Departamento de Ingeniería de Sistemas y Automática

Universidad Politécnica de Valencia

Programación del sistema de simulación gráfica de robots *VRS*

Martin Mellado Arteche

Escuela Técnica Superior de Ingeniería Informática

Octubre, 2009

Programación del sistema de simulación gráfica de robots *VRS*

1. Introducción

El objetivo de este documento es presentar un tutorial para introducir al lector en la programación en C del sistema de simulación y programación *off-line* de robots denominado *Virtual Robot Simulator* (*VRS*) y el posterior planteamiento de programación de aplicaciones industriales típicas. En la parte inicial del tutorial se muestra como se puede realizar la programación de robots en C dentro del entorno Microsoft Visual Studio 2008 ® para C++ usando la librería específica para dicho propósito, **Virtual Robot External Access Library** (***VREAL***).

El objetivo de esta práctica será generar programas ejecutables externos a *VRS* que se podrán comunicar con *VRS* mediante el acceso a las funciones de la librería *VREAL*. Con esta librería la aplicación externa puede realizar acciones en *VRS*. *VREAL* está explicada en el documento **VREAL**, aunque la mayoría de las funciones que se van a utilizar aparecen en el documento resumido **VREAL Summary**. Las aplicaciones externas pueden ser ejecutadas desde el operativo o desde la opción de menú **File>>Run Application** de *VRS*.

Es importante resaltar que este guión no pretende ser una guía del entorno Microsoft Visual Studio 2008 ® para C++, por lo que sólo se mostrará lo necesario para realizar un programa basado en un diálogo con unos botones básicos. Si el lector ya está familiarizado con el entorno, podrá realizar directamente las tareas de la sección 2 y pasar a la sección 3.

2. Creación de una aplicación basada en diálogos

2.1. Creación de un proyecto

Para mostrar las posibilidades de programación de *VRS* en este tutorial se trabajará en el entorno de programación Microsoft Visual Studio 2008 ® para C++, si bien es posible realizarlo con cualquier otro que permita el acceso a las librerías suministradas con *VRS*. En concreto se mostrará cómo crear un programa basado en MFC (**Microsoft Fundation Classes**) como un diálogo con botones.

En este entorno de programación, para crear una aplicación basada en diálogo, se debe crear un proyecto de la siguiente forma:

- Ejecutar Microsoft Visual Studio 2008 ® para C++ .
- Seleccionar la opción **Archivo>>Nuevo>>Proyecto..** y aparece el diálogo de la Figura 1.
- Seleccionar el **tipo de proyectos** como **Visual C++ y MFC**.
- Seleccionar como plantilla la de **Aplicación MFC**.
- Introducir el nombre deseado, por ejemplo **Tutorial** (automáticamente aparece como nombre de la solución).
- Introducir como ubicación el directorio siguiente:
  C:\Archivos de programa\VIRTUALROBOT\VRS\SourceCode\Users
- Pulsar **Aceptar**. Aparecerán diálogos para los cinco pasos de creación del proyecto, pudiendo pasar entre ellos mediante los botones **Siguiente** y **Anterior**.

- Para que la aplicación sea basada en diálogo se debe activar la opción **Basada en cuadros de diálogo** en el paso 2 (Tipo de aplicación) .
- El resto de parámetros de los otros diálogos se pueden dejar por defecto, terminando el último diálogo con **Finalizar.**

A partir de entonces se dispone de un proyecto vacío pero con una plantilla de proyecto que ha creado algunas clases. La información de los proyectos se guarda en ficheros de solución (**sln**). Conviene saber que para abrir una solución se puede hacer desde Visual con la opción **Archivo>>Abrir>>Proyecto o solución...** o realizando doble *click* sobre un fichero de extensión **sln**.

En la zona de la izquierda se dispone de ventanas para ver:

- los ficheros del proyecto con el **Explorador de soluciones.**
- las clases del proyecto con la **Vista de clases**.
- las propiedades del proyecto con el **Administrador de propiedades**.
- los recursos del proyecto con la **Vista de recursos** .

En la zona de la derecha se pueden seleccionar por pestañas los ficheros en edición. En la zona inferior se verán los resultados de compilación, depuración, búsquedas, etc.

Se habrán creado automáticamente al menos dos clases, las clases aplicación y diálogo del proyecto (terminadas en **App** y **Dlg**), tal como se puede ver con la **Vista de clases**. Pinchando sobre el signo + de cada clase se ven los métodos y atributos creados para cada clase. Igualmente se habrán creado varios ficheros de cabecera (h) e implementación (cpp) para las clases y ficheros de recursos para la aplicación (rc, rc2, ico) tal como se puede ver con el **Explorador de soluciones**.

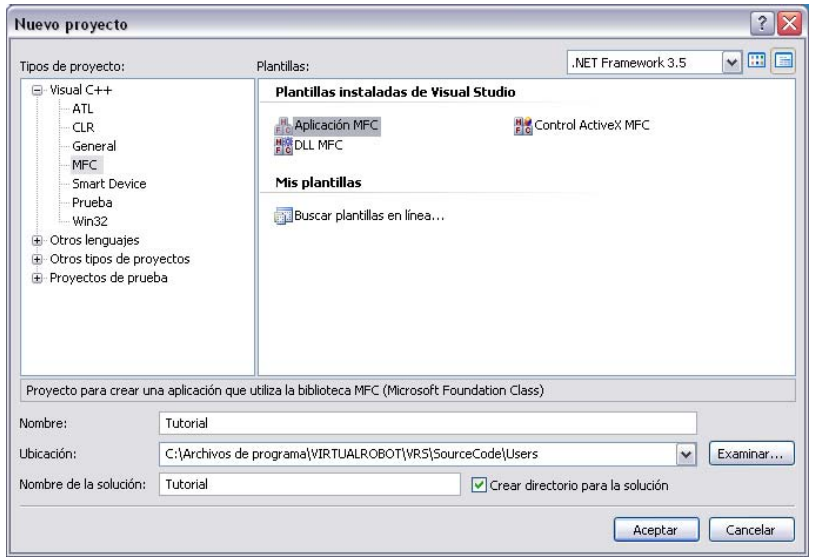


Figura 1. Diálogo para la Creación de un Proyecto en Microsoft Visual Studio 2008 ® para C++.

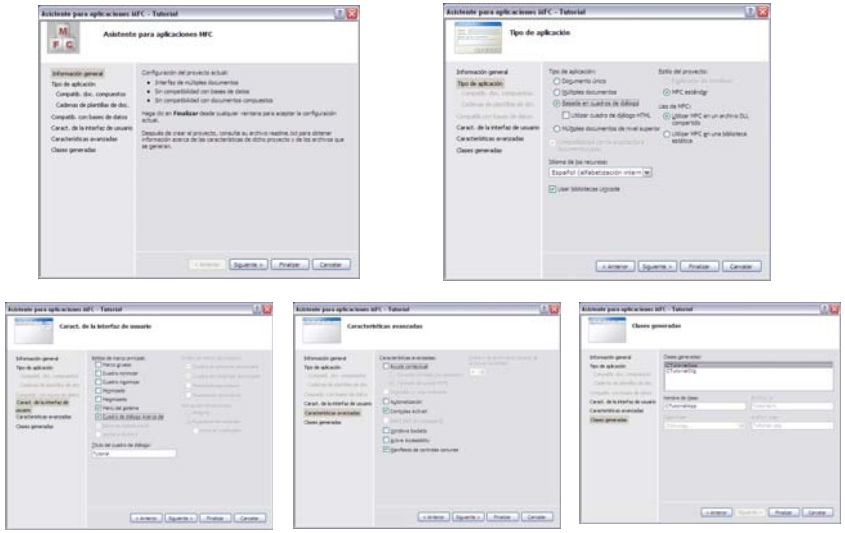


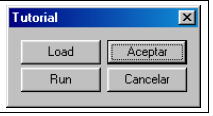
Figura 2. Diálogos de los cinco pasos para la creación de un proyecto.

Tarea 2.1:
Crea un proyecto con las siguientes características:
- Usa el *path* de la carpeta **SourceCode\Users** de *Virtual Robot*[1].
- Fija el nombre del proyecto deseado, por ejemplo **Tutorial**.
- Ajusta los parámetros tal como se muestra en la Figura 2 (es importante usar las MFC como DLL compartida si se va a usar la librería **VRStdIO**).

2.2. Definición del interfaz de la aplicación

En la vista de los recursos muestra en la parte derecha un diálogo prácticamente vacío, ya que sólo dispone de un texto y los botones **Aceptar** y **Cancelar**. Para introducir elementos en un diálogo se cuenta con el **Cuadro de herramientas** (ventana flotante a la derecha del todo). De aquí se pueden arrastrar los botones de una aplicación. Pulsando con el botón derecho del ratón sobre un elemento del diálogo y seleccionando la opción **Propiedades** del menú, se puede configurar el elemento, por ejemplo, cambiando el texto que aparece con **Caption en Apariencia**, o cambiando su identificador con **ID en Varios**. Cualquier elemento del diálogo o éste mismo, se puede mover o cambiar su tamaño (incluso borrar) seleccionándolo y ajustándolo con el ratón. Por ejemplo, el texto "A HACER: Colocar aquí controles de cuadro de diálogo." se debe borrar.

Tarea 2.2:
Diseña el interfaz de la aplicación como el que se muestra a la derecha con los siguientes identificadores:
- IDC_BUTTON_LOAD para el botón Load
- IDC_BUTTON_RUN para el botón Run



2.3. Enlace entre interfaz e implementación: Message Maps

Para enlazar el interfaz con la implementación de la clase, se puede pulsar el diálogo con el botón derecho y seleccionar la opción **Agregar controlador de eventos…** que abre el

[1] El *path* correcto depende del especificado en la instalación de *VRS*, siendo el que se utiliza por defecto en la versión 6.6 el siguiente: "C:\Archivos de programa\VIRTUALROBOT\VRS".

Asistente para controladores de eventos. Para cada botón existente en el diálogo se debe asociar un mensaje para crear una función miembro, de forma que su implementación se ejecute al pulsar el botón. Para ello, se selecciona como tipo de mensaje de botón pulsado (**BN_CLICKED**) indicando un nombre del controlador de funciones y pulsando **Agr./Editar**. El asistente creará la función miembro de la clase con un comentario indicando que se debe rellenar la implementación (// TODO:).

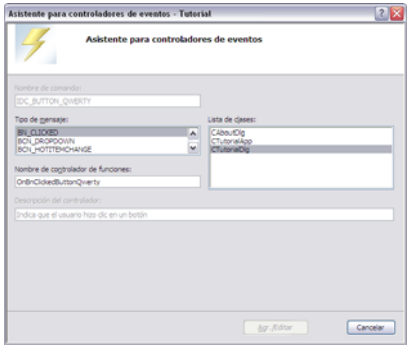


Figura 3. Diálogo del Asistente para controladores de eventos

Tarea 2.3:
En la clase **CTutorialDlg** asocia mensajes **BN_CLICKED** a los botones **Load**, **Run**, **Cancel** y **OK** admitiendo los nombres propuestos (**OnButtonLoad**, **OnButtonRun**, **OnCancel** y **OnOK**).

2.4. Implementación de las funciones miembro

Tal como ya se ha comentado, seleccionando la pestaña **Class View** se verán las clases existentes y las funciones y variables miembro de cada clase. Pulsando dos veces sobre una función miembro de una clase se abrirá el fichero de implementación de la clase donde está el código de implementación de la misma y se podrá editar.

Tarea 2.4:
Observa que está vacía la implementación de las cuatro funciones miembro creadas en la clase **CTutorialDlg** (**OnButtonLoad**, **OnButtonRun**, **OnOK** y **OnCancel**).

2.5. Definición de variables miembro

Para definir una variable miembro de una clase asociada a un diálogo se puede pinchar el diálogo con el botón derecho y elegir la opción **Agregar variable…** que abrirá el **Asistente para agregar variables miembro** (Figura 4). En este diálogo se debe meter el **Acceso**, el **Tipo de variable** y el **Nombre de variable**.

También se puede acceder al mismo asistente, en **Vista de clases**, seleccionando la clase deseada con el botón derecho del ratón y eligiendo la opción **Agregar>>Agregar variable…**

Cualquier variable agregada lo será en el fichero cabecera de la clase, donde evidentemente también se podrá incluir nuevas definiciones.

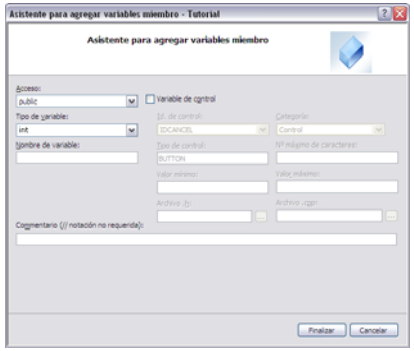


Figura 4. Diálogo del Asistente para agregar variables miembro

Tarea 2.5:
Define una variable miembro de la clase **CTutorialDlg** de tipo entero con nombre **m_robot**.

2.6. Añadir ficheros al proyecto

Si se desean añadir ficheros al proyecto, cosa necesaria para incluir librerías (ficheros **lib**), se debe seleccionar la opción **Projecto>>Agregar elemento existente…** y añadir el fichero deseado. También se pueden añadir a un proyecto ficheros cabecera y de implementación para tener acceso rápido a su contenido, pero no se usarán a menos que se realicen *includes* en los ficheros fuente.

Tarea 2.6.:
Añade al proyecto el fichero **VReal.lib** disponible en **SourceCode\Libs** (seleccionar en el campo **Tipo de Archivos: Library Files (.lib)** para poder verlo).

2.7. Compilación, linkado y ejecución de la aplicación

Para compilar y *linkar* la aplicación se dispone de las opciones correspondientes en el menú **Build**. El acceso más rápido es **F7** para compilar y montar la aplicación e incluso **F5** para además ejecutar la aplicación en modo depuración. Cuando se compila y monta una aplicación, se salvan automáticamente todos los ficheros. En la ventana de salida (**Output**) situada inicialmente en la parte inferior, se mostrarán los mensajes de compilación. Si hay algún error, pulsando sobre dos veces el mismo se accede a la línea que lo ha producido.

Tarea 2.7:
Pulsa **F5** para compilar, montar y ejecutar la aplicación. Si no hay errores, se arrancará y se podrá salir con **Aceptar** o **Cancelar**, sin realizar ninguna acción al pulsar **Load** o **Run**.

2.8. El ejecutable de la aplicación

La ejecución dentro del entorno Visual C es cómoda al realizar pruebas o para depurar los programas[2]. Una vez se dispone de la aplicación funcionando de forma correcta, el fichero ejecutable estará disponible en la carpeta **Debug** del proyecto creado. Se recomienda que las aplicaciones creadas se copien a una carpeta personal creada en **Applications\Users** para poder ejecutarlas cómodamente fuera del entorno Visual, con lo que además funcionarán más rápido.

[2] El entorno Visual dispone de facilidades para la depuración de los programas en el menú Build.

3. Desarrollo de una aplicación externa a VRS

Como ya se ha comentado, las aplicaciones externas a *VRS* hacen uso de la librería *VREAL*. Por tanto, conviene, antes de seguir, leer al menos las secciones 1-8 del documento VREAL (o VREAL Summary).

3.1. Inicialización y cierre de la librería

Como se puede observar, la librería hay que inicializarla para establecer comunicación con *VRS* y cerrarla para terminar la comunicación. Esto se realiza con las funciones **`alInitialize`** y **`alClose`**. Como la comunicación se va a realizar de forma local, a la función **`alInitialize`** no se le debe de pasar ningún parámetro. Ambas funciones, como todas las funciones de la librería *VREAL*, devolverán **`RET_OK`** si han funcionado correctamente y otro valor si se ha producido algún error.

Todas las funciones creadas con los **Asistentes** disponen de comentarios del tipo `// TODO:` donde se deben añadir las líneas de código de implementación de la función correspondiente. La inicialización de la librería se debe realizar en la función **`OnInitDialog`** mientras que la librería se debe cerrar tanto en la función **`OnOK`** como en la función **`OnCancel`**[3]. En todos los casos se debe comprobar que no se produce error, mostrándose un mensaje en caso contrario, si bien para el caso de cerrar o cancelar no resulta necesario ya que la aplicación va a terminar.

Tarea 3.1:
Implementa la función **`OnInitDialog`** de la siguiente forma:

```
BOOL CTutorialDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here
	if (alInitialize() != RET_OK)
	{
		AfxMessageBox((CString)"Error Initializing VReal");
		exit(-1);
	}
	return TRUE;  // return TRUE  unless you set the focus to a control
}
```

Tarea 3.2:
Implementa las funciones **`OnOK`** y **`OnCancel`** de la siguiente forma:

```
void CTutorialDlg::OnCancel()
{
	// TODO: Add extra cleanup here
	alClose();
	CDialog::OnCancel();
}

void CTutorialDlg::OnOK()
{
	// TODO: Add extra validation here
	alClose();
	CDialog::OnOK();
}
```

[3] En la aplicación inicial del tutorial las funciones **OnOK** y **OnCancel** realizarán lo mismo, pero puede haber aplicaciones en que no sea así.

Para que la compilación se realice correctamente, hay que incluir el fichero cabecera de *VREAL*. Hay que asegurarse de usar el *path* correcto en los ficheros cabecera, lo que se puede comprobar pulsando con el botón derecho sobre el nombre del fichero y seleccionando la opción **Open Document "..."**. Para que el *link* se realice correctamente, el proyecto debe tener el fichero **VREAL.LIB** tal como se realizó en la **Tarea 2.6**.

Tarea 3.3:
Incluye el fichero **vreal.h** de la siguiente forma:

```
// TutorialDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Tutorial.h"
#include "TutorialDlg.h"
#include "../../../Includes/VReal/VReal.h"
```

Si ahora se compila, monta y ejecuta el programa con **F5** mostrará mensajes de error si *VRS* no está arrancado, o actuará de forma correcta si está en ejecución y no hay problemas de comunicación.

Tarea 3.4:
Sin tener en ejecución *VRS*, ejecuta el programa con **F5**.
Con *VRS* en ejecución, ejecuta el programa con **F5**.

3.2. Carga de robots

Leer las secciones 9 y 10 del documento VREAL. Todas las funciones de estas secciones corresponden a acciones que se pueden realizar en *VRS Loader*, por lo que es fácil intuir su resultado.

Se desea que la aplicación cargue un robot con el botón **Load**. Para ello hay que implementar la función **`OnButtonLoad`** haciendo uso de la función **`alLoadRobot`**. Esta función devolverá el identificador del robot en una variable entera (se puede utilizar la creada en la **Tarea 2.5**). Para que se borre todo lo cargado en *VRS* antes de cargar el robot, se usa la función **`alCloseAll`**.

Tarea 3.5:
Implementa la función **`OnButtonLoad`** de la siguiente forma:

```
void CTutorialDlg::OnButtonLoad()
{
	// TODO: Add your control notification handler code here
	if (alCloseAll() != RET_OK)
		AfxMessageBox((CString)"Error Closing All");
	if (alLoadRobot("Models/Tutorial/CRS A465.rkf",&m_robot) != RET_OK)
		AfxMessageBox((CString)"Error Loading Robot");
}
```

Prueba la aplicación y el efecto del botón **Load**.

Tarea 3.6:
Modifica la función **`OnButtonLoad`** para que cargue también la mesa del CRS y sitúe el robot encima de ella.
Prueba la aplicación y el efecto del botón **Load**.

3.3. Control de herramientas

Lee la sección 13 del documento VREAL para controlar herramientas.

Tarea 3.7:
Realiza el *include* del fichero **Robot Definitions.h** disponible en la carpeta **Includes** y observa las definiciones que contiene. Haciendo uso de ellas, implementa la función **`OnButtonRun`** para que el robot cambie el estado de la herramienta al pulsar el botón **Run** de la siguiente forma:

```
void CTutorialDlg::OnButtonRun()
{
	// TODO: Add your control notification handler code here
	if (alSetActiveToolFrame(m_robot,toolFrame1) != RET_OK)
		AfxMessageBox((CString)"Error in alSetActiveToolFrame");
	if (alSetActiveTool(m_robot,tool1) != RET_OK)
		AfxMessageBox((CString)"Error in alSetActiveTool");
	int ret = alSetToolStatus(m_robot,toolOFF);
	Sleep(500);
	ret += alSetToolStatus(m_robot,toolDot25);
	Sleep(500);
	ret += alSetToolStatus(m_robot,toolHalf);
	Sleep(500);
	ret += alSetToolStatus(m_robot,toolDot75);
	Sleep(500);
	ret += alSetToolStatus(m_robot,toolON);
	Sleep(500);
	ret += alSetToolStatus(m_robot,toolDot75);
	Sleep(500);
	ret += alSetToolStatus(m_robot,toolHalf);
	Sleep(500);
	ret += alSetToolStatus(m_robot,toolDot25);
	Sleep(500);
	ret += alSetToolStatus(m_robot,toolOFF);
	if (ret != RET_OK)
		AfxMessageBox((CString)"Error in alSetToolStatus");
}
```

En el código anterior se ha aprovechado que `RET_OK` está definido como cero, pudiendo detectarse condiciones de error cuando la suma del resultado de varias funciones no es `RET_OK`.

Aunque a las funciones se les puede llamar sin comprobar el valor que retornan, es función del programador controlar los errores que se pueden producir en las llamadas a la librería VREAL.

Tarea 3.8:
Copia el ejecutable tal como se comentó en la sección 2.8.
Ejecútalo desde *VRS* con la opción **File>>Run Application**.

Durante la ejecución de las aplicaciones externas a *VRS* se puede hacer uso de las opciones de visualización disponibles en *VRS* (*zoom*, punto de vista, ...) para observar mejor el efecto de los programas.

4. Movimientos de robot en VRS

Una vez se conoce el funcionamiento básico de la programación, generación y ejecución de aplicaciones externas a *VRS*, se van a plantear una serie de ejercicios que se deben resolver con ayuda del propio *VRS* y del documento *VREAL* para conocer las funciones a llamar. Para la realización de cada ejercicio se puede usar el proyecto anterior, modificarlo o crear otro.

El objetivo de los ejercicios es comprender el uso de las funciones y sus parámetros, no generar aplicaciones concretas, por lo que se pueden modificar libremente las tareas de cada ejercicio si así se facilita su comprensión.

4.1. Movimientos de robot por articulaciones

Ejercicio 4.1:
Realiza una aplicación (cambiando el contenido de la función **`OnButtonRun`**) que:
- Inicialice el robot CRS A465 en su posición de sincronismo y con sistema de herramienta 1.
- Mueva el robot a varias configuraciones válidas obtenidas con *VRS*.
- Termine llevando el robot a su posición de sincronismo.

Usa las funciones **`alRobotReset`** y **`alMoveRobotJoints`**.
Para el uso de esta función habrá que definir un vector con:

```
double joints[NUM_DOF];
```

Si el robot tiene seis articulaciones, como es el caso del CRS A465, habrá que **asignar valores a los elementos `joints[0]` ... `joints[5]`** y llamar a la función con:

```
int ret = alMoveRobotJoints(m_robot,joints);
```

Ejercicio 4.2:
Amplia el ejercicio anterior para que:
- Realice los movimientos con diferentes velocidades.
- Dibuje la traza con diferentes colores en los movimientos.

Usa las funciones **`alSetRobotSpeed`**, **`alActiveTrace`** y **`alSetColorTrace`**.

4.2. Movimientos de robot en el Espacio Cartesiano

Ejercicio 4.3:
Realiza una aplicación (cambiando el contenido de la función **`OnButtonRun`**) que:
- Inicialice el robot CRS A465 en su posición de sincronismo y con sistema de herramienta 1.
- Mueva el robot a varias localizaciones válidas obtenidas con *VRS* con movimientos absolutos respecto al sistema origen de coordenadas. Haz que algunos movimientos sean punto a punto y otros en línea recta.
- Termine llevando el robot a su posición de sincronismo.

Usa la función **`alMoveRobot`** o la función **`alMove`**.

Ejercicio 4.4:
Modifica el programa anterior para usar:
- Movimientos relativos respecto al sistema origen de programación del robot.
- Movimientos absolutos y relativos respecto al sistema de herramienta.
- Movimientos absolutos y relativos respecto al sistema del mundo.

Ejercicio 4.5:
Implementa una función interna para el movimiento DEPART.
Opcionalmente implementa una función interna para el movimiento APPROX a una localización (puedes hacer uso de la librería **VRTransf** documentada en **VRAULS**).
Ambas funciones deben permitir controlar si el movimiento es punto a punto o en línea recta.
Implementa un programa que las utilice.

Ejercicio 4.6:
Realiza una aplicación que:
- Inicialice el robot CRS A465 en su posición de sincronismo y con sistema de herramienta 1.
- Realice todas los movimientos y acciones de herramienta necesarias para simular que coge una pieza del entorno y la deja en otro lugar.

Usa las funciones APPROX y DEPART de la tarea anterior.
Para conocer la localización de cogida de las piezas visualizar sus sistemas de coordenadas.

5. Programación de robots orientada a objetos en VRS

Ya se conoce que en el entorno se pueden distinguir:

- Objetos (*objects*): son fijos y el robot no puede actuar sobre ellos
- Piezas (*parts*): el robot los puede manipular, para lo que disponen de sistemas de coordenadas de operación sobre los que el robot puede realizar movimientos relativos e incluso agarrar las piezas.

En esta sección se va a explicar cómo se pueden programar estas acciones que están disponibles en *VRS PartHandling*.

5.1. Movimientos relativos a las piezas

Existen dos funciones que permiten realizar movimientos relativos a las piezas del entorno:

- **`alMoveToPart`**: Mueve un robot hasta conseguir que el sistema de herramienta activo quede en la misma localización que el sistema de coordenadas de operación de una pieza del entorno. El movimiento se puede realizar punto a punto o en línea recta.
- **`alApproxToPart`**: Mueve un robot hasta conseguir que el sistema de herramienta activo quede en la misma localización que el sistema de coordenadas de operación de una pieza del entorno pero con un desplazamiento en posición (x,y,z) relativo al sistema de coordenadas de operación de la pieza. El movimiento se puede realizar punto a punto o en línea recta.

Ejercicio 5.1:
Modifica el **Ejercicio 4.6** usando las funciones **`alMoveToPart`** y **`alApproxToPart`**.
Usa la pieza con identificador=1 y sistema de operación=1.

Ejercicio 5.2:
Modifica el ejercicio anterior usando la función **`alGetAvailableParts`** y las funciones disponibles en **VRStdIO** (documentada en **VRAULS**) para que solicite el nombre de la pieza a manipular y su sistema de operación. Usa el entorno **CRS Table.enf**.

5.2. Manipulación de piezas

Las piezas del entorno pueden ser manipuladas mediante dos funciones:

- **`alPickPart`**: Esta función hace que el robot capture una pieza según la relación entre el sistema de herramienta activo del robot y el sistema de operación de una pieza. Se puede fijar que la relación sea la identidad, de forma que sólo se coja la pieza si coinciden ambos sistemas (con cierta tolerancia). Este es el modo natural de trabajo cuando el robot se ha movido con la función **`alMoveToPart`**. Por otra parte se puede fijar que esta relación no sea idéntica, de forma que el robot coja la pieza en cualquier caso. Adicionalmente, la herramienta puede cambiar su estado. En definitiva, el efecto de la función es que el robot coge una pieza (normalmente cerrando una pinza o activando una ventosa) y la transporta consigo hasta que se ejecute la función **`alPlacePart`** sobre esta pieza.
- **`alPlacePart`**: Esta función hace que el robot libere una pieza que tenga cogida. Adicionalmente, la herramienta puede cambiar su estado. Por tanto, el efecto de la función es que la pieza queda libre del robot (normalmente se abrirá la pinza o desactivará la ventosa). El sistema no comprueba que la pieza quede en una situación estable, por lo que puede quedar suspendida en el aire si no se realizan movimientos adecuados.

Ejercicio 5.3:
Modifica el ejercicio anterior usando las funciones **`alPickPart`** y **`alPlacePart`**.

6. Conexión de señales entre robots

Los robots disponen de entradas y salidas tanto digitales como analógicas. Su uso está comentado en las *VRS Tools VRS TeachPendant* y *VRS IOConnection*.

Ejercicio 6.1:
Haz un programa que:

- Conecte la salida digital 1 del robot **CRS A465** a su entrada digital 1.
- Conecte la salida analógica 1 del robot **CRS A465** a su entrada analógica 1.
- Comprueba con el ***VRS TeachPendant*** el funcionamiento correcto actuando sobre las salidas y visualizando el estado de las entradas.

Ejercicio 6.2:
Modifica el programa anterior para que:

- Permita pedir al usuario el valor de las salidas digitales y analógicas conectadas. Puedes hacer uso de las funciones disponibles en **VRStdIO** (documentada en **VRAULS**).
- Comprueba con el ***VRS TeachPendant*** el funcionamiento correcto visualizando el estado de las entradas.

Ejercicio 6.3:
Haz un programa que:

- Cargue dos robots en *VRS*
- Conecte cuatro señales digitales de un robot al otro.
- Pida al usuario un valor entre 0 y 15 y el primer robot lo pase por las salidas digitales.
- El segundo robot teste el valor de las entradas y muestre su valor.

7. Comunicación entre aplicaciones mediante señales de robots

Tal como se ha visto en el ejercicio 6.2, es posible hacer una aplicación externa a *VRS* que controle a más de un robot. Esto no tiene un sentido realista, ya que la aplicación controlará de forma secuencial los movimientos y acciones que realicen los diferentes robots. Es decir, no moverá un robot hasta que los movimientos de otros robots hayan terminado.

Sin embargo, es posible realizar diferentes aplicaciones que se comuniquen mediante las señales conectadas entre los robots, de forma que tengan un sentido más realista, ya que cada aplicación realmente se ejecutaría sobre una unidad de control independiente. De esta forma, los movimientos se pueden realizar de forma solapada y establecer protocolos de comunicación entre los robots.

Ejercicio 7.1:
Realiza el ejercicio 6.3 con dos programas:

- Uno como maestro que realiza las conexiones, pida el valor al usuario y lo envíe por las salidas.
- Otro como esclavo que recibe las entradas y lo muestra al usuario.

Estable un protocolo que garantice el correcto funcionamiento independientemente de qué programa se arranca antes.

Ejercicio 7.2:
Mediante la ejecución en *VRS* de aplicaciones externas, ten arrancadas simultáneamente:

- Applications\Demos\Car Assembly\UpperTrack.exe.

Esta aplicación carga el entorno y las cintas transportadoras.

- Applications\Demos\Car Assembly\LeftKuka.exe.

Esta aplicación carga el robot Kuka de la izquierda.

- Applications\Demos\Car Assembly\RightKuka.exe.

Esta aplicación carga el robot Kuka de la derecha.
Hay tres aplicaciones en marcha que están sincronizadas mediante las conexiones entre los robots. Inicia sus procesos en el orden deseado.