punto destino* →Punto de aproximación. Cuando el manipulador esté en el punto destino, deberá activar la ventosa para coger la pieza.
     - Procedimiento *Place*: debe tener un parámetro de entrada que indique el punto al que se debe desplazar el robot para soltar la pieza, pasando por un punto de aproximación. El movimiento deberá ser, tal y como indica la Figura 7: Punto de aproximación → *Punto destino* →Punto de aproximación. Cuando el manipulador esté en el punto destino, deberá activar la ventosa para soltar la pieza.

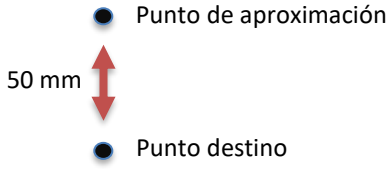
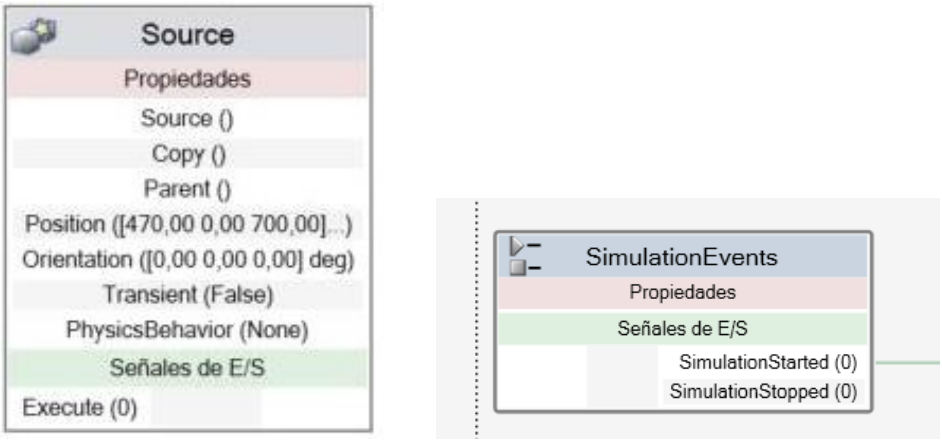


Figura 7. Aproximación a los puntos en los procedimientos Pick y Place.

     - Procedimiento *Main*, deberá cumplir con los siguientes requerimientos:
       - En primer lugar, deberá llamar al procedimiento de inicialización de variables.
       - Debe comenzarse en la posición de reposo y al terminar la ejecución, en la posición Jhome.
       - Se mantendrá ejecutándose el ciclo del manipulador hasta que se coloquen 10 piezas.
       - Se deberá crear una variable numérica donde se almacenará mediante la función TPReadNum, la posición en la que se desea colocar la pieza.
       - El procesamiento consistirá en coger la pieza, depositarla en la torre seleccionada y actualizar la variable donde debe soltarse, teniendo en cuenta la nueva altura.
   - ❑ Comprobar la sintaxis del programa y simularlo para comprobar que la ejecución es correcta.

4. Configurar la lógica de la estación para que genere piezas continuamente y la herramienta pueda cogerlas y soltarlas.
   - ❑ Para que se generen copias de las piezas, se deberá incluir a la lógica un componente denominado *Source* (Figura 8.a), que deberá ejecutarse cada vez que se suelte la pieza anterior (es decir, cuando se ejecute el componente *dettacher*). Para que inicialmente aparezca la primera pieza, también habrá que indicarle que se ejecute cuando comience la simulación, lo que se consigue añadiendo un componente denominado *SimulationEvents* (Figura 8.b)



(a) (b)

Figura 8. Generación de copias del objeto Pieza.

- La lógica para coger y soltar piezas será similar a la de la práctica anterior solo que, en este caso, la pieza que se debe coger y soltar no es el objeto original sino las copias que se van creando, lo que se puede definir directamente en la lógica tal y como aparece en la Figura 9.

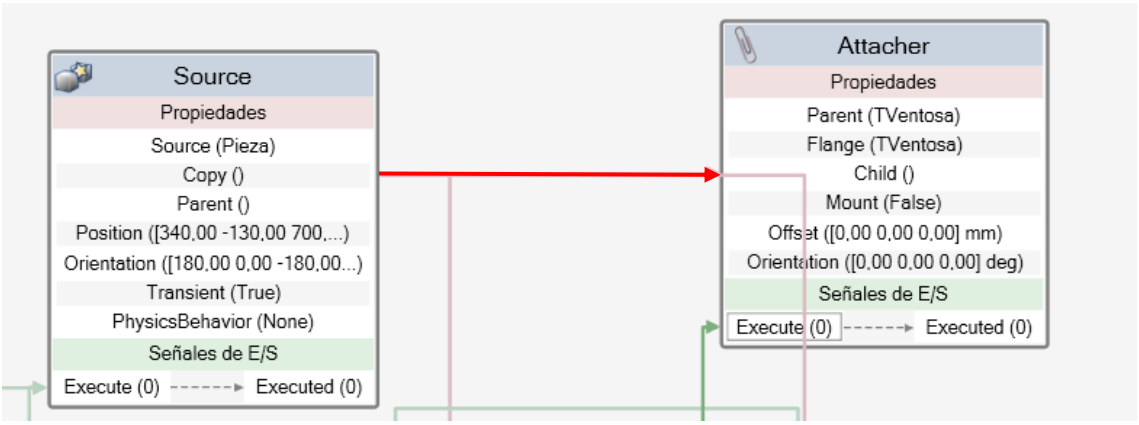


Figura 9. Configuración del attacher y dettacher.

5. Simular el sistema.
   - Ocultar el objeto Pieza para que no sea visible durante la simulación.
   - Mostrar el panel de Entradas y Salidas.
   - Simular el sistema con el modo *un solo ciclo*.