ANEXO B. CÓDIGO FUENTE.

En este apartado se muestra el código fuente generado en el programa de desarrollo Microsoft Visual Studio C++ 6.0 para las dos grandes aplicaciones comentadas en este proyecto: 'Corte3D' y 'Rx90'. Por un lado, la primera obtiene y procesa toda la información para ejecutar una trayectoria de corte y establece una comunicación con el controlador CS7, por otro, la segunda simula en el ordenador al robot siguiendo la trayectoria obtenida entorno a una pieza diseñada previamente.

Desglosamos el código fuente, para una mayor facilidad en la lectura, en clases, y a su vez, éstas las dividimos en archivos de cabecera (extensión ".h") y de implementación (extensión ".cpp").

B.1 APLICACIÓN 'CORTE3D'.

B.1.1 CCorte3D.

B.1.1.1. Corte3D.h

```
//////////////////////////////////////////////////////////////////////
//                                                                  //
//      Corte3D.h: Archivo de cabecera del la clase principal       //
//      CCorte3D.                                                   //
//                                                                  //
//////////////////////////////////////////////////////////////////////
#if !defined(AFX_CORTE3D_H__4B0F6DF2_B5C3_461B_BF07_84ED9419B3FC__INCLUDED_)
#define AFX_CORTE3D_H__4B0F6DF2_B5C3_461B_BF07_84ED9419B3FC__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#ifndef __AFXWIN_H__
        #error include 'stdafx.h' before including this file for PCH
#endif
#include "resource.h"            // main symbols

#include "PuertoSerie.h"
#include "PuntoTrayectoria.h"

/////////////////////////////////////////////////////////////////////////////
// CCorte3DApp:
// See Corte3D.cpp for the implementation of this class
//

class CDlgSimulacion;
class CCorte3DApp : public CWinApp
{
public:
        void ComprobarPosiblesSaltos(int PuntosAPT);
        void PuntosDeSeparacion(int nNumeroPuntosAPT);
        void PuntosDeAproximacion();
        void CalculaPuntoCentral(int nPuntos);
        CPuertoSerie m_serie;
        CPuntoTrayectoria* m_pPuntoTrayectoria;
        CPuntoTrayectoria m_MundoReal;
        CPuntoTrayectoria m_MundoVirtual;
        CPuntoTrayectoria m_Transformacion;
        CCorte3DApp();
        void EnviarOrden(LPCTSTR lpszOrden);
        BOOL AbrirYLeerFicheroAPT(CString& strContFicheroAPT);
        BOOL ProcesarContenidoFicheroAPT(CString& strContFicheroAPT);

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CCorte3DApp)
        public:
        virtual BOOL InitInstance();
        //}}AFX_VIRTUAL
```

```
// Implementation

        //{{AFX_MSG(CCorte3DApp)
        // NOTE - the ClassWizard will add and remove member functions here.
        //    DO NOT EDIT what you see in these blocks of generated code !
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before // the previous line.
#endif // !defined(AFX_CORTE3D_H__4B0F6DF2_B5C3_461B_BF07_84ED9419B3FC__INCLUDED_)
```

B.1.1.2. Corte3D.cpp

```
////////////////////////////////////////////////////////////
//                                                        //
//      Corte3D.cpp: implementa la clase principal de la  //
//      aplicación.                                       //
//                                                        //
////////////////////////////////////////////////////////////
#include "stdafx.h"
#include "Corte3D.h"
#include "Corte3DDlg.h"
#include <string.h>
#include <math.h>
#include <iostream.h>
#include <fstream.h>
#define PI       3.1415926
#define pasos 5

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CCorte3DApp

BEGIN_MESSAGE_MAP(CCorte3DApp, CWinApp)
        //{{AFX_MSG_MAP(CCorte3DApp)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
        //}}AFX_MSG
        ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CCorte3DApp construction

CCorte3DApp::CCorte3DApp()
```

```
{
        // TODO: add construction code here,
        // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CCorte3DApp object

CCorte3DApp theApp;


/////////////////////////////////////////////////////////////////////////////
// CCorte3DApp initialization

BOOL CCorte3DApp::InitInstance()
{
        // Standard initialization
        // If you are not using these features and wish to reduce the size
        //  of your final executable, you should remove from the following
        //  the specific initialization routines you do not need.

#ifdef _AFXDLL
        Enable3dControls();             // Call this when using MFC in a shared DLL
#else
        Enable3dControlsStatic();// Call this when linking to MFC statically
#endif
        // Dialog creation may fail if the dialog template has a Rich Edit   //control in it because
        the Rich Edit control is not initialized.
        // Call the MFC global function AfxInitRichEdit before you create the //dialog to initialize
        the Rich Edit Control. A good place to initialize //the Rich Edit control is in the
        application's InitInstance function //before you create the dialog.
        AfxInitRichEdit();

        CCorte3DDlg dlg;
        m_pMainWnd = &dlg;
        int nResponse = dlg.DoModal();
        if (nResponse == IDOK)
        {
                // TODO: Place code here to handle when the dialog is
                //  dismissed with OK
        }
        else if (nResponse == IDCANCEL)
        {
                // TODO: Place code here to handle when the dialog is
                //  dismissed with Cancel
        }
        // Since the dialog has been closed, return FALSE so that we exit the
        //  application, rather than start the application's message pump.
        return FALSE;
}
/////////////////////////////////////////////////////////////////////////////
//
// Método que sirve para enviar órdenes al controlador.
//
// Parámetros:
```

```
//       - Puntero a CString constante (LPCTSTR) que almacena la orden.
//         Como no se va a modificar se pasa de esta forma y no como CString&.
//
//         PROBAR CSTRING& Y NO HAY Q DEFINIR strComandoAEnviar DENTRO
//
//       QUIZAS DEBERIAS ESTAR DENTRO DE LA CLASE CPUERTOSERIE
//
////////////////////////////////////////////////////////////////////////////
void CCorte3DApp::EnviarOrden(LPCTSTR lptszOrden)
{
        //char chRetornoCarro[1] = {'\r'};
        //char chRetornoCarro = '\r';


// Añadimos el caracter RC al final de la orden para que pueda ser
// interpretada por el controlador.
        CString strComandoAEnviar;
        strComandoAEnviar.Format("%s\r", lptszOrden);

// Enviamos los datos por el puerto serie carácter a carácter. Como sólo
// podemos enviar carácteres de 1 byte, hemos de convertir el CString, que
// es un LPCTSTR, a LPCSTR (es decir, de TCHAR* a char*). El método miembro
// Write de la clase CSerial admite o un void* pData o un LPCSTR.
//
// Paso la cadena de TCHAR a una cadena de char.
// Como está contenida en una instancia de la clase CString (que contiene
// TCHAR), hay q pasarla a LPCSTR (const char*) o LPSTR (char*) para poder
// enviarla caracter a caracter por el puerto serie. Mediante un cast solo
// podemos obtener un LPCTSTR. Usando el método miembro GetBuffer() podemos
// obtener un puntero LPTSTR, que debemos posteriormente liberar usando el
// método miembro ReleaseBuffer(). Hasta que no lo hayamos liberado no
// podemos usar ningún otro método miembro de esa instancia CString.
//
// Por tanto, hemos de diferenciar si estamos usando UNICODE o MCBS (Ansi)
// (#ifdef _UNICODE (...) #endif):
//       a)       UNICODE: un TCHAR contiene un wchar_t de 2bytes.
//                Por tanto, hacemos un cast y obtenemos un LPCTSTR que contiene
//                caracteres UNICODE (2 bytes por caracter) y no lo podemos usar
//                para transmitir por el puerto serie. Podemos usar las macros ATL
//                para conversión entre formatos de cadena (hemos de incluir
//                #include <atlconv.h> y poner la macro USES_CONVERSION al
//                función que las usa, pues en ella se definen algunas variables
//                principio de la locales necesarias).
//                En concreto usaremos T2A(LPTSTR xx), que devuelve un LPSTR, o
//                T2CA(LPTSTR xx), que devuelve un LPCSTR. Sin embargo, como vemos,
//                necesitamos un LPTSTR, por lo que debemos usar los métodos
//                miembro GetBuffer()      y ReleaseBuffer().
//       b)       MCBS: un TCHAR contiene un char (1 byte por caracter).
//                Hacemos un cast y obtenemos un LPCTSTR que contiene caracteres
//                MCBS (1 byte por caracter), es decir, es directamente un LPCSTR
//                (const char*) y lo podemos usar tal cual.

//       a) Convierto CString a LPCSTR

//       b) Convierto CString a LPCTSTR=LPCSTR
```

```
	LPCSTR lpszComandoAEnviar = strComandoAEnviar;

//	CAMBIAR EXPLICACION,XQ NECESITO LPSTR y no LPCSTR, asi q tng q usar
//	GetBuffer y releaseBuffer.
//	LPSTR lpszComandoAEnviar = strComandoAEnviar.GetBuffer(0);
// Realizamos una petición al hilo escritor para que envie los datos por el
// puerto serie abierto.
	m_serie.HacerPeticionEscritura(lpszComandoAEnviar);

	//	strComandoAEnviar.ReleaseBuffer();
}

////////////////////////////////////////////////////////////////////////////
//
// Método que sirve para abrir y leer el contenido del fichero APT.
//
// Parámetros:
//	- Referencia a CString para que guarde el contenido del fichero.
//
// Devuelve TRUE si no han existido errores y FALSE si ha habido.
// De esta forma no se ejecuta V_TRAJSIG si no es necesario.
//
////////////////////////////////////////////////////////////////////////////
BOOL CCorte3DApp::AbrirYLeerFicheroAPT(CString& strContFicheroAPT)
{
// Leyendo y escribiendo en un archivo de text ASCII:
// Si usamos Unicode o MBCS hemos de tener cuidado cuando escribamos en
// archivos de texto ASCII. La manera mas fácil y segura es usar la
// clase CStdioFile proporcionada por MFC. Sólo debemos usar
// la clase CString y los métodos miembros ReadString y WriteString:
//		CStdioFile file(..);
//		CString str = _T("Hola");
//		file.WriteString(str);
//
// Si hemos de usar la clase CFile forzósamente, lo haremos así:
//		CFile file(..);
//		CString str = _T("Hola");
//		file.Write(str,(str.GetLength()+1)*sizeof(TCHAR));
// Otro comentario importante es q siempre leamos y escribamos usando captura
// de excepciones try&catch(CFileException *e).

	BOOL fRes = TRUE;
// Mostramos un diálogo modal que nos permite seleccionar el fichero APT a
// abrir.
	CFileDialog dlgAbrirAPT(TRUE,"dxf",NULL,OFN_FILEMUSTEXIST|OFN_HIDEREADONLY,"Datos APT de
	CATIA (*.aptsource)|*.aptsource||");
	CString strNombreFichero;

	if(dlgAbrirAPT.DoModal() == IDOK)
	{
		strNombreFichero = dlgAbrirAPT.GetFileName( );
		CFile fileAPT;
		// Abrimos el fichero seleccionado.
		try
		{
```

```
                fileAPT.Open(strNombreFichero, CFile::modeRead);
        }
        catch (CFileException *e)
        {
                CString strError;
                strError.Format("Error al abrir el archivo. Error número %d",e->m_cause);
                e->Delete();
                AfxMessageBox(strError);
                return FALSE;
        }
        ofstream out("nombrepieza.txt");
        out << strNombreFichero;
        out << "\0";
        out.close();

        // Creamos un buffer temporal de lectura
        int nFileLength = fileAPT.GetLength();
        char *lpBuffer = new char[nFileLength +1];
        // Leemos todo el contenido del buffer del puerto serie y lo
        // almacenamos en nuestro buffer temporal
        try
        {
                fileAPT.Read(lpBuffer,nFileLength);
        }
        catch (CFileException *e)
        {
                CString strError;
                strError.Format("Error al leer el archivo. Error número %d",e->m_cause);
                AfxMessageBox(strError);
                e->Delete();
                delete[] lpBuffer;
                return FALSE;
        }

        // Añadimos un carácter NULL para terminar la cadena.
        lpBuffer[nFileLength] = '\0';

        // Cerramos el fichero.
        fileAPT.Close();

        // Almacenamos el contenido del fichero en una instancia de la
        // clase CString, más fácil de usar.
        strContFicheroAPT.Format(_T("%s"),lpBuffer);

        // Vaciamos y eliminamos el buffer temporal.
        delete[] lpBuffer;
    }
    else
        fRes =FALSE;
    return fRes;
}

////////////////////////////////////////////////////////////////////////////
//
// Método que sirve para procesar el contenido del fichero APT.
```

```
//
// Parámetros:
//       - Referencia a CString para que contiene el contenido del fichero APT,
//         que va a ser retocado dentro de la función (podría ser LPCTSTR xq no
//        es retocado)
// Devuelve TRUE si todos los puntos son alcanzables,y FALSE si alguno no lo
// es. De esta forma no se ejecuta V_TRAJSIG si no es necesario.
//
////////////////////////////////////////////////////////////////////////////
BOOL CCorte3DApp::ProcesarContenidoFicheroAPT(CString& strContFicheroAPT)
{
        int TotalPuntosAPT;
        int nIndex, nIndex2, nIndex3, nNumeroPuntosAPT, nLongitudLinea;
        CString strBuffer;
        BOOL fRes = TRUE;
        BOOL fRes2 = TRUE;

        // Contamos el número de puntos que hay en el fichero APT. Para ello
        // recorremos el objeto CString y vamos extrayendo las líneas donde hay
        // puntos y eliminado aquellas que no los contienen. Los puntos
        // extraidos se almacenan en strBuffer.
        for (nNumeroPuntosAPT = 0; ((nIndex = strContFicheroAPT.Find(_T("GOTO/"))) != -1);
        nNumeroPuntosAPT++)
        {
                        // Recorto la cadena hasta la primera coordenada del punto.
                //strContFicheroAPT = strContFicheroAPT.Right(strContFicheroAPT.GetLength() - nIndex
                - sizeof(_T("GOTO/")));
                strContFicheroAPT.Delete(0, nIndex + sizeof(_T("GOTO/")));

                // Contamos la longitud de la línea para poder extraerla.
                nLongitudLinea = strContFicheroAPT.Find(_T('\n'));

                //Extraemos las coordenadas del punto, separadas por coma. Cada //punto en una línea
                (cogemos tambien '\n').
                strBuffer += strContFicheroAPT.Left(nLongitudLinea+1);
        }

        if (nNumeroPuntosAPT==0)
                for (nNumeroPuntosAPT = 0; ((nIndex = strContFicheroAPT.Find(_T("GOTO  /"))) != -1);
nNumeroPuntosAPT++)
                {
                        // Recorto la cadena hasta la primera coordenada del punto.
                        //strContFicheroAPT = strContFicheroAPT.Right(strContFicheroAPT.GetLength()
                        - nIndex - sizeof(_T("GOTO/")));
                        strContFicheroAPT.Delete(0, nIndex + sizeof(_T("GOTO  /")));

                // Contamos la longitud de la línea para poder extraerla.
                        nLongitudLinea = strContFicheroAPT.Find(_T('\n'));

                //Extraemos las coordenadas del punto, separadas por coma. Cada //punto en una línea
                (cogemos tambien '\n').
                        strBuffer += strContFicheroAPT.Left(nLongitudLinea+1);
                }
        // Hacemos que las coordenadas de cada punto estén separadas por un solo // espacio en
        blanco,
```

```
        // pues el formato que tenia era: xXXX.XXXX, xXXX.XXXX, xXXX.XXXX ...
        // donde X es un digito
        // y x es el signo - si el número es negativo.
        strBuffer.Remove(' ');
        strBuffer.Replace(',', ' ');

// Eliminamos el primer punto del fichero APT, que no pertenece a la
// trayectoria.
//      // TENERLO EN CUENTA A LA HORA DE CALCULAR
//nNumeroPuntosTray!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!1
        nIndex = strBuffer.Find('\n') + 1;
        strBuffer = strBuffer.Right(strBuffer.GetLength() - nIndex);
        nNumeroPuntosAPT--;
//
// Ahora sabemos el número de instancias de la clase CPuntoTrayectoria que
// necesitamos. Declaramos el array donde almacenaremos los puntos
// definitivamente y los inicializamos.
        CCorte3DApp* pAplicacion = (CCorte3DApp*)AfxGetApp();

        // Añadimos unos puntos para hacer lamaniobra de aproximación.
        TotalPuntosAPT = nNumeroPuntosAPT+2*pasos;
        pAplicacion->m_pPuntoTrayectoria = new CPuntoTrayectoria[TotalPuntosAPT];

        // Para cada punto de la trayectoria...
        double dPuntoTrayectoria[6];
        CString strBuffer2;

        for (nIndex = 0; nIndex < nNumeroPuntosAPT; nIndex++)
        {
        // Calculamos la longitud de la línea para poder trabajar con ella.
                nLongitudLinea = strBuffer.Find(_T('\n'));

        // Recorremos cada línea y vamos almacenado las coordenadas en un array // double[6].
                for (nIndex2 = 0, nIndex3 = 0; nIndex2 < nLongitudLinea; nIndex2++)
                {
                        if ((strBuffer[nIndex2] != ' ') && (strBuffer[nIndex2] != '\r'))
                                strBuffer2 += strBuffer[nIndex2];
                        else
                        {
                                // Estamos en un caracter delimitador (bien el
                                // caracter ' ' o bien '\r').
                                // Por tanto, lo que tenemos hasta ahora es una
                                // coordenada completa y debemos
                                // convertirla en un double y pasarla al array
                                // temporal.
                                dPuntoTrayectoria[nIndex3] = _tcstod(strBuffer2, '\0');
                                strBuffer2.Empty();
                                nIndex3++;
                        }
                }

                // Eliminamos la línea procesada correctamente.
                //strBuffer = strBuffer.Right(strBuffer.GetLength() - nLongitudLinea);
                //DA ERROR TB
                strBuffer.Delete(0, nLongitudLinea + 1);
```

```
                // COMENTARIO ACLARATORIO EN MODO DE DEPURACIÓN
                TRACE(_T("\n\nzfree[1][%d]\n"),nIndex);

                // Hacemos las operaciones que haya que hacer en cada punto.
                // A) Rellenamos todas las coordenadas XYZvector.
                pAplicacion->m_pPuntoTrayectoria[nIndex+pasos].SetXYZvector(dPuntoTrayectoria);

                // B) Pasamos cada punto a su forma matricial.
                pAplicacion->m_pPuntoTrayectoria[nIndex+pasos].XYZvector2Matriz();

                if (nIndex==0)
                        PuntosDeAproximacion();
        }
        PuntosDeSeparacion(nNumeroPuntosAPT);

// Ahora ya tenemos cada punto del fichero APT expresado por medio de su
// matriz de transformación homogénea, todos ellos referidos respecto al
// origen del sistema de referencia WORLD del mundo virtual.

////////////////////////////////////////////////////////////////////////////

//      ofstream out("posiciones.txt");

// Ahora ya tenemos preparados todos los puntos de la trayectoria y podemos
// realizar sobre ellos las operaciones necesarias.
        for (nIndex = 0; nIndex < TotalPuntosAPT; nIndex++)
        {
                // COMENTARIO ACLARATORIO EN MODO DE DEPURACIÓN
                TRACE(_T("\n\nzfree[1][%d]\n"),nIndex);

                //1. Componemos cada punto respecto a la transformación wHwv.
                pAplicacion->m_pPuntoTrayectoria[nIndex].Componer(pAplicacion->m_MundoVirtual);


                //2. Resolvemos el modelo cinemático inverso en cada punto (con configuración -1,1,-
1)
                pAplicacion->m_pPuntoTrayectoria[nIndex].ResolverModeloCinematicoInverso(-1,1,-1);

//              for (nIndex2 = 0; nIndex2 < 6; nIndex2++)
//              {
//                      out << pAplicacion-
>m_pPuntoTrayectoria[nIndex].m_sCoordTheta.theta[nIndex2];
//                      out << " ";
//              }
//              out << ";";
//              out << "\n";

                //3. Comprobamos si el punto es alcanzable. Si no lo es, indicamos el error.

        // SOLO PARA COMPROBAR.SE HACE LUEGO SI LA TRAYECTORIA ES CORRECTA
                pAplicacion->m_pPuntoTrayectoria[nIndex].Matriz2XYZeulerZYZ();

                fRes = pAplicacion->m_pPuntoTrayectoria[nIndex].EsAlcanzable();
```

```
                if (fRes == FALSE)
                {
                        CString strError;
                        strError.Format(_T("El punto %d no es alcanzable."), nIndex+1);
                        AfxMessageBox(strError,MB_ICONEXCLAMATION|MB_OK);
                        fRes2 = FALSE;
                }
        }

        ComprobarPosiblesSaltos(TotalPuntosAPT);
//      out.close();
//      system("Rx90.exe");

        //      Lo siguiente sirve para escribir en fichero
        //      las matrices de cada punto de la trayectoria respecto al
        //      sistema world del robot, y hacer las comprobaciones necesarias
        //      en Matlab.

                ofstream tout("matrices.txt");
                for (int k=0;k<TotalPuntosAPT;k++)
                {
                        tout << "T(:,:," << k+1 << ")=[";
                        for (int i=0;i<4;i++)
                        {
                                for (int j=0;j<4;j++)
                                {
                                        tout << pAplicacion-
>m_pPuntoTrayectoria[k].m_dMatriz[i][j];
                                        tout << " ";
                                }
                                if ((i==3) && (j==4))
                                        tout << "];" << "\n";
                                else
                                {
                                        tout << ";";
                                        tout << "\n";
                                }
                        }
                }
                tout.close();

                ofstream cout("XYZeulerZYZ.txt");
                for (k=0;k<TotalPuntosAPT;k++)
                {
                        for (int j=0;j<6;j++)
                                cout << pAplicacion->m_pPuntoTrayectoria[k].LeerCoordXYZeulerZYZ(j)
<< " ";
                        cout << ";" << "\n";
                }
                cout.close();

// AQUI DEBE ALMACENAR LA TRAYECTORIA EN UNA BBDD y APARECER EL CUADRO DE
// DIÁLOGO PARA DECIDIR SI SIMULAR o ENVIAR DIRECTAMENTE.
// Si todos los puntos eran alcanzables, los mandamos por el puerto serie.
        if (fRes2)
```

```
        {
        // Para que el programa rx90.exe pueda ubicar la pieza virtual en el
        // sitio correcto calculamos el punto central de la pieza y también lo
        // guardamos en un fichero.

                CalculaPuntoCentral(nNumeroPuntosAPT);

                AfxMessageBox("La trayectoria es correcta. El robot se colocará en la posición
inicial e iniciará el corte",MB_ICONINFORMATION|MB_OK);

                CString strCoord, strPuntoTrayectoria, strMandarTrayectoria, strOrden,
strPuntoTheta;
                double dPuntoTray[6];

                for (nIndex = pasos; nIndex < (pasos+nNumeroPuntosAPT); nIndex++)
                {
                        pAplicacion->m_pPuntoTrayectoria[nIndex].GetXYZeulerZYZ(dPuntoTray);
                        for (nIndex2 = 0; nIndex2 < 5; nIndex2++)
                        {
                                strCoord.Format(_T("%f"),dPuntoTray[nIndex2]);
                                strPuntoTrayectoria += strCoord +_T(",");
                        }
                        strPuntoTrayectoria += _T("0");

                        double dTheta[6];

// Lo siguiente sirve para ubicar el extremo de la fresa en una
// posición próxima al primer punto de la trayectoria (unos 10cm
// separado en la dirección del eje -Z del extremo.
                        if (nIndex==pasos)
                        {
                                for (int i=0;i<5;i++)
                                {
                                        dTheta[i]=pAplicacion-
>m_pPuntoTrayectoria[0].m_sCoordTheta.theta[i];
                                        strCoord.Format(_T("%f"),dTheta[i]);
                                        strPuntoTheta +=strCoord +_T(",");
                                }
                                strPuntoTheta +=_T("0");

                                strOrden.Format(_T("DO SPEED 20 ALWAYS"));
                                EnviarOrden((LPCTSTR)strOrden);

//                              AfxMessageBox(strPuntoTheta);

                                // Asignamos pa la posición "ppio" las coordenadas
                                // del primer punto de la trayectoria.

                                strOrden.Format(_T("DO SET ppio = TRANS(%s)"),strPuntoTrayectoria);
                                EnviarOrden((LPCTSTR)strOrden);

                                // Localizamos la fresa a 100mm del punto "ppio".

                                strOrden.Format(_T("DO APPRO ppio,100"));
                                EnviarOrden((LPCTSTR)strOrden);
```

```
                                // La siguiente instrucción (cuando está inserta
                                // en un programa) evita que se ejecute la siguiente
                                // instrucción mientras el brazo del robot esté en
                                // movimiento; en nuestro caso, como no es un
                                // programa lo que se ejecuta sino comandos uno
                                // detrás de otro, no funciona. Se puede quitar y no
                                // pasa nada.

                                strOrden.Format(_T("DO BREAK"));
                                EnviarOrden((LPCTSTR)strOrden);
                        }

                        strMandarTrayectoria.Format(_T("DO SET zfree[1,%d] = TRANS(%s)"), nIndex-
pasos, strPuntoTrayectoria);
                        EnviarOrden((LPCTSTR)strMandarTrayectoria);

                        TRACE(strMandarTrayectoria + _T("\n"));
//                      m_riedMostrar.Mostrar(strMandarTrayectoria+_T("\n"), RGB(0,0,0));

                // Reseteamos el contenido de los objetos CString que emplean el operador +=.
                        strCoord.Empty();
                        strPuntoTrayectoria.Empty();
                }
        }

        // Vaciamos y eliminamos los puntos de la trayectoria.

        delete[] pAplicacion->m_pPuntoTrayectoria;
        return fRes2;
}
//////////////////////////////////////////////////////////////
//                                                          //
//      Función que calcula el punto central de la pieza    //
//      objetivo para poder ubicarla más adelante en la     //
//      escena.                                             //
//                                                          //
//////////////////////////////////////////////////////////////
void CCorte3DApp::CalculaPuntoCentral(int NPuntos)
{
        CCorte3DApp* pAplicacion = (CCorte3DApp*)AfxGetApp();

        double MaxDistancia=0;
        Vector posicion;
        double th[6];
        int d6=360;
        int a2=450;
        int d4=450;
        Vector PuntoCentral;
        Vector maximos;
        Vector minimos;
        Vector orientacion;
        int diferencia=100;

        orientacion.x=pAplicacion->m_MundoReal.m_dMatriz[0][2];
```

```
		orientacion.y=pAplicacion->m_MundoReal.m_dMatriz[1][2];
		orientacion.z=pAplicacion->m_MundoReal.m_dMatriz[2][2];

		for (int i=pasos;i<(NPuntos+pasos);i++)
		{
			for (int j=0;j<6;j++)
				th[j]=pAplicacion->m_pPuntoTrayectoria[i].m_sCoordTheta.theta[j]/180*PI;

// Ya tenemos en th[i] las coordenaddas articulares en RADIANES de el punto [i].
// Ahora aplicamos el modelo directo para obtener sus coordenadas cartesianas
// respecto al sistema World.
			// X=((c1*c23*c4-s1*s4)*s5+c1*s23*c5)*d6+c1*s23*d4+c1*c2*a2;
			posicion.x=((cos(th[0])*cos(th[1]+th[2])*cos(th[3])-
sin(th[0])*sin(th[3]))*sin(th[4])+cos(th[0])*sin(th[1]+th[2])*cos(th[4]))*d6+
cos(th[0])*sin(th[1]+th[2])*d4+cos(th[0])*cos(th[1])*a2;

			// Y=((s1*c23*c4+c1*s4)*s5+s1*s23*c5)*d6+s1*s23*d4+s1*c2*a2;
		posicion.y=((sin(th[0])*cos(th[1]+th[2])*cos(th[3])+cos(th[0])*
sin(th[3]))*sin(th[4])+sin(th[0])*sin(th[1]+th[2])*cos(th[4]))*d6+sin(th[0])*
sin(th[1]+th[2])*d4+sin(th[0])*cos(th[1])*a2;

			// Z=-(s23*c4*s5-c23*c5)*d6+c23*d4-s2*a2+d1;
			posicion.z=-(sin(th[1]+th[2])*cos(th[3])*sin(th[4])-
cos(th[1]+th[2])*cos(th[4]))*d6+cos(th[1]+th[2])*d4-sin(th[1])*a2;

			if (i==pasos)
			{
				maximos.x=minimos.x=posicion.x;
				maximos.y=minimos.y=posicion.y;
				maximos.z=minimos.z=posicion.z;
			}
			else
			{
				if (posicion.x>maximos.x) maximos.x=posicion.x;
				else if (posicion.x<minimos.x) minimos.x=posicion.x;

				if (posicion.y>maximos.y) maximos.y=posicion.y;
				else if (posicion.y<minimos.y) minimos.y=posicion.y;

				if (posicion.z>maximos.z) maximos.z=posicion.z;
				else if (posicion.z<minimos.z) minimos.z=posicion.z;
			}
		}

		PuntoCentral.x=(maximos.x+minimos.x)/2;
		PuntoCentral.y=(maximos.y+minimos.y)/2;
		PuntoCentral.z=(maximos.z+minimos.z)/2;

//		Modificamos el punto central en la dirección del vector orientación.

		PuntoCentral.x=PuntoCentral.x-diferencia*orientacion.x;
		PuntoCentral.y=PuntoCentral.y-diferencia*orientacion.y;
		PuntoCentral.z=PuntoCentral.z-diferencia*orientacion.z;

//		Para abrir un fichero para escritura, y añadir texto al final
```

```
//      hay que utilizar el modo "app".

        ofstream cout("puntocentral.txt",ios::app);

        cout << PuntoCentral.x << " " << PuntoCentral.y << " " << PuntoCentral.z << "\n";
        cout.close();
}

////////////////////////////////////////////////////////////
//                                                        //
//      Función que incorpora unos puntos al inicio de    //
//      la trayectoria (sólo para la del robot virtual)   //
//      para realizar una maniobra de aproximación.       //
//                                                        //
////////////////////////////////////////////////////////////
void CCorte3DApp::PuntosDeAproximacion()
{
        int separacion=100;
        CCorte3DApp* pAplicacion = (CCorte3DApp*)AfxGetApp();

        for (int i=0;i<pasos;i++)
        {
                for (int j=0;j<4;j++)
                        for (int k=0;k<4;k++)
                                pAplicacion->m_pPuntoTrayectoria[i].m_dMatriz[j][k]=pAplicacion-
>m_pPuntoTrayectoria[pasos].m_dMatriz[j][k];
                for (j=0;j<3;j++)
                        pAplicacion->m_pPuntoTrayectoria[i].m_dMatriz[j][3]-
=(separacion/pasos)*(pasos-i)*pAplicacion->m_pPuntoTrayectoria[pasos].m_dMatriz[j][2];
        }
}

////////////////////////////////////////////////////////////
//                                                        //
//      Función que incorpora unos puntos al final de     //
//      la trayectoria (sólo para la del robot virtual)   //
//      para realizar una maniobra de separación.         //
//                                                        //
////////////////////////////////////////////////////////////
void CCorte3DApp::PuntosDeSeparacion(int nNumeroPuntosAPT)
{
        int separacion=100;
        int PuntoInicial=nNumeroPuntosAPT+pasos;
        CCorte3DApp* pAplicacion = (CCorte3DApp*)AfxGetApp();
        double aux[4][4];

        for (int i=0;i<pasos;i++)
        {
                for (int j=0;j<4;j++)
                        for (int k=0;k<4;k++)
                        {
                                aux[j][k]=pAplicacion->m_pPuntoTrayectoria[PuntoInicial-
1].m_dMatriz[j][k];
                                pAplicacion-
>m_pPuntoTrayectoria[PuntoInicial+i].m_dMatriz[j][k]=aux[j][k];
```

```
                        }
                for (j=0;j<3;j++)
                        pAplicacion->m_pPuntoTrayectoria[PuntoInicial+i].m_dMatriz[j][3]-
=(separacion*i/pasos)*pAplicacion->m_pPuntoTrayectoria[PuntoInicial-1].m_dMatriz[j][2];
        }
}

////////////////////////////////////////////////////////////////////
//                                                                //
//      Función que pretende emular al programa V_TRAJSIG         //
//      del controlador CS7 cuando en alguna de las               //
//      articulaciones se produce un salto brusco,                //
//      modificando otros valores angulares mientras aquél        //
//      que cambia bruscamente lo hace lentamente evitando        //
//      que la fresa se desplace de forma indeseada.              //
//                                                                //
//      Sin esta función, el robot virtual modificará             //
//      de una vez esa variable angular moviendo con ella         //
//      todos los elementos del manipulador que dependan de       //
//      ella produciendo un efecto muy perjudicial.               //
//                                                                //
//      Esta función crea puntos intermedios que varíen           //
//      sus coordenadas articulares permaneciendo la fresa        //
//      fija en su posición actual.                               //
//                                                                //
////////////////////////////////////////////////////////////////////
void CCorte3DApp::ComprobarPosiblesSaltos(int PuntosAPT)
{
        CCorte3DApp* pAplicacion = (CCorte3DApp*)AfxGetApp();

        double Delta;
        int Pasos_intermedios, i, j, k;
        double th[6];
        double MatrizAux[4][4];
        double dAux1, dAux2, dAux;
// Esta función la utilizamos para buscar posibles saltos bruscos
// en la cuarta articulación, que es donde se suele producir.

// Guardaremos en el archivo de texto "posiciones.txt" los puntos
// que conforman la traycetoria.
        ofstream out("posiciones.txt");
        for (j = 0; j < 6; j++)
        {
                th[j]= pAplicacion->m_pPuntoTrayectoria[0].m_sCoordTheta.theta[j];
                if (j==3) dAux=th[j];
                out << th[j];
                out << " ";
        }
        out << ";";
        out << "\n";

        for (i = 1; i < PuntosAPT; i++)
        {
                dAux1=dAux;
                dAux2=pAplicacion->m_pPuntoTrayectoria[i].m_sCoordTheta.theta[3];
```

```
                dAux=dAux2;
                Delta=dAux1-dAux2;
// Si el salto entre ésta y la siguiente variable es grande
// incorporamos puntos intermedios.
                if (Delta>60 && Delta<300)
                {
                        Pasos_intermedios=Delta/10;
                        Delta=Delta/Pasos_intermedios;

        // Necesitamos para los cálculos la matriz de transformación
        // homogénea.
                        for (j=0;j<4;j++)
                                for (k=0;k<4;k++)
                                        MatrizAux[j][k]=pAplicacion-
>m_pPuntoTrayectoria[i+1].m_dMatriz[j][k];

                        for (j=1;j<(Pasos_intermedios);j++)
                        {
                                th[3]-=Delta;
                                for (k=0;k<6;k++)
                                        th[k]*=PI/180;
        // Las 3 primeras coordenadas articulares no cambian, la cuarta
        // la cambiamos nosotros.
        // La quinta la hemos de calcular a continuación.
        dAux1=MatrizAux[0][2]*(cos(th[0])*cos(th[1]+th[2])*cos(th[3])-
sin(th[0])*sin(th[3]))+MatrizAux[1][2]*(sin(th[0])*cos(th[1]+th[2])*cos(th[3])+cos(th[0])*sin(th[3]))
-MatrizAux[2][2]*cos(th[3])*sin(th[1]+th[2]);

        dAux2=MatrizAux[0][2]*cos(th[0])*sin(th[1]+th[2])+MatrizAux[1][2]*sin(th[0])*sin(th[1]+th[2]
)+MatrizAux[2][2]*cos(th[1]+th[2]);

                                th[4]=atan2(dAux1,dAux2);
                                for (k=0;k<6;k++)
                                {
                                        th[k]*=180/PI;
                                        out << th[k];
                                        out << " ";
                                }
                                out << ";";
                                out << "\n";
                        }
                }
                for (j = 0; j < 6; j++)
                {
                        th[j]= pAplicacion->m_pPuntoTrayectoria[i].m_sCoordTheta.theta[j];
                        out << th[j];
                        out << " ";
                }
                out << ";";
                out << "\n";
        }
        out.close();
}
```

B.1.2. CCorte3DDlg.

B.1.2.1. Corte3DDlg.h

```
// Corte3DDlg.h : header file
//

#if !defined(AFX_CORTE3DDLG_H__BC6636C1_D952_4D28_971C_6BED4AE60AF3__INCLUDED_)
#define AFX_CORTE3DDLG_H__BC6636C1_D952_4D28_971C_6BED4AE60AF3__INCLUDED_

#include "PuntoTrayectoria.h"     // Added by ClassView
#include "RichEditEx.h"  // Added by ClassView
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

/////////////////////////////////////////////////////////////////////////////
// CCorte3DDlg dialog

class CCorte3DDlg : public CDialog
{
// Construction
public:
//       int prueba;
        CCorte3DDlg(CWnd* pParent = NULL);        // standard constructor
        Vector m_vPuntos[3];
//       Joint m_vPuntoInicial[3];

// Dialog Data
        //{{AFX_DATA(CCorte3DDlg)
        enum { IDD = IDD_CORTE3D_DIALOG };
        CComboBox        m_cboEscribir;
        CRichEditEx m_riedMostrar;
        //}}AFX_DATA

        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CCorte3DDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);   // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:
        HICON m_hIcon;

//       void EnviarOrden(LPCTSTR lpszOrden);
//       BOOL AbrirYLeerFicheroAPT(CString& strContFicheroAPT);
//       BOOL ProcesarContenidoFicheroAPT(CString& strContFicheroAPT);

        // Generated message map functions
        //{{AFX_MSG(CCorte3DDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
        afx_msg void OnPaint();
```

```
        afx_msg HCURSOR OnQueryDragIcon();
        afx_msg void OnBtConectarcom();
        afx_msg void OnBtCerrarcom();
        afx_msg void OnBtLimpiar();
        afx_msg void OnBtApt();
        afx_msg void OnBtEnviar();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_CORTE3DDLG_H__BC6636C1_D952_4D28_971C_6BED4AE60AF3__INCLUDED_)
```

B.1.2.2. Corte3DDlg.cpp

```
// Corte3DDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Corte3D.h"
#include "Corte3DDlg.h"
#include "SelectConfigCOMDlg.h"
#include "MundoVirtualDlg.h"
#include "DlgSimulacion.h"
#include <process.h>
#include "stdlib.h"
#include <fstream.h>
#include <iostream.h>
#include "PuntoTrayectoria.h"
#include "TCHAR.H"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// Deshabilitamos el warning C4129: ' ' : unrecognized character escape sequence,
// que se produce al usar GOTO\ en una cadena que mandamos por el puerto serie.

#pragma warning(disable:4129)

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
        CAboutDlg();

// Dialog Data
```

```
        //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
        //}}AFX_DATA

        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CAboutDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:
        //{{AFX_MSG(CAboutDlg)
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
        //{{AFX_DATA_INIT(CAboutDlg)
        //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAboutDlg)
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
        //{{AFX_MSG_MAP(CAboutDlg)
                // No message handlers
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CCorte3DDlg dialog

CCorte3DDlg::CCorte3DDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CCorte3DDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CCorte3DDlg)
        // NOTE: the ClassWizard will add member initialization here
        m_vPuntos[0].x = -99999.0;
        m_vPuntos[0].y = -99999.0;
        m_vPuntos[0].z = -99999.0;
        m_vPuntos[1].x = -99999.0;
        m_vPuntos[1].y = -99999.0;
        m_vPuntos[1].z = -99999.0;
        m_vPuntos[2].x = -99999.0;
        m_vPuntos[2].y = -99999.0;
        m_vPuntos[2].z = -99999.0;

        //}}AFX_DATA_INIT
```

```
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CCorte3DDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CCorte3DDlg)
	DDX_Control(pDX, IDC_CO_ESCRIBIR, m_cboEscribir);
	DDX_Control(pDX, IDC_RIED_MOSTRAR, m_riedMostrar);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CCorte3DDlg, CDialog)
	//{{AFX_MSG_MAP(CCorte3DDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_BT_CONECTARCOM, OnBtConectarcom)
	ON_BN_CLICKED(IDC_BT_CERRARCOM, OnBtCerrarcom)
	ON_BN_CLICKED(IDC_BT_LIMPIAR, OnBtLimpiar)
	ON_BN_CLICKED(IDC_BT_APT, OnBtApt)
	ON_BN_CLICKED(IDC_BT_ENVIAR, OnBtEnviar)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CCorte3DDlg message handlers

BOOL CCorte3DDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
		}
	}

	// 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

	return TRUE;  // return TRUE  unless you set the focus to a control
}

void CCorte3DDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

//  If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CCorte3DDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
	}
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CCorte3DDlg::OnQueryDragIcon()
{
	return (HCURSOR) m_hIcon;
}
```

```
////////////////////////////////////////////////////////////////////////////
//
// Método manejador del mensaje lanzado cuando se pulsa el botón Conectar.
// Abre el puerto COMM con la configuración seleccionada y habilita o
// deshabilita botones.
//
////////////////////////////////////////////////////////////////////////////
void CCorte3DDlg::OnBtConectarcom()
{
	// TODO: Add your control notification handler code here

	// Abrimos un cuadro de diálogo donde seleccionamos y configuramos un puerto serie.
	int nRes;

	CSelectConfigCOMDlg dlgSelectConfigCOM(this);
	nRes = dlgSelectConfigCOM.DoModal();

	if(nRes == IDOK)
	{
		// Habilitamos el editor y los botones de enviar, enviar APT, limpiar y desconectar
y deshabilito el de conectar.
		GetDlgItem(IDC_CO_ESCRIBIR)->EnableWindow(TRUE);
		GetDlgItem(IDC_BT_CERRARCOM)->EnableWindow(TRUE);
		GetDlgItem(IDC_BT_ENVIAR)->EnableWindow(TRUE);
		GetDlgItem(IDC_BT_APT)->EnableWindow(TRUE);
		GetDlgItem(IDC_BT_CONECTARCOM)->EnableWindow(FALSE);

		//	Pasamos el foco al editor.
		m_cboEscribir.SetFocus();
	}
	else
	{
		AfxMessageBox(_T("No se ha abierto ningún puerto COM"),MB_ICONSTOP|MB_OK);
	}
}

////////////////////////////////////////////////////////////////////////////
//
// Método manejador del mensaje lanzado cuando se pulsa el botón Desconectar.
// Cierra el puerto COMM abierto y habilita o deshabilita botones.
//
////////////////////////////////////////////////////////////////////////////
void CCorte3DDlg::OnBtCerrarcom()
{
	// TODO: Add your control notification handler code here

	// Si el puerto serie estaba abierto, lo cerramos. La comprobación se hace internamente.
	CCorte3DApp* pAplicacion = (CCorte3DApp*)AfxGetApp();
	pAplicacion->m_serie.Cerrar();

	// Deshabilitamos el editor y los botones de enviar, enviar APT, limpiar y desconectar y
habilito el de conectar.
	GetDlgItem(IDC_CO_ESCRIBIR)->EnableWindow(FALSE);
	GetDlgItem(IDC_BT_CERRARCOM)->EnableWindow(FALSE);
```

```
        GetDlgItem(IDC_BT_ENVIAR)->EnableWindow(FALSE);
        GetDlgItem(IDC_BT_APT)->EnableWindow(FALSE);
        GetDlgItem(IDC_BT_LIMPIAR)->EnableWindow(FALSE);
        GetDlgItem(IDC_BT_CONECTARCOM)->EnableWindow(TRUE);
}

////////////////////////////////////////////////////////////////////////////
//
// Método manejador del mensaje lanzado cuando se pulsa el botón Limpiar.
// Borra todo el contenido del control RichEdirEx.
//
////////////////////////////////////////////////////////////////////////////
void CCorte3DDlg::OnBtLimpiar()
{
        // TODO: Add your control notification handler code here

        // Borra todo el texto del control RichEdit
        m_riedMostrar.Borrar();

        // Desactivamos el botón de Limpiar, porque la pantalla estará limpia.
        GetDlgItem(IDC_BT_LIMPIAR)->EnableWindow(FALSE);
}


////////////////////////////////////////////////////////////////////////////
//
// Método manejador del mensaje lanzado cuando se pulsa el botón Enviar.
// Si hemos escrito algo en el ComboBox, actualiza el histórico de comandos,
// lo muestra por pantalla en negro y lo envia al controlador.
//
////////////////////////////////////////////////////////////////////////////
void CCorte3DDlg::OnBtEnviar()
{
        // TODO: Add your control notification handler code here

        // Actualizamos el contenido de la pantalla.
        UpdateData(TRUE);

        // Capturamos el contenido escrito.
        CString strComandoEscrito;
        m_cboEscribir.GetWindowText(strComandoEscrito);

        // Si hemos escrito algo.
        if (strComandoEscrito.IsEmpty() == FALSE)                                       {
                // Enviamos la orden por el puerto serie. La función se encarga de transformar
                // a char* internamente.
                CCorte3DApp* pAplicacion = (CCorte3DApp*)AfxGetApp();
                pAplicacion->EnviarOrden((LPCTSTR)strComandoEscrito);

                // Si el contenido ya existe en la lista, lo borramos.
                int nIndex;
                if ((nIndex = m_cboEscribir.FindStringExact(-1,strComandoEscrito)) != LB_ERR)

                {
                        m_cboEscribir.DeleteString(nIndex);
```

```
                }

                // Añadimos el contenido al principio de la lista.
                m_cboEscribir.InsertString(0,strComandoEscrito);

                // Escribimos en la pantalla con color negro.
                CString strComandoAMostrarPorPantalla;
                strComandoAMostrarPorPantalla.Format(_T("%s\n"), strComandoEscrito);
                m_riedMostrar.Mostrar(strComandoAMostrarPorPantalla, RGB(0,0,0));

                // Borramos el contenido del editor.
                strComandoEscrito.Empty();
                m_cboEscribir.SetWindowText(strComandoEscrito);

                // Actualizamos los datos presentados en la pantalla.
                UpdateData(FALSE);

                // Activamos el botón de Limpiar, porque la pantalla ya no estará limpia.
                GetDlgItem(IDC_BT_LIMPIAR)->EnableWindow(TRUE);
        }
}

////////////////////////////////////////////////////////////////////////////
//
// Método manejador del mensaje lanzado cuando se pulsa el botón Generar Trayectoria.
// Realiza las operaciones necesarias para que el robot ejecute la trayectoria
// contenida en el fichero APT que abramos.
//
////////////////////////////////////////////////////////////////////////////
void CCorte3DDlg::OnBtApt()
{
        // TODO: Add your control notification handler code here

        //Vector vPuntos[3];
        //Vector vAux;
        CCorte3DApp* pAplicacion = (CCorte3DApp*)AfxGetApp();

// 1. Calculamos la matriz de transformación wHp (transformación de {W} a {P},
// es decir, nos da las coordenadas de cualquier punto respecto al sistema
// de referencia ligado a la pieza (pP = (wHp)-1 * wP).
// Mostramos consecutivamente 3 cuadros de diálogo que indiquen al operario
// que debe colocar el robot en las posiciones iniciales deseadas y vamos
// almacenándolas.
// Cuando pulse Aceptar en los 3 cuadros continuará el programa. Si no, se
// cancela.
        int nRes = AfxMessageBox(_T("Coloque el extremo de la fresa en la posición inicial 1
(Esquina superior izquierda) y pulse ACEPTAR"),MB_ICONINFORMATION|MB_OKCANCEL);
        if (nRes == IDOK)
        {
// Almacenamos la primera posición inicial (p0 = Esquina superior izquierda)
                pAplicacion->m_serie.CapturarDatosLeidos(&m_vPuntos[0]);
//      pAplicacion->m_serie.CapturarPuntoInicial(&m_vPuntoInicial[0]);

//              CString msg;
//              msg.Format("Esq.Sup.Izda:[%f,%f,%f]",m_vPuntos[0].x,m_vPuntos[0].y,m_vPuntos[0].z);
```

```
//              AfxMessageBox(msg);

// En CapturarDatosLeidos(&m_vPuntos[0]) se envía la orden where, que devuelve
// la posición del extremo en coordenadas cartesianas y articulares, pero sólo // toma las tres
primeras (x,y,z). Para que en primer lugar dé las 6
// coordenadas articulares hay    que escribir "here #A" donde # requiere la
// posición del punto actual.
// Habría que definir una estructura con 6 valores double para calcular las
// posiciones iniciales del tablero.
// Luego habría que con las matrices de cada punto de la trayectoria, habría
// que cambiar la posición (x,y,z) restándole la dimensión de la fresa en la
// dirección del vector orientación del extremo (3ª columna de la matriz de
// transformación homogénea).


//              m_vPuntos[0].x=-213.95;
//              m_vPuntos[0].y=762.86;
//              m_vPuntos[0].z=362.34;

//              m_vPuntos[0].x=-201.86;
//              m_vPuntos[0].y=626.77;
//              m_vPuntos[0].z=342.24;

//              m_vPuntos[0].x = -133.013;
//              m_vPuntos[0].y = 657.891;
//              m_vPuntos[0].z = 417.842;

                Sleep(500);

                nRes = AfxMessageBox(_T("Coloque el extremo de la fresa en la posición inicial 2
(Esquina superior derecha) y pulse ACEPTAR"),MB_ICONINFORMATION|MB_OKCANCEL);
                if (nRes == IDOK)
                {
                        // Almacenamos la segunda posición inicial(p1 = Esquina superior derecha).
                        pAplicacion->m_serie.CapturarDatosLeidos(&m_vPuntos[1]);

//                      m_vPuntos[1].x=261.29;
//                      m_vPuntos[1].y=747.95;
//                      m_vPuntos[1].z=359.16;

//                      m_vPuntos[1].x=166.66;
//                      m_vPuntos[1].y=635.34;
//                      m_vPuntos[1].z=345.83;

//                      m_vPuntos[1].x = 133.023;
//                      m_vPuntos[1].y = 657.884;
//                      m_vPuntos[1].z = 417.836;

                        Sleep(500);

                        nRes = AfxMessageBox(_T("Coloque el extremo de la fresa en la posición
inicial 3 (Esquina inferior derecha) y pulse ACEPTAR"),MB_ICONINFORMATION|MB_OKCANCEL);
                        if (nRes == IDOK)
                        {
```

```
                                // Almacenamos la tercera posición inicial (p2 = Esquina inferior
derecha).
                                pAplicacion->m_serie.CapturarDatosLeidos(&m_vPuntos[2]);

//                              m_vPuntos[2].x=239.15;
//                              m_vPuntos[2].y=689.48;
//                              m_vPuntos[2].z=77.79;

//                              m_vPuntos[2].x=166.66;
//                              m_vPuntos[2].y=515.44;
//                              m_vPuntos[2].z=21.15;

//                              m_vPuntos[2].x = 133.015;
//                              m_vPuntos[2].y = 467.573;
//                              m_vPuntos[2].z = 240.372;

                                Sleep(500);

                                AfxMessageBox(_T("Ahora calcularemos el sistema de referencia en el
entorno real"),MB_ICONINFORMATION|MB_OK);

                                // Una vez que tenemos los 3 puntos, calculamos wHp.
                                pAplicacion->m_MundoReal.CrearMatrizDesplazada(m_vPuntos);
                        }
                }
        }

        // Si alguno de los puntos no se ha almacenado, se indica.
        if (nRes != IDOK)
                AfxMessageBox(_T("Error al calcular el punto inicial en el entorno real. Repita el
proceso"),MB_ICONEXCLAMATION|MB_OK);

        else
        {
                // 2. Calculamos la matriz de transformación wvHpv
                // (transformación de {Wv} a {Pv},
                // es decir, nos da las coordenadas de cualquier punto respecto
                // al sistema de referencia ligado a la pieza (pvP = (wvHpv)-1 *
                // wvP) en el entorno virtual.
                CMundoVirtualDlg dlgEntornoVirtual(this);
                nRes = dlgEntornoVirtual.DoModal();

                if(nRes == IDOK)
                {
                        // Ya tenemos almacenadas en vPuntos[3] los puntos p0,p1 y
                        // p2 en el entorno virtual.
                        // Creamos la matriz de transformación homogénea.
                        pAplicacion->m_MundoVirtual.CrearMatriz(m_vPuntos);

                        // 3. Calculamos la matriz de transformación wHwv
                        // (transformación de {Wv} a {V}, es decir, nos da las
                        // coordenadas de cualquier punto respecto al sistema
                        // de referencia WORLD del entorno real:
                        // wHwv = wHp * (wvHpv)-1tal que wP = wHwv * wvP
```

```
                        pAplicacion->m_MundoVirtual.InvertirMatriz();
                        pAplicacion->m_MundoVirtual.Componer(pAplicacion->m_MundoReal);

        //
        // LA TRANSFORMACIÓN QUE DEBEMOS USAR AHORA ES PREMULTIPLICAR CADA PTO
        // DE LA TRAYECTORIA POR ptoMundoVirtual, que deberia llamarse
        // ptoTransformacion, o algo así.
        //

                        // 4.Abrimos, leemos y procesamos el contenido del fichero
                        // APT.
                        // Primero abrimos el fichero .APTsource.
                        //BOOL fRes;
                        CString strContenidoFicheroAPT;

                        BOOL fRes = pAplicacion->AbrirYLeerFicheroAPT(strContenidoFicheroAPT);


                        // Si no han ocurrido errores, procesamos el fichero.
                        if(fRes)
                                fRes = pAplicacion-
>ProcesarContenidoFicheroAPT(strContenidoFicheroAPT);

                        // Si no han ocurrido errores y todos los puntos son alcanzables, deben
haberse
                        // almacenado en el controlador todos los puntos que forman
                        // la trayectoria.
                        // En este caso, continuamos mandando los comandos
                        // necesarios y finalmente ejecutamos el programa
                        // V_TRAJSIG.
                        if(fRes)
                        {

                        // Llamamos al diálogo que nos pregunta si deseamos hacer
                        // la simulacion.

                                CDlgSimulacion dlgSimular(this);
                                nRes = dlgSimular.DoModal();

                        // Internamente trabajamos siempre con CString (pues
                        // contiene caracteres Unicode o MBCS según se haya
                        // especificado.
                        // La función encargada de transmitir por el puerto serie
                        // realiza la conversión a char*, pues se debe hacer
                        // carácter a carácter.
                                CString strOrden;

        // Borramos todos los puntos zfree[1,x] de trayectorias anteriores.
        // Para hacerlo corectamente habría que mirar cuantos zfree[1,x] hay. SE
        // PUEDE HACER CON ORDEN last.
        // Tener en cuenta que la cola de peticiones de escritura tiene un
        // tamaño limitado.
        //                      for (int nIndex=0; nIndex<50; nIndex++)
        //                      {
        //                              strOrden.Format(_T("DELETEL zfree[1,%d]"), nIndex);
```

```
        //                              EnviarOrden((LPCTSTR)strOrden);
        //                              strOrden.Format(_T("Y"));
        //                              EnviarOrden((LPCTSTR)strOrden);
        //                      }

                                // Establecemos el valor de outil[1].
                                // Aq en la memoria ponga  (0 0 275 0 0 0), hay que
                                // ponerlo todo a 0 xq si no no funciona!!!!!
                                // Internamente en algún lado se debe tener en cuenta!!!!


//      PUEDO HACER UNA APROXIMACION CON "do appro punto_incicial,100"
//      BASTARÍA CON METER EN punto_inicial LAS COORDENADAS DEL PRIMER PUNTO DE
//      LA TRAYECTORIA

                                strOrden.Format(_T("DO SET outil[1] = TRANS(0,0,0,0,0,0)"));
                                pAplicacion->EnviarOrden((LPCTSTR)strOrden);

                        // Establecemos el valor de #start[1], cuyas componentes
                        // deben coincidir con el valor de las coordenadas
                        // articulares en zfree[1,1].

//                              int nIndex;
//                              double dTheta[6];
//                              pAplicacion->m_pPuntoTrayectoria[10].GetXYZeulerZYZ(dTheta);
//                              CString msg;
//                              msg.Format("DO SET ppio =
TRANS(%f,%f,%f,%f,%f,%f",dTheta[0],dTheta[1],dTheta[2],dTheta[3],dTheta[4],dTheta[5]);
//                              AfxMessageBox(msg);
//                              strOrden.Format(_T("DO SET ppio =
TRANS(%f,%f,%f,%f,%f,%f)"),dTheta[0],dTheta[1],dTheta[2],dTheta[3],dTheta[4],dTheta[5]);
//                              for (nIndex = 0; nIndex<5; nIndex++)
//                                      strOrden += _T(dTheta[nIndex],",");
//                              strOrden += _T(dTheta[nIndex],")");

//                              pAplicacion->EnviarOrden((LPCTSTR)strOrden);


                                strOrden.Format(_T("DO APPRO ppio,100"));
                                pAplicacion->EnviarOrden((LPCTSTR)strOrden);


                                strOrden.Format(_T("DO HERE #start[1]"));
                                pAplicacion->EnviarOrden((LPCTSTR)strOrden);

                                // Establecemos las velocidades en el robot.
                                strOrden.Format(_T("DO SPEED 20 MONITOR"));
                                pAplicacion->EnviarOrden((LPCTSTR)strOrden);

                                strOrden.Format(_T("DO SPEED 20 ALWAYS"));
                                pAplicacion->EnviarOrden((LPCTSTR)strOrden);

                                // Ejecutamos V_TRAJSIG.
                                strOrden.Format(_T("EXE traject.main(1,ON)"));
                                pAplicacion->EnviarOrden((LPCTSTR)strOrden);
```

```
                        }
                }
        }
}
```

B.1.3. CSelectConfigCOMDlg.

B.1.3.1. SelectConfigCOMDlg.h

```
// SelectConfigCOMDlg.h : header file
//

#if !defined(AFX_SELECTCONFIGCOMDLG_H__DF93BB40_B22D_47EE_86BB_3B45E0729C72__INCLUDED_)
#define AFX_SELECTCONFIGCOMDLG_H__DF93BB40_B22D_47EE_86BB_3B45E0729C72__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "Corte3DDlg.h"

/////////////////////////////////////////////////////////////////////////////
// CSelectConfigCOMDlg dialog

class CSelectConfigCOMDlg : public CDialog
{
// Construction
public:
//      CSelectConfigCOMDlg(CWnd* pParent = NULL);   // standard constructor
        CSelectConfigCOMDlg(CCorte3DDlg* pParent = NULL);

        // Dialog Data
        //{{AFX_DATA(CSelectConfigCOMDlg)
        enum { IDD = IDD_COM_SELECTCONFIG };
        //}}AFX_DATA

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CSelectConfigCOMDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:
        CCorte3DDlg* m_pDlgPrincipal;     //Puntero al diálogo llamador (el ppal) necesario para
poder acceder a la variable miembro m _serie.
        int             m_nPuerto;
        CString  m_strTasaBaudios;
        CString  m_strBitsDatos;
        CString  m_strParidad;
        CString  m_strBitsParada;
        CString  m_strControlFlujo;
```

```
        // Generated message map functions
        //{{AFX_MSG(CSelectConfigCOMDlg)
        virtual void OnOK();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_SELECTCONFIGCOMDLG_H__DF93BB40_B22D_47EE_86BB_3B45E0729C72__INCLUDED_)
```

B.1.3.2. SelectConfigCOMDlg.cpp

```
// SelectConfigCOMDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Corte3D.h"
#include "SelectConfigCOMDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CSelectConfigCOMDlg dialog

//CSelectConfigCOMDlg::CSelectConfigCOMDlg(CWnd* pParent /*=NULL*/)
//       : CDialog(CSelectConfigCOMDlg::IDD, pParent)
CSelectConfigCOMDlg::CSelectConfigCOMDlg(CCorte3DDlg* pParent /*=NULL*/)
        : CDialog(CSelectConfigCOMDlg::IDD, pParent),m_pDlgPrincipal(pParent)
{
        //{{AFX_DATA_INIT(CSelectConfigCOMDlg)
        m_nPuerto = 0;
        m_strTasaBaudios = _T("9600");
        m_strBitsDatos = _T("8");
        m_strParidad = _T("Deshabilitada");
        m_strBitsParada = _T("1");
        m_strControlFlujo = _T("Deshabilitado");
        //}}AFX_DATA_INIT
}

void CSelectConfigCOMDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CSelectConfigCOMDlg)
        DDX_Radio(pDX, IDC_RA_COM1, m_nPuerto);
        DDX_CBString(pDX, IDC_CO_BAUDIOS, m_strTasaBaudios);
        DDX_CBString(pDX, IDC_CO_DATABITS, m_strBitsDatos);
        DDX_CBString(pDX, IDC_CO_PARIDAD, m_strParidad);
        DDX_CBString(pDX, IDC_CO_STOPBITS, m_strBitsParada);
```

```
        DDX_CBString(pDX, IDC_CO_CTRLFLUJO, m_strControlFlujo);
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CSelectConfigCOMDlg, CDialog)
        //{{AFX_MSG_MAP(CSelectConfigCOMDlg)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
////////////////////////////////////////////////////////////////////////////
// CSelectConfigCOMDlg message handlers

////////////////////////////////////////////////////////////////////////////
//
// Método manejador del mensaje lanzado cuando se pulsa el botón OK.
// Lee la configuración seleccionada en pantalla y abre el puerto serie.
//
////////////////////////////////////////////////////////////////////////////
void CSelectConfigCOMDlg::OnOK()
{
        // TODO: Add extra validation here

        // Actualizamos el valor de las variables para asegurarnos de que son
        // los seleccionados.
        UpdateData(TRUE);

        // Abre el puerto serie con los parámetros seleccionados.
        CString strPuerto;
        strPuerto.Format(_T("COM%d"),m_nPuerto + 1);

        // Fijamos la tasa de baudios.
        CPuertoSerie::ETasaBaudios eTasaBaudios;
        if(m_strTasaBaudios == _T("9600"))
                eTasaBaudios = CPuertoSerie::EBaud9600;
        else if(m_strTasaBaudios == _T("110"))
                eTasaBaudios = CPuertoSerie::EBaud110;
        else if(m_strTasaBaudios == _T("300"))
                eTasaBaudios = CPuertoSerie::EBaud300;
        else if(m_strTasaBaudios == _T("600"))
                eTasaBaudios = CPuertoSerie::EBaud600;
        else if(m_strTasaBaudios == _T("1200"))
                eTasaBaudios = CPuertoSerie::EBaud1200;
        else if(m_strTasaBaudios == _T("2400"))
                eTasaBaudios = CPuertoSerie::EBaud2400;
        else if(m_strTasaBaudios == _T("4800"))
                eTasaBaudios = CPuertoSerie::EBaud4800;
        else if(m_strTasaBaudios == _T("19200"))
                eTasaBaudios = CPuertoSerie::EBaud19200;
        else if(m_strTasaBaudios == _T("38400"))
                eTasaBaudios = CPuertoSerie::EBaud38400;
        else
                ASSERT(FALSE);
                // podemos añadir aki un lanzamiento de excepcion + TRACE

        // Fijamos el tipo de paridad.
        CPuertoSerie::EParidad eParidad;
```

```
        if (m_strParidad = _T("Deshabilitada"))
                eParidad = CPuertoSerie::ENoParidad;
        else if (m_strParidad = _T("Paridad par"))
                eParidad = CPuertoSerie::EParidadPar;
        else if (m_strParidad = _T("Paridad impar"))
                eParidad = CPuertoSerie::EParidadImpar;
        else
                ASSERT(FALSE);

        // Fijamos el número de bits de datos.
        CPuertoSerie::EBitsDatos eBitsDatos;
        if (m_strBitsDatos == _T("8"))
                eBitsDatos = CPuertoSerie::EDatos8;
        else if (m_strBitsDatos == _T("7"))
                eBitsDatos = CPuertoSerie::EDatos7;
        else
                ASSERT(FALSE);

        // Fijamos el número de bits de parada.
        CPuertoSerie::EBitsParada eBitsParada;
        if (m_strBitsParada == _T("1"))
                eBitsParada = CPuertoSerie::EStop1;
        else if (m_strBitsDatos == _T("2"))
                eBitsParada = CPuertoSerie::EStop2;
        else
                ASSERT(FALSE);

        // Fijamos el tipo de control de flujo.
        CPuertoSerie::EControlFlujo eControlFlujo;
        if (m_strControlFlujo = _T("Deshabilitado"))
                eControlFlujo = CPuertoSerie::ENoControlFlujo;
        else if (m_strControlFlujo = _T("Por hardware (RTS/CTS)"))
                eControlFlujo = CPuertoSerie::ECtsRts;
        else if (m_strControlFlujo = _T("Por software (CTRL+S, CTRL+Q)"))
                eControlFlujo = CPuertoSerie::EXonXoff;
        else
                ASSERT(FALSE);

        // Abrimos el puerto serie con la configuración especificada. Le pasamos
        //como parámetro [un puntero al] el objeto m_riedMostrar para que pueda
        //escribir en la zona adecuada.
        CCorte3DApp* pAplicacion = (CCorte3DApp*)AfxGetApp();
        pAplicacion->m_serie.Abrir(strPuerto, eTasaBaudios, eParidad, eBitsDatos, eBitsParada,
eControlFlujo, &m_pDlgPrincipal->m_riedMostrar);
        //m_pDlgPrincipal->m_serie.Abrir(strPuerto, eTasaBaudios, eParidad, eBitsDatos, eBitsParada,
eControlFlujo, /*, TRUE*/);

        CDialog::OnOK();
}
```

B.1.4. CMundoVirtualDlg.

B.1.4.1. MundoVirtualDlg.h

```
//////////////////////////////////////////////////////
```

```
//                                                      //
//      MundoVirtualDlg.h: Definición de la clase       //
//      CMundoVirtualDlg.                               //
//                                                      //
//////////////////////////////////////////////////////////

#if !defined(AFX_MUNDOVIRTUALDLG_H__A6772C9D_F333_4D9D_BCD1_F4F1BD09BCC2__INCLUDED_)
#define AFX_MUNDOVIRTUALDLG_H__A6772C9D_F333_4D9D_BCD1_F4F1BD09BCC2__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// MundoVirtualDlg.h : header file
//

#include "Corte3DDlg.h"

/////////////////////////////////////////////////////////////////////////////
// CMundoVirtualDlg dialog

class CMundoVirtualDlg : public CDialog
{
// Construction
public:
        CMundoVirtualDlg(CCorte3DDlg* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(CMundoVirtualDlg)
protected:
        enum { IDD = IDD_ENTORNOVIRTUAL };
        double  m_edP0X;
        double  m_edP0Y;
        double  m_edP0Z;
        double  m_edP1X;
        double  m_edP1Y;
        double  m_edP1Z;
        double  m_edP2X;
        double  m_edP2Y;
        double  m_edP2Z;
        //}}AFX_DATA

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMundoVirtualDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:
        CCorte3DDlg* m_pDlgPrincipal;     //Puntero al diálogo llamador (el ppal) necesario para
poder acceder a la variable miembro m _serie.

        // Generated message map functions
        //{{AFX_MSG(CMundoVirtualDlg)
```

```
        virtual void OnOK();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.
#endif // !defined(AFX_MUNDOVIRTUALDLG_H__A6772C9D_F333_4D9D_BCD1_F4F1BD09BCC2__INCLUDED_)
```

B.1.4.2. MundoVirtualDlg.cpp

```
// MundoVirtualDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Corte3D.h"
#include "MundoVirtualDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMundoVirtualDlg dialog

CMundoVirtualDlg::CMundoVirtualDlg(CCorte3DDlg* pParent /*=NULL*/)
        : CDialog(CMundoVirtualDlg::IDD, pParent),m_pDlgPrincipal(pParent)
{
        //{{AFX_DATA_INIT(CMundoVirtualDlg)
        m_edP0X = 165.831;
        m_edP0Y = 165.831;
        m_edP0Z = 102.506;
        m_edP1X = -165.831;
        m_edP1Y = 165.831;
        m_edP1Z = 102.506;
        m_edP2X = -165.831;
        m_edP2Y = -165.831;
        m_edP2Z = 102.506;
        //}}AFX_DATA_INIT
}

void CMundoVirtualDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CMundoVirtualDlg)
        DDX_Text(pDX, IDC_ED_P0X, m_edP0X);
        DDX_Text(pDX, IDC_ED_P0Y, m_edP0Y);
        DDX_Text(pDX, IDC_ED_P0Z, m_edP0Z);
        DDX_Text(pDX, IDC_ED_P1X, m_edP1X);
        DDX_Text(pDX, IDC_ED_P1Y, m_edP1Y);
        DDX_Text(pDX, IDC_ED_P1Z, m_edP1Z);
        DDX_Text(pDX, IDC_ED_P2X, m_edP2X);
        DDX_Text(pDX, IDC_ED_P2Y, m_edP2Y);
```

```
        DDX_Text(pDX, IDC_ED_P2Z, m_edP2Z);
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CMundoVirtualDlg, CDialog)
        //{{AFX_MSG_MAP(CMundoVirtualDlg)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMundoVirtualDlg message handlers

void CMundoVirtualDlg::OnOK()
{
        // TODO: Add extra validation here

        // Actualizamos el valor de las variables para asegurarnos de que son
        // los seleccionados.
        UpdateData(TRUE);

        // Almacenamos los datos en las variables Vector[3].
        m_pDlgPrincipal->m_vPuntos[0].x = m_edP0X;
        m_pDlgPrincipal->m_vPuntos[0].y = m_edP0Y;
        m_pDlgPrincipal->m_vPuntos[0].z = m_edP0Z;
        m_pDlgPrincipal->m_vPuntos[1].x = m_edP1X;
        m_pDlgPrincipal->m_vPuntos[1].y = m_edP1Y;
        m_pDlgPrincipal->m_vPuntos[1].z = m_edP1Z;
        m_pDlgPrincipal->m_vPuntos[2].x = m_edP2X;
        m_pDlgPrincipal->m_vPuntos[2].y = m_edP2Y;
        m_pDlgPrincipal->m_vPuntos[2].z = m_edP2Z;

        CDialog::OnOK();
}
```

B.1.5. CRichEditEx.

B.1.5.1. RichEditEx.h

```
/////////////////////////////////////////////////////////////////////////////
//                                                                         //
//      Módulo : RichEditEx.h                                              //
//                      Definición de la clase CRichEditEx.                //
//                                                                         //
//      Propósito:      Clase derivada de CRichEditCtrl que tiene la       //
//      particularidad de que hace autoscroll vertical incluso después de  //
//      que el contenido se actualice, bien con UpdateData o bien de otra  //
//      forma. Además, contiene un método para mostrar objetos CString     //
//      de un color determinado en el control y otro para borrar su        //
//      contenido.                                                         //
//                                                                         //
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_RICHEDITEX_H__62A1C006_D011_455B_AE49_D043B22FB575__INCLUDED_)
#define AFX_RICHEDITEX_H__62A1C006_D011_455B_AE49_D043B22FB575__INCLUDED_
```

```
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

/////////////////////////////////////////////////////////////////////////////
// CRichEditEx window

class CRichEditEx : public CRichEditCtrl
{
// Construction
public:
        CRichEditEx();

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CRichEditEx)
        //}}AFX_VIRTUAL

// Implementation
public:
        void Borrar();
        void Mostrar(LPCTSTR lpszMsg, COLORREF Color);
        virtual ~CRichEditEx();

        // Generated message map functions
protected:
        //{{AFX_MSG(CRichEditEx)
                // NOTE - the ClassWizard will add and remove member functions here.
        //}}AFX_MSG

        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_RICHEDITEX_H__62A1C006_D011_455B_AE49_D043B22FB575__INCLUDED_)
```

B.1.5.2.RichEditEx.cpp

```
////////////////////////////////////////////////////////////////////
//                                                                //
//      Módulo : RichEditEx.cpp                                   //
//                      Implementación de la clase CRichEditEx.   //
//                                                                //
////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////
```

```
// Incluimos el fichero de cabecera estándar
////////////////////////////////////////////////////////////////////////////

#include "stdafx.h"


////////////////////////////////////////////////////////////////////////////
// Incluimos los ficheros de cabecera del módulo y del programa proyecto
////////////////////////////////////////////////////////////////////////////

#include "RichEditEx.h"
#include "Corte3D.h"

////////////////////////////////////////////////////////////////////////////
// Habilitamos el 'debug memory manager'
////////////////////////////////////////////////////////////////////////////

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

////////////////////////////////////////////////////////////////////////////
// Código
////////////////////////////////////////////////////////////////////////////

CRichEditEx::CRichEditEx()
{
}

CRichEditEx::~CRichEditEx()
{
}

BEGIN_MESSAGE_MAP(CRichEditEx, CRichEditCtrl)
        //{{AFX_MSG_MAP(CRichEditEx)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()


////////////////////////////////////////////////////////////////////////////
// CRichEditEx message handlers

//////////////////////////////////////////////////////////////////////////
//                                                                      //
// Método que sirve para mostrar en el control RichEdit la cadena      //
// especificada. Además realiza un scroll vertical automático para que  //
// siempre sea visible la parte inferior del contor RichEdit.           //
//                                                                      //
// Parámetros:                                                          //
// - Puntero a CString cte que almacena la cadena a mostrar por pantalla. //
// - Color en el que se mostrará la cadena de caracteres.               //
//                                                                      //
```

```
////////////////////////////////////////////////////////////////////////
void CRichEditEx::Mostrar(LPCTSTR lpszMsg, COLORREF Color)
{
// Configuramos el color con el que vamos a escribir en el control RichEdit.
        CHARFORMAT cf;
        cf.cbSize = sizeof(CHARFORMAT);
        cf.dwMask = CFM_COLOR;
        GetSelectionCharFormat(cf);
        cf.crTextColor = Color;
        cf.dwEffects &= ~CFE_AUTOCOLOR;
        SetSelectionCharFormat(cf);

        // No seleccionamos ningún texto del control RichEdit. Así al usar el
        // método ReplaceSel() añadiremos la nueva cadena al final.
        SetSel(-1,-1);
        ReplaceSel(lpszMsg);

// Mandamos un mensaje para que se produzca un autoscroll vertical para que
// la última línea sea visible (realmente se hace un scroll hacia la esquina
// inferior derecha del control RichEdit.
        SendMessage(EM_SCROLL, SB_BOTTOM, 0);
}

////////////////////////////////////////////////////////////////////////
//                                                                    //
// Método que sirve para borrar el contenido del control RichEditEx.  //
//                                                                    //
////////////////////////////////////////////////////////////////////////
void CRichEditEx::Borrar()
{
// Seleccionamos todo el texto del contro RicheEdit y lo reemplazamos por
// una cadena vacia.
        SetSel(0,-1);
        ReplaceSel(_T(""));
}
```

B.1.6. CPuertoSerie.

B.1.6.1. PuertoSerie.h

```
////////////////////////////////////////////////////////////////////////
//                                                                    //
//      Módulo : PuertoSerie.h                                        //
//                      Definición de la clase CPuertoSerie.          //
//                                                                    //
//      Propósito:      Clase para usar el puerto serie para comunicarse //
//                      con el controlador del robot Staübli RX-90.   //
//                                                                    //
//      Creado: Manuel Gómez Langley / 2003-2004                      //
//                                                                    //
////////////////////////////////////////////////////////////////////////

#include "RichEditEx.h"
```

```
#include "Corte3D.h"
#include "PuntoTrayectoria.h"


//////////////////// Macros / Estructuras / etc ////////////////////////////

#ifndef __PUERTOSERIE_H__
#define __PUERTOSERIE_H__

/////////////////////////// Clases ///////////////////////////////////////////

////// Clase CExcepcionPuertoSerie ///////////////////////////////////////////

void AfxLanzaExcepcionPuertoSerie(DWORD dwError = 0);

class CExcepcionPuertoSerie : public CException
{
public:
//Constructor / Destructor
        CExcepcionPuertoSerie(DWORD dwError);
        ~CExcepcionPuertoSerie();

//Methods

#ifdef _DEBUG
        virtual void Dump(CDumpContext& dc) const;
#endif
        virtual BOOL ObtenerMsjError(LPTSTR lpstrError, UINT nMaxError,     PUINT pnHelpContext =
NULL);
        CString ObtenerMsjError();

//Data members
        DWORD m_dwError;

protected:
        DECLARE_DYNAMIC(CExcepcionPuertoSerie)
};

//// Clase CPuertoSerie ////////////////////////////////////////////////////

class CPuertoSerie : public CObject
{
public:

// Enumeraciones.
        typedef enum
        {
            EBaud110 = CBR_110,
                EBaud300 = CBR_300,
                EBaud600 = CBR_600,
                EBaud1200 = CBR_1200,
                EBaud2400 = CBR_2400,
                EBaud4800 = CBR_4800,
                EBaud9600 = CBR_9600,
                EBaud19200 = CBR_19200,
```

```
                EBaud38400 = CBR_38400
        } ETasaBaudios;

        typedef enum
        {
            EDatos7 = 7,
                EDatos8 = 8
        } EBitsDatos;

        typedef enum
        {
            ENoParidad = NOPARITY,
            EParidadImpar = ODDPARITY,
            EParidadPar = EVENPARITY
        } EParidad;

        typedef enum
        {
            EStop1 = ONESTOPBIT,
                EStop2 = TWOSTOPBITS
        } EBitsParada;

        typedef enum
        {
                ENoControlFlujo,
                ECtsRts,
                EXonXoff
        } EControlFlujo;


// Estructura para almacenar las peticiones de escritura
        typedef struct PETICION_ESCRITURA
        {
                //HANDLE hHeap;
                char chBuffer[100];
                struct PETICION_ESCRITURA* pNodoAnterior;
                struct PETICION_ESCRITURA* pNodoSiguiente;
        } PETICION_ESCRITURA, *PPETICION_ESCRITURA;


//Constructor / Destructor.
        //      CPuertoSerie(CRichEditEx* pMostrar);
        CPuertoSerie();
        virtual ~CPuertoSerie();

// Métodos generales.
        //void Abrir(LPCTSTR lpszPuerto, ETasaBaudios eTasaBaudios, EParidad eParidad, EBitsDatos
eBitsDatos, EBitsParada eBitsParada,EControlFlujo eControlFlujo/*, BOOL bOverlapped*/);
        void Abrir(LPCTSTR lpszPuerto, ETasaBaudios eTasaBaudios, EParidad eParidad, EBitsDatos
eBitsDatos, EBitsParada eBitsParada,EControlFlujo eControlFlujo, CRichEditEx* pMostrar);
        void Cerrar();

        void ComenzarHilo();
        void TerminarHilo();
```

```
        BOOL EstaAbierto() const { return m_hComm != INVALID_HANDLE_VALUE; };

#ifdef _DEBUG
        void Dump(CDumpContext& dc) const;
#endif


// Procesos realizados en hilos independientes. PODRIAMOS DEFINIRLOS COMO PRIVATE
        static DWORD WINAPI ProcesoTrabajador(LPVOID lpParam);
        DWORD ProcesoTrabajador(void);

/*
Si intentamos usar un método miembro de una clase como funcion de un hilo (lo llamaremos
procedimiento), al compilar nos aparecerá un error como:

error C2664: 'CreateThread' : cannot convert parameter 3 from
'unsigned long (void *)' to 'unsigned long (__stdcall *)(void *)'
None of the functions with this name in scope match the target type

o

error C2665: 'AfxBeginThread' : none of the 2 overloads can convert
parameter 1 from type 'unsigned int (void *)'

El problema es que todo procedimiento tiene su propio prototipo, que determina los parámetros que se
le pasan desde el SO.
En C++ cada método miembro tiene un parámetro oculto (el puntero 'this') que es pasado a la función
automáticamente.
C++ usa este puntero 'this' para asociar la función a una instancia en particular de una clase. Así
los métodos miembro pueden acceder a las variables miembro a través de 'this'...

Como el SO no llama a los procedimientos a través de objetos, no puede manejar el puntero 'this' que
se añade automáticamente...
Para conseguir que un método miembro funcione como un procedimiento de un hilo es necesario
especificar al compilador que no espere un puntero 'this'.
Para evitar que se añada automáticamente hay 2 posibilidades:

Funciones que no sean métodos miembros de una clase: como no son parte de la clase no llevan puntero
'this'.
Métodos miembros 'static': no reciben el puntero 'this'.

Por lo tanto, si queremos usar un método miembro como un procedimiento de un hilo debemos declararlo
como 'static'.
Esto conlleva que:
- No es necesario crear un objeto antes de que un método miembro estático del mismo es usado o una
variable miembro estática es accedida.
- El operador de acceso de la clase puede acceder a miembros estáticos de la clase sin que exista
ninguna instancia de la misma.
- Un método miembro estático no puede acceder a miembros no estáticos de su propia clase si no existe
una instancia de la clase.
  Por tanto, todos los accesos a objetos deben ser explicitos o mediante un puntero al objeto.

Este último punto es la clave: un método miembro estático no puede acceder implicitamente a miembros
no estáticos de la clase.
```

```
Puesto que necesitamos acceder a variables miembro no estáticas desde el procedimiento del hilo
(definido como un método miembro estático)
hemos de usar una técnica que permita actuar a un método estático como si fuera no estático. El
método escogido consiste en
pasar como parámetro un puntero 'this' al procedimiento del hilo para que éste pueda llamar a un
método no estático de la clase,
que será donde se realice todo el trabajo del hilo.
*/

// Métodos para gestionar las peticiones de escritura.
        void InicializarElementosEscritura();
        void FinalizarElementosEscritura();

        void HacerPeticionEscritura(LPCSTR lpszBuffer);
        void PonerNodo(PPETICION_ESCRITURA pNodo);
        void EliminarNodo(PPETICION_ESCRITURA pNodoAEliminar);

        void ProcesarPeticionEscritura();
        void Escribir(PPETICION_ESCRITURA pNodo);

        void ProcesarEventoComm();
        void Leer(LPVOID lpBuffer, DWORD dwBytesALeer, DWORD* pBytesLeidos);
        void ProcesarDatosLeidos(LPCTSTR lpszBuffer);
        void CapturarDatosLeidos(Vector* vPunto);
        void ProcesarCaptura(LPCTSTR lpszBuffer);
//      void CapturarPuntoInicial(Joint *m_pAngulos);

// Métodos para lectura /escritura.
  //DWORD Read(void* lpBuf, DWORD dwCount);
        //BOOL  Leer(void* lpBuf, DWORD dwCount, OVERLAPPED& overlapped, DWORD* pBytesRead=NULL);

  //DWORD Write(const void* lpBuf, DWORD dwCount);
        //BOOL  Escribir(const void* lpBuf, DWORD dwCount, OVERLAPPED& overlapped, DWORD*
pBytesWritten=NULL);

//      void  GetOverlappedResult(OVERLAPPED& overlapped, DWORD& dwBytesTransferred, BOOL bWait);
        void  CancelIo();
/*  DWORD BytesWaiting();
        BOOL  DataWaiting(DWORD dwTimeout);

//Configuration Methods
  void GetConfig(COMMCONFIG& config);
  static void GetDefaultConfig(int nPort, COMMCONFIG& config);
  void SetConfig(COMMCONFIG& Config);
  static void SetDefaultConfig(int nPort, COMMCONFIG& config);

//Misc RS232 Methods
  void ClearBreak();
  void SetBreak();
  void ClearError(DWORD& dwErrors);
  void GetStatus(COMSTAT& stat);
*/  void FijarEstadoPuertoSerie(DCB& dcb);
        void ObtenerEstadoPuertoSerie(DCB& dcb);
/*  void Escape(DWORD dwFunc);
  void ClearDTR();
```

```
  void ClearRTS();
  void SetDTR();
  void SetRTS();
  void SetXOFF();
  void SetXON();
  void GetProperties(COMMPROP& properties);
  void GetModemStatus(DWORD& dwModemStatus);
*/
//Timeouts
        void FijarTimeouts(COMMTIMEOUTS& timeouts);
        void ObtenerTimeouts(COMMTIMEOUTS& timeouts);
        void FijarNoTimeouts();
/*  void Set0WriteTimeout();
  void Set0ReadTimeout();
*/
//Event Methods
  void FijarMascaraEventosPuertoSerie(DWORD dwMascara);
  void ObtenerMascaraEventosPuertoSerie(DWORD& dwMascara);
/*  void WaitEvent(DWORD& dwMask);
  BOOL WaitEvent(DWORD& dwMask, OVERLAPPED& overlapped);
*/
//Queue Methods
        void LimpiarBuffers();
/*  void Flush();
  void Purge(DWORD dwFlags);
  void TerminateOutstandingWrites();
  void TerminateOutstandingReads();
  void ClearWriteBuffer();
  void ClearReadBuffer();
  void Setup(DWORD dwInQueue, DWORD dwOutQueue);
*/
//Overridables
        virtual void OnCompletion(DWORD dwErrorCode, DWORD dwCount, LPOVERLAPPED lpOverlapped);

protected:
        HANDLE m_hComm;                                                 // Manejador para el
puerto serie
        HANDLE m_hEventoTerminaHilo;                    // Manejador de eventos necesario para
ordenar a los hilos que terminen su ejecución.
        HANDLE m_hHiloTrabajador;                       // Manejador para el hilo trabajador
        HANDLE m_hHeapEscritura;                        // Manejador para la memoria de
apilamiento donde se almacenan las peticiones de escritura.
//      HANDLE m_hEventoPermisoEscritura;       // Manejador de eventos necesario para indicar que
existe una petición de escritura.
        HANDLE m_hSemafPeticionEscritura;       // Manejador de semáforo necesario para gestionar
las peticiones de escritura.

        PPETICION_ESCRITURA m_pCabezaHeapEscritura;
        PPETICION_ESCRITURA m_pColaHeapEscritura;

        CRITICAL_SECTION m_csHeapEscritura;

        COMMTIMEOUTS m_CommTimeoutsOriginal;    // Estructura COMMTIMEOUTS para guardar los
valores originales antes de abrir el puerto serie.
```

```
        DWORD m_dwMascaraEventosOriginal;                   // Variable para guardar la mascara de
eventos de comunicaciones original del puerto serie abierto.

        BOOLEAN m_fCapturarDatosLeidos;                     // Bandera que indica cuando debemos coger
los datos leidos del puerto serie para calcular posiciones iniciales.
        Vector* m_pPunto;                                                   // Puntero al objeto
vector donde se almacenará el punto leido.
//      BOOLEAN m_fCapturarPuntoInicial;          // Bandera que indica cuando debemos coger los
datos leidos del puerto serie para calcular posiciones iniciales.
//      Joint* m_pPuntoInicial;                                     // Puntero al objeto vector donde
se almacenará el punto leido en coord. articulares.

        CRichEditEx* m_pRiedMostrar;


        static void WINAPI _OnCompletion(DWORD dwErrorCode, DWORD dwCount, LPOVERLAPPED
lpOverlapped);

        DECLARE_DYNAMIC(CPuertoSerie)
};


#endif //__PUERTOSERIE_H__
/*
A la hora de abrir el puerto de comunicaciones lo podemos hacer de 2 modos diferentes:
síncrono (non-overlapped) o asíncrono (overlapped).

  -     Modo síncrono (NON-OVERLAPPED): Es el modo tradicidional de hacer
        operaciones de entrada/salida: cuando se produce una petición para
        realizar una operación de E/S se  asume que ésta habrá terminado cuando
        la función implicada devuelva el control del programa. Es decir, la
        operación tiene lugar mientras el hilo llamante es bloqueado.
        Una vez que la operación termina, la función devuelve el control del
        programa y el hilo llamante continua su trabajo.
        Este modo es útil en aplicaciones multihilos, pues mientras un hilo está
        bloqueado realizando una operación de E/S, el resto puede continuar
        ejecutándose. Sin embargo, si un hilo se encuentra bloqueado en una
        operación de E/S, cualquier otro hilo que realice una llamada a una API
        de comunicación sobre el mismo puerto será bloqueado hasta que la
        operación anterior termine.
        Un factor a favor de usar este modo de funcionamiento es la
        portabilidad: no todos los SO permiten el modo OVERLAPPED. Sin embargo,
        casi todos soportan las aplicaciones multihilos. Por tanto, una buena elección son las
        aplicaciones multihilos que realizan operaciones de E/S
        de modo NON-OVERLAPPED.
        Particularmente interesante resulta usar este modo (multihilo + non-verlapped) junto con los
        eventos de comunicaciones. Esto permite que un programa lea datos de un puerto de
        comunicaciones sólo cuando han llegado, y durante el resto del tiempo el hilo
        permanece dormido, a diferencia de realizar una lectura que espera que los datos lleguen.
        Esta técnica evita la necesidad de estar consultando si han llegado datos constantemente,
        por lo que aumenta la eficiencia.
        Usando la bandera EV_RXCHAR para notificar al hilo que ha llegado un byte al puerto de
        comunicaciones, en cuyo caso usamos la función ReadFile para leer todo el buffer.
```

- Modo ásincrono (OVERLAPPED): No es tan sencillo pero es más flexible y eficiente. Un puerto de comunicaciones abierto en modo OVERLAPPED permite que se realicen distintas operaciones de E/S "al mismo tiempo" sobre el mismo puerto, pues cada éstas no tienen por qué terminar inmediatamente, sino que pueden quedar pendientes hasta ser completadas.
  En este tiempo intermedio mientras las operaciones de E/S están esperando se pueden realizar otras tareas (background work). En caso de que no sea necesario realizar ninguna tarea, la única ventaja del modo OVERLAPPED es que permite una mejor respuesta al usuario.
  Este modo puede ser utilizado tanto en aplicaciones de un solo hilo (que recibe peticiones y las atiende cuando puede, además de realizar trabajo extra mientras las operaciones están pendientes) como en aplicaciones multihilo (donde cada hilo puede realizar operaciones de E/S sobre le mismo puerto). En ambos casos es necesario que exista algún tipo de sincronización entre las llegada de peticiones y el procesamiento
  de los resultados de las mismas.
  Una operación de E/S en modo OVERLAPPED tiene 2 partes: la creación de la operación y la detección de su finalización. Crear la operación consiste en crear una estructura OVERLAPPED y rellenarla, crear un evento con reset manual para sincronización, y llamar a la función ReadFile o WriteFile. La operación puede o no terminar inmediatamente. Si es así, el programa debe continuar ejecutándose normalmente. Si no, debemos esperar a que se complete. Detectar el fin de la operación consiste en esperar a que el evento de sincronización ocurra, chequear el resultado de la espera, y procesar los datos si ésta ha terminado.

```
*/
/*
Los programas ejecutables creados con estos compiladores dividen la memoria disponible en varios
segmentos , uno para el código ( en lenguaje máquina ) , otro para albergar las variables globales ,
otro para el stack ( a travez del cual se pasan argumentos y donde residen las variables locales ) y
finalmente un último segmento llamado memoria de apilamiento ó amontonamiento ( Heap ) .
El Heap es la zona destinada a albergar a las variables dinámicas , es decir aquellas que crecen ( en
el sentido de ocupación de memoria ) y decrecen a lo largo del programa , pudiendose crear y
desaparecer (desalojando la memoria que ocupaban) en cualquier momento de la ejecución .
*/
```

B.1.6.2. PuertoSerie.cpp

```
////////////////////////////////////////////////////////////////////
//                                                                //
//      Módulo : PuertoSerie.cpp                                  //
//                       Implementación de la clase CPuertoSerie. //
//                                                                //
//      Creado:          Manuel Gómez Langley / 2003-2004          //
//                                                                //
////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////////////
// Incluimos el fichero de cabecera estándar
////////////////////////////////////////////////////////////////////////////

#include "stdafx.h"
//#include <tchar.h>

////////////////////////////////////////////////////////////////////////////
// Incluimos los ficheros de cabecera del módulo y del programa proyecto
////////////////////////////////////////////////////////////////////////////

#include "PuertoSerie.h"
```

```
#ifndef _WINERROR_
#include "winerror.h"
#endif

/////////////////////////////////////////////////////////////////////////////
// Habilitamos el 'debug memory manager'
/////////////////////////////////////////////////////////////////////////////

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

/////////////////////////////////////////////////////////////////////////////
// Código
/////////////////////////////////////////////////////////////////////////////

//////////////////////////////// Implementación ////////////////////////////

//Class which handles CancelIo function which must be constructed at run time
//since it is not imeplemented on NT 3.51 or Windows 95. To avoid the loader
//bringing up a message such as "Failed to load due to missing export...", the
//function is constructed using GetProcAddress. The CSerialPort::CancelIo
//function then checks to see if the function pointer is NULL and if it is it
//throws an exception using the error code ERROR_CALL_NOT_IMPLEMENTED which
//is what 95 would have done if it had implemented a stub for it in the first
//place !!

class _SERIAL_PORT_DATA
{
public:
//Constructors /Destructors
  _SERIAL_PORT_DATA();
  ~_SERIAL_PORT_DATA();

  HINSTANCE m_hKernel32;
  typedef BOOL (WINAPI CANCELIO)(HANDLE);
  typedef CANCELIO* LPCANCELIO;
  LPCANCELIO m_lpfnCancelIo;
};

_SERIAL_PORT_DATA::_SERIAL_PORT_DATA()
{
  m_hKernel32 = LoadLibrary(_T("KERNEL32.DLL"));
  VERIFY(m_hKernel32 != NULL);
  m_lpfnCancelIo = (LPCANCELIO) GetProcAddress(m_hKernel32, "CancelIo");
}

_SERIAL_PORT_DATA::~_SERIAL_PORT_DATA()
{
  FreeLibrary(m_hKernel32);
  m_hKernel32 = NULL;
}
```

```
//The local variable which handle the function pointers

_SERIAL_PORT_DATA _SerialPortData;

////////// Código manejador de excepciones

void AfxLanzaExcepcionPuertoSerie(DWORD dwError /* = 0 */)
{
        if (dwError == 0)
                dwError = ::GetLastError();

        CExcepcionPuertoSerie* pExcepcion = new CExcepcionPuertoSerie(dwError);

        TRACE(_T("AVISO: CExcepcionPuertoSerie lanzada debido a error %d.\n"), dwError);
        THROW(pExcepcion);
}

BOOL CExcepcionPuertoSerie::ObtenerMsjError(LPTSTR pstrError, UINT nMaxError, PUINT pnHelpContext)
{
        ASSERT(pstrError != NULL && AfxIsValidString(pstrError, nMaxError));

        if (pnHelpContext != NULL)
                *pnHelpContext = 0;

        LPTSTR lpBuffer;
        BOOL bRet = FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER | FORMAT_MESSAGE_FROM_SYSTEM,
                                                NULL,  m_dwError, MAKELANGID(LANG_NEUTRAL,
SUBLANG_SYS_DEFAULT),
                                                (LPTSTR) &lpBuffer, 0, NULL);

        if (bRet == FALSE)
                *pstrError = '\0';
        else
        {
                lstrcpyn(pstrError, lpBuffer, nMaxError);
                bRet = TRUE;

                LocalFree(lpBuffer);
        }
        return bRet;
}

CString CExcepcionPuertoSerie::ObtenerMsjError()
{
  CString rVal;
  LPTSTR pstrError = rVal.GetBuffer(4096);
  ObtenerMsjError(pstrError, 4096, NULL);
  rVal.ReleaseBuffer();
  return rVal;
}

CExcepcionPuertoSerie::CExcepcionPuertoSerie(DWORD dwError):
        m_dwError(dwError)
{
}
```

```
CExcepcionPuertoSerie::~CExcepcionPuertoSerie()
{
}

IMPLEMENT_DYNAMIC(CExcepcionPuertoSerie, CException)

#ifdef _DEBUG
void CExcepcionPuertoSerie::Dump(CDumpContext& dc) const
{
        CObject::Dump(dc);

        dc << "m_dwError = " << m_dwError;
}
#endif

////////// Código de la clase CPuertoSerie

//CPuertoSerie::CPuertoSerie(CRichEditEx* pMostrar):
CPuertoSerie::CPuertoSerie():
        m_hComm(INVALID_HANDLE_VALUE),
        m_hEventoTerminaHilo(INVALID_HANDLE_VALUE),
        m_hHiloTrabajador(INVALID_HANDLE_VALUE),
        m_hHeapEscritura(INVALID_HANDLE_VALUE),
        m_hSemafPeticionEscritura(INVALID_HANDLE_VALUE),
//      m_hEventoPermisoEscritura(INVALID_HANDLE_VALUE),
        m_pCabezaHeapEscritura(NULL),
        m_pColaHeapEscritura(NULL),
        m_fCapturarDatosLeidos(FALSE)
//      m_pRiEdMostrar(pMostrar)
//      m_fTerminar(FALSE)
//      m_bOverlapped = TRUE;
//      m_hEvent = NULL;
{
}

CPuertoSerie::~CPuertoSerie()
{
        Cerrar();
}

IMPLEMENT_DYNAMIC(CPuertoSerie, CObject)

#ifdef _DEBUG
void CPuertoSerie::Dump(CDumpContext& dc) const
{
        CObject::Dump(dc);

        dc << _T("m_hComm = ") << m_hComm << _T("\n");
//      dc << _T("m_bOverlapped = ") << m_bOverlapped;
}
#endif


//////////////////////////////////////////////////////////////////////////
```

```
//                                                                          //
// Método miembro que sirve para abrir el puerto de comunicaciones          //
// elegido con la configuración escogida en el cuadro de diálogos que lo    //
// llama. Siempre abrimos el puerto en modo asíncrono (OVERLAPPED).         //
// Una vez que el puerto está abierto inicia un hilo trabajador             //
// encargado de gestionar la lectura y escritura de datos por dicho         //
// puerto.                                                                  //
//                                                                          //
// Parámetros:                                                              //
//      - Puntero a cadena de TCHAR cosntante (CString) que contiene el     //
//        puerto COM a abrir.                                               //
//      - Variables de tipo enumerado para las distintas características    //
//        con las que configuramos la comunicación por dicho puerto.        //
//                                                                          //
//////////////////////////////////////////////////////////////////////////////
void CPuertoSerie::Abrir(LPCTSTR lpszPuerto, ETasaBaudios eTasaBaudios, EParidad eParidad, EBitsDatos
eBitsDatos, EBitsParada eBitsParada,EControlFlujo eControlFlujo, CRichEditEx* pMostrar)
{
        // Asignamos el puntero al objeto CRichEditEx, que utilizamos para poder escribir en él.
        m_pRiedMostrar = pMostrar;

        // Por si acaso estaba abierto, antes de abrir el puerto lo cerramos.
        Cerrar();

        // Llamamos a CreateFile para abrir el puerto de comunicaciones.
        m_hComm = CreateFile(lpszPuerto, GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING,
FILE_FLAG_OVERLAPPED /*0*/, NULL);
        if (m_hComm == INVALID_HANDLE_VALUE)
        {
                TRACE(_T("ERROR en Abrir(): Fallo al intentar abrir el puerto serie. GetLastError():
%d.\n"), GetLastError());
                AfxLanzaExcepcionPuertoSerie();
        }

        // Obtenemos el estado original del puerto antes de cambiarlo para después poder
restaurarlo.
        DCB dcb;
        dcb.DCBlength = sizeof(DCB);
        ObtenerEstadoPuertoSerie(dcb);

        // Fijamos la tasa de bits seleccionada.
        dcb.BaudRate = DWORD(eTasaBaudios);

        // Fijamos tipo de paridad seleccionado.
        dcb.Parity = BYTE(eParidad);

        // Fijamos el número de bits de datos seleccionado.
        dcb.ByteSize = BYTE(eBitsDatos);

        // Fijamos el número de bits de parada seleccionado.
        dcb.StopBits = BYTE(eBitsParada);

        // Fijamos el tipo de control de flujo seleccionado.
        dcb.fDsrSensitivity = FALSE;              //MIRAR PARA QUE SIRVE ESTO
        switch (eControlFlujo)
```

```
	{
		case ENoControlFlujo:
			dcb.fOutxCtsFlow = FALSE;
			dcb.fOutxDsrFlow = FALSE;
			dcb.fDtrControl = DTR_CONTROL_DISABLE;
			dcb.fRtsControl = RTS_CONTROL_DISABLE;
			dcb.fOutX = FALSE;
			dcb.fInX = FALSE;
			break;

		case ECtsRts:                              // MIRAR SI CONFIG CORRECTA
			dcb.fOutxCtsFlow = TRUE;
			dcb.fOutxDsrFlow = TRUE;
			dcb.fDtrControl = DTR_CONTROL_HANDSHAKE;
			dcb.fRtsControl = RTS_CONTROL_HANDSHAKE;
			dcb.fOutX = FALSE;
			dcb.fInX = FALSE;
			break;

		case EXonXoff:
			dcb.fOutxCtsFlow = FALSE;
			dcb.fOutxDsrFlow = FALSE;
			dcb.fDtrControl = DTR_CONTROL_DISABLE;
			dcb.fRtsControl = RTS_CONTROL_DISABLE;
			dcb.fOutX = TRUE;
			dcb.fInX = TRUE;
			dcb.XonChar = 0x11;
			dcb.XoffChar = 0x13;
			dcb.XoffLim = 100;
			dcb.XonLim = 100;
			break;
	}

	// Una vez que hemos fijado la configuración deseada, la aplicamos en el puerto serie
abierto.
	FijarEstadoPuertoSerie(dcb);

	// Almacenamos la mascara de eventos de comunicaciones original y activamos el evento
EV_RXCHAR para notificar llegada de datos que leer.
	ObtenerMascaraEventosPuertoSerie(m_dwMascaraEventosOriginal);
	DWORD dwMascara = EV_RXCHAR;
	FijarMascaraEventosPuertoSerie(dwMascara);

	// Ahora almacenamos la estructura COMMTIMEOUTS antigua (controla los timeouts de las
comunicaciones en el puerto abierto) y ponemos nuestros propios valores. Cuando cerremos el puerto
serie restauraremos los valores originales.
	ObtenerTimeouts(m_CommTimeoutsOriginal);
	COMMTIMEOUTS Timeouts = { 0x100, 0, 0, 0, 0 };
	FijarTimeouts(Timeouts);
	//FijarNoTimeouts();

	// Finalmente vaciamos los buffers de entrada y salida.
	LimpiarBuffers();

	// Ahora lanzamos los hilos lector y escritor.
```

```
	ComenzarHilo();
}


////////////////////////////////////////////////////////////////////////
//                                                                    //
// Método miembro que sirve para finalizar el hilo encargado de la    //
// lectura y escritura y para cerrar el puerto de comunicaciones abierto.  //
//                                                                    //
////////////////////////////////////////////////////////////////////////
void CPuertoSerie::Cerrar()
{
	if (EstaAbierto())
	{
		// Primero cerramos los hilos lector y escritor.
		TerminarHilo();

		// Ahora restauramos la estructura COMMTIMEOUTS antigua.
		FijarTimeouts(m_CommTimeoutsOriginal);

		// Restauramos la mascara de eventos de comunicaciones original.
		FijarMascaraEventosPuertoSerie(m_dwMascaraEventosOriginal);

		// Restauramos cualquier otra configuración del puerto.

		// Vaciamos todos los buffers de entrada y salida del puerto
		LimpiarBuffers();

		// Finalmente cerramos los puertos de comunicaciones abiertos
		BOOL bSinError = CloseHandle(m_hComm);
		m_hComm = INVALID_HANDLE_VALUE;
		if (!bSinError)
			TRACE(_T("ERROR en Cerrar(): Fallo al intentar cerrar el puerto.
GetLastError(): %d.\n"), GetLastError());
	}
}

////////////////////////////////////////////////////////////////////////
//                                                                    //
// Método miembro que sirve para inicializar los elementos de         //
// sincronización necesarios para terminar el hilo y para crear el    //
// propio hilo, indicando cual es la rutina que debe realizar.        //
//                                                                    //
////////////////////////////////////////////////////////////////////////
void CPuertoSerie::ComenzarHilo()
{
	DWORD dwHiloTrabajadorId = 0;

	//Creamos el evento que ordena terminar al hilo.
	m_hEventoTerminaHilo = CreateEvent(NULL, TRUE, FALSE, NULL);
	if (m_hEventoTerminaHilo == NULL)
		//AVISAR DEL ERROR Y ABORTAR
		TRACE(_T("ERROR en ComenzarHilos(): Fallo al intentar crear el evento para finalizar
el hilo trabajador.\n"));
	else
```

```
                TRACE(_T("ComenzarHilos(): Creado evento para finalizar el hilo trabajador.\n"));

        // Le pasamos como parámetro un puntero a this para que pueda acceder a variables miembro no
estáticas.
        m_hHiloTrabajador = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)ProcesoTrabajador,
(LPVOID)this, 0, &dwHiloTrabajadorId);
        if (m_hHiloTrabajador == NULL)
        {
                //AVISAR DEL ERROR Y ABORTAR
                TRACE(_T("ERROR en ComenzarHilos(): Fallo al intentar crear el hilo
trabajador.\n"));
                AfxLanzaExcepcionPuertoSerie();
        }
        else
                TRACE(_T("ComenzarHilos(): Creado hilo trabajador.\n"));
}


////////////////////////////////////////////////////////////////////////
//                                                                    //
// Método miembro que sirve para dinalizar el hilo y los elementos de //
// sincronización asociados al mismo.                                 //
//                                                                    //
////////////////////////////////////////////////////////////////////////
void CPuertoSerie::TerminarHilo()
{
        DWORD dwRes;

        // Marcamos el evento que obliga a los hilos lector y escritor a terminar.
        SetEvent(m_hEventoTerminaHilo);

        // Esperamos a que los hilos terminen
        dwRes = WaitForSingleObject(m_hHiloTrabajador, 2000);                    // Definir
constante TIMEOUT_FIN_HILOS mjor, no cm #define, sino cm const int
        switch(dwRes)
        {
                case WAIT_OBJECT_0:
                        break;

        case WAIT_TIMEOUT:
            // AVISAR ERROR Y ABORTAR
                        TRACE(_T("ERROR en TerminarHilo(): El hilo trabajador no termina.\n"));
            break;

                default:
            // AVISAR Q EXISTE UN ERROR EN WaitForMultipleObjects, probablement con hHilos
                        TRACE(_T("ERROR en TerminarHilo(): Fallo en WaitForMultipleObjects().\n"));
            break;
        }

    // Reseteamos el evento.
    ResetEvent(m_hEventoTerminaHilo);

        // Cerramos los manejadores del evento y de los hilos.
        BOOL bSinError = CloseHandle(m_hEventoTerminaHilo);
```

```
	m_hEventoTerminaHilo = INVALID_HANDLE_VALUE;
	if (!bSinError)
		TRACE(_T("ERROR en TerminarHilos(): Fallo al intentar cerrar el manejador del evento
para terminar los hilos. GetLastError(): %d.\n"), GetLastError());

	bSinError = CloseHandle(m_hHiloTrabajador);
	m_hHiloTrabajador = INVALID_HANDLE_VALUE;
	if (!bSinError)
		TRACE(_T("ERROR en TerminarHilos(): Fallo al intentar cerrar el manejador del hilo
trabajador. GetLastError(): %d.\n"), GetLastError());

}

////////////////////////////////////////////////////////////////////////
//                                                                    //
// Método miembro que hace las veces de hilo trabajador. Es el proceso //
// en paralelo que se encarga de realizar la lectura y escritura por el //
// puerto serie de forma transparente al usuario.                     //
//                                                                    //
////////////////////////////////////////////////////////////////////////
DWORD WINAPI CPuertoSerie::ProcesoTrabajador(LPVOID lpParam)
{
	//TRACE(_T("Dentro del hilo trabajador.\n"));

	// Llamamos al método del objeto actual que realiza el trabajo.
	CPuertoSerie* pThis = reinterpret_cast<CPuertoSerie*>(lpParam);
	return pThis->ProcesoTrabajador();
}

DWORD CPuertoSerie::ProcesoTrabajador()
{
	HANDLE hEventos[3];
	DWORD dwEventoComm;
	DWORD dwRes;
//	DWORD dwRes2;
	DWORD dwOvRes;
	OVERLAPPED osEventoComm = {0};
	BOOL fTerminar = FALSE;
	BOOL fEsperandoEventoComm = FALSE;

	// Inicializamos los elementos necesarios para realizar escrituras.
	InicializarElementosEscritura();

	// Inicializamos los elementos necesarios para realizar lecturas.
	osEventoComm.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
	if (osEventoComm.hEvent == NULL)
		// AVISAR ERROR Y ABORTAR
		TRACE(_T("ERROR en ProcesoTrabajador(): Fallo al crear el evento de la estructura
OVERLAPPED para notificar la llegada de eeventos de comunicaciones.\n"));

	// Rellenamos el array con los eventos que queremos esperar.
	hEventos[0] = m_hEventoTerminaHilo;
	hEventos[1] = osEventoComm.hEvent;
//	hEventos[2] = m_hEventoPermisoEscritura;
	hEventos[2] = m_hSemafPeticionEscritura;
```

```
	while(!fTerminar)
	{
		// Realizamos un chequeo del estado de los eventos del puerto serie si no se ha
producido uno ya.
		if (!fEsperandoEventoComm)
		{
			if (!WaitCommEvent(m_hComm, &dwEventoComm, &osEventoComm))
			{
				if (GetLastError() == ERROR_IO_PENDING)
				{
					TRACE(_T("ProcesoTrabajador(): Esperando lectura.\n"));
					fEsperandoEventoComm = TRUE;
				}
				else
					// AVISAR ERROR Y ABORTAR
					TRACE(_T("ERROR en ProcesoTrabajador(): Fallo en
WaitCommEvent().\n"));
			}

			else
			{
				TRACE(_T("ProcesoTrabajador(): WaitCommEvent() terminó
inmediatamente.\n"));

				// WaitCommEvent terminó inmediatamente. Como el
				// único evento que aviso es la llegada de
				// caracteres por el puerto serie, realizar una
				// lectura.
		ProcesarEventoComm();
			}
		}

		else
		{
			// Esperamos a que ocurra algún evento.
			dwRes = WaitForMultipleObjects(3, hEventos, FALSE, 1000);
	//#define CHEQUEOEVENTOSTIMEOUT 500
			switch(dwRes)
			{
				case WAIT_OBJECT_0:
					fTerminar = TRUE;
					TRACE(_T("ProcesoTrabajador(): El hilo trabajador va a
terminar.\n"));
					break;

				case WAIT_OBJECT_0 + 1:
					//TRACE(_T("ProcesoTrabajador(): WAIT_OBJECT_0+1.\n"));

		// Tengo que desactivar el aviso de eventos debido a la llegada de múltiples bytes
seguidos. Solo quiero que se me avise de la llegada del primer caracter.
		// Cuando realice la lectura debo activar de nuevo el evento.


				if (!GetOverlappedResult(m_hComm, &osEventoComm, &dwOvRes, FALSE))
```

```
                        {
                                // Ha ocurrido un error en la operación OVERLAPPED.
                                // Averiguamos cuál es el error con GetLastError().
                                // Avisamos del error y si es fatal, abortamos.
                                        TRACE(_T("ERROR en ProcesoTrabajador(): Fallo en
WaitForMultipleObjects(). GetLastError(): %d.\n"), GetLastError());
                                }
                                else
                        {
                                // Se ha activado algún evento de comunicaciones. Lo
                                // atendemos.
                                        ProcesarEventoComm();

                                        // Ahora fijamos la bandera fEsperandoEventoComm =
FALSE para empezar una nueva espera.
                                        fEsperandoEventoComm = FALSE;
                                }
                                break;

                                case WAIT_OBJECT_0 + 2:
                                //TRACE(_T("ProcesoTrabajador(): WAIT_OBJECT_0+2.\n"));

                                TRACE(_T("ProcesoTrabajador(): Petición de escritura
atendida.\n"));
                                ProcesarPeticionEscritura();

        /*      // Comprobamos si existe una petición de escritura para atender.
                                        dwRes2 = WaitForSingleObject(m_hSemafPeticionEscritura, 0);
                                        switch (dwRes2)
                                        {
                                                case WAIT_OBJECT_0:
                                                        TRACE(_T("ProcesoTrabajador(): Petición de
escritura atendida.\n"));
                                                        ProcesarPeticionEscritura();
                                                        break;

                                                case WAIT_TIMEOUT:
                                                        TRACE(_T("ProcesoTrabajador(): Hay permiso
de escritura pero nada que escribir.\n"));
                                                        break;
                                        }
                        */
                                break;

                                case WAIT_TIMEOUT:
                                        TRACE(_T("ProcesoTrabajador(): TIMEOUT -> No han ocurrido
eventos.\n"));
                                        break;

                        default:
                                // AVISAR ERROR Y ABORTAR
                                TRACE(_T("ERROR en ProcesoTrabajador(): Fallo en
WaitForMultipleObjects().\n"));
                                return 0;
                                break;
```

```
                        }
                }
        }

        // Cerramos los elementos necesarios para la escritura.
        FinalizarElementosEscritura();

        // Cerramos el manejador de eventos necesario para la estructura OVERLAPPED de espera de
eventos de comunicaciones.
        CloseHandle(osEventoComm.hEvent);

        return 1;
}
////////////////////////////////////////////////////////////////////////////
//                                                                        //
//                                                                        //
////////////////////////////////////////////////////////////////////////////
void CPuertoSerie::ProcesarEventoComm()
{
        // En nuestro caso, como sólo hemos activado el evento que indica la recepción de un byte
        // en el puerto serie, cada vez que se active este evento hay que leer todo el contenido
        // del buffer de llegada.

        // Creamos un buffer limpio.
        DWORD dwBytesLeidos = 0;
        DWORD dwTotalBytesLeidos = 0;
        CString strBuffer;
        char chBuffer[301];              //definir const int tam_buffer_lectura = 512;
        const int nLongBuffer = sizeof(chBuffer) - 1;

        // Obtenemos los datos del puerto de comunicaciones.
        Leer(chBuffer, nLongBuffer, &dwBytesLeidos);

        // Ponemos el caracter terminador al final de cada buffer lleno.
        chBuffer[dwBytesLeidos] = '\0';

        // Añadimos la cadena al buffer contenido en el objeto CString.
        strBuffer += chBuffer;

        // Si hemos leido datos, se procesan.
        if (dwBytesLeidos)
        {
                TRACE(_T("ProcesarEventoComm(): Hemos leido: %d caracteres.\n"),dwBytesLeidos);
                //TRACE(_T("ProcesarEventoComm(): Hemos leido:\n%s\n"),strBuffer);

                ProcesarDatosLeidos(strBuffer);

        //      if (!SetEvent(m_hEventoPermisoEscritura))
        //              TRACE(_T("ERROR en ProcesarEventoComm(): Fallo al marcar el evento de
permiso de escritura.\n"));
        }
}

/*
void CPuertoSerie::ProcesarEventoComm()
```

```
{
        // En nuestro caso, como sólo hemos activado el evento que indica la recepción de un byte
        // en el puerto serie, cada vez que se active este evento hay que leer todo el contenido
        // del buffer de llegada.

        // Creamos un buffer limpio.
        DWORD dwBytesLeidos = 0;
        DWORD dwTotalBytesLeidos = 0;
        CString strBuffer;
        char chBuffer[301];                                     //definir const int
tam_buffer_lectura = 512;
        const int nLongBuffer = sizeof(chBuffer) - 1;

        // Realizamos lecturas mientras haya leido el máximo de datos posibles, pues es probable que
        // queden datos en el buffer de entrada.
        do
        {
                // Obtenemos los datos del puerto de comunicaciones.
                Leer(chBuffer, nLongBuffer, &dwBytesLeidos);

                // Ponemos el caracter terminador al final de cada buffer lleno.
                chBuffer[dwBytesLeidos] = '\0';

                // Añadimos la cadena al buffer contenido en el objeto CString.
                        strBuffer += chBuffer;

                // Actualizamos el contador de bytes leidos en total.
                dwTotalBytesLeidos += dwBytesLeidos;

        }while (dwBytesLeidos == nLongBuffer);

        //LimpiarBuffers();

        if (dwTotalBytesLeidos)
        {
                TRACE(_T("ProcesarEventoComm(): Hemos leido: %d caracteres.\n"),dwTotalBytesLeidos);
                //TRACE(_T("ProcesarEventoComm(): Hemos leido:\n%s\n"),strBuffer);

                CString strDebug;
                strDebug.Format(_T("PROC_COMM: ")+strBuffer);

                ProcesarDatosLeidos(strDebug);
                //ProcesarDatosLeidos(strBuffer);

        //      if (!SetEvent(m_hEventoPermisoEscritura))
        //              TRACE(_T("ERROR en ProcesarEventoComm(): Fallo al marcar el evento de
permiso de escritura.\n"));
        }
}
*/

////////////////////////////////////////////////////////////////////////
//                                                                    //
//                                                                    //
////////////////////////////////////////////////////////////////////////
```

```
void CPuertoSerie::Leer(LPVOID lpBuffer, DWORD dwBytesALeer, DWORD* pBytesLeidos)
{
	OVERLAPPED osLector = {0};
	COMSTAT ComStat;
	DWORD dwCommErrors;
//	DWORD dwBytesALeer = 0;
	//BOOL fEsperandoLectura = FALSE;


	// Determinamos los bytes que hay que leer.
	ClearCommError(m_hComm, &dwCommErrors, &ComStat);

/*	if ((sizeof(lpBuffer)-1) > (DWORD)ComStat.cbInQue)
		dwBytesALeer = (DWORD)ComStat.cbInQue;
	else
		dwBytesALeer = (DWORD)(sizeof(lpBuffer) - 1);

	if (dwBytesALeer > 0)
*/
	if (ComStat.cbInQue > 0)
	{
		// Creamos el evento de la estructura OVERLAPPED.
		osLector.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
		if (osLector.hEvent == NULL)
			// AVISAR ERROR Y ABORTAR
			TRACE(_T("ERROR en Leer(): Fallo al crear el evento de la estructura
OVERLAPPED para lectura.\n"));

		if (!ReadFile(m_hComm, lpBuffer, dwBytesALeer, pBytesLeidos, &osLector))
	//leemos solo 300 para poder poner al terminador al final
		{
			// O se han producido errores o la lectura está pausada.
			if (GetLastError() != ERROR_IO_PENDING)
			{
				//AVISAR DEL ERROR Y ABORTAR
				TRACE(_T("ERROR en Leer(): Fallo al llamar a ReadFile.
GetLastError(): %d.\n"), GetLastError());
				//AfxLanzaExcepcionPuertoSerie();
				//return 0;
			}

			else
// Se han leido datos y la lectura está pausada. Esperamos que termine.
			{
				if (!GetOverlappedResult(m_hComm, &osLector, pBytesLeidos, TRUE))
				{
					//AVISAR DEL ERROR Y ABORTAR
					TRACE(_T("ERROR en Leer(): Fallo en GetOverlappedResult().
GetLastError(): %d.\n"), GetLastError());
					//AfxLanzaExcepcionPuertoSerie();
					//return 0;
				}
				//else
				//	TRACE(_T("Leer(): Tras espera hemos leido:
%s.\n"),lpBuffer);
```

```
/*                              // Puedo cambiar este código por el equivalente:

                                DWORD dwRes = WaitForSingleObject(osLector.hEvent, INFINITE);
                                switch(dwRes)
                                {
                                        case WAIT_OBJECT_0:
        // Lectura completada. Si no se han producido errores no se hace nada.
                                                if (!GetOverlappedResult(m_hComm, &osLector,
pBytesLeidos, FALSE))
                                                {
                                                        //AVISAR DEL ERROR Y ABORTAR
                                                        TRACE(_T("ERROR en Leer(): Fallo en
GetOverlappedResult(). GetLastError(): %d.\n"), GetLastError());
                                                        //AfxLanzaExcepcionPuertoSerie();
                                                        //return 0;
                                                }
                                                else
                                                        //TRACE(_T("Leer(): Tras espera hemos
leido: %s.\n"),lpBuffer);
                                                break;

                                        case WAIT_TIMEOUT:
                                                TRACE(_T("Leer(): Timeout.\n"));
                                                // Este caso no puede ocurrir. Quitar si todo OK.
                                                break;

                                        default:
                                                //AVISAR DEL ERROR Y ABORTAR
                                                TRACE(_T("ERROR en Leer(): Fallo en
WaitForSingleObject(). GetLastError(): %d.\n"), GetLastError());
                                                //AfxLanzaExcepcionPuertoSerie();
                                                //return 0;
                                                break;
                        }
*/
                        }
                }

                else
                {
                        //TRACE(_T("Leer(): Inmediatamente hemos leido: %s.\n"),lpBuffer);
                        // Se han leido datos y la lectura terminó inmediatamente.
                        // Simplemente para estar seguros marcamos el evento de la
                        // estructura OVERLAPPED.

                        if (&osLector)
                                SetEvent(osLector.hEvent);
                }

        // Cerramos el manejador de eventos necesario para la estructura OVERLAPPED de lectura.
        CloseHandle(osLector.hEvent);

        }
}
```

```
////////////////////////////////////////////////////////////////////////
//                                                                    //
//                                                                    //
////////////////////////////////////////////////////////////////////////
void CPuertoSerie::ProcesarDatosLeidos(LPCTSTR lpszBuffer)
{
        //      int nIndex;

//      for (nIndex = 0; nIndex < 1000; nIndex++)
//              if (lpszBuffer[nIndex] == 'Ì')
//                      lpszBuffer[nIndex] = '\0';

        // Uso el constructor de Cstring q acepta LPCSTR
        CString  strBuffer = lpszBuffer;

        // Cuando mandamos un comando+RC, el controlador nos devuelve el
comando+ESC+'['+'4'+'l'+RC+LF.
        // Hemos de filtrar estos caracteres. Para ello buscamos la cadena anterior y la eliminamos.
        CString strBasura;
        CString strVacio;

        strBasura.Format(_T("\x1B[4l\r\n"));
        strVacio.Format(_T("\n"));
        strBuffer.Replace(strBasura,strVacio);

        strBasura.Format(_T("\x7"));
        strVacio.Format(_T(""));
        strBuffer.Replace(strBasura,strVacio);

        // Muestro por pantalla en rojo cada cadena recibida
        m_pRiedMostrar->Mostrar(strBuffer,RGB(255,0,0));

        // Si está activada la bandera de datos caputrados, los procesamos.
        if (m_fCapturarDatosLeidos)
        {
                ProcesarCaptura(strBuffer);
                m_fCapturarDatosLeidos = FALSE;
        }



        //TRACE(_T("ProcesarDatosLeidos(): %s\n"),lpszBuffer);
}

////////////////////////////////////////////////////////////////////////////
//
//
////////////////////////////////////////////////////////////////////////////
void CPuertoSerie::CapturarDatosLeidos(Vector* pPunto)
{
        CString strOrden;

        // Enviamos un comando HERE para que el controlador nos devuelva la posición del extremo.
        CCorte3DApp* pAplicacion = (CCorte3DApp*)AfxGetApp();
```

```
	strOrden.Format(_T("WHERE"));
	pAplicacion->EnviarOrden((LPCTSTR)strOrden);

	m_pPunto = pPunto;
	m_fCapturarDatosLeidos = TRUE;

	// SI PUEDE DEBE DECIRLE AL OTRO METODO DE LECTURA QUE LO ALMACENE AQUI (podriamos usar un
vAuxiliar y asi no hay que indicarle en cual de los 3 hay q escribir)
}

////////////////////////////////////////////////////////////////////////////
//
//
////////////////////////////////////////////////////////////////////////////
void CPuertoSerie::ProcesarCaptura(LPCTSTR lpszBuffer)
{
	CString strBuffer = lpszBuffer;
	CString strDatos;
	int nIndex;
	int nIndex2;
	double dPunto[3];

	nIndex = strBuffer.Find(_T("Hand"));
	strBuffer = strBuffer.Right(strBuffer.GetLength() - nIndex - sizeof(_T("Hand")));

	for (nIndex = 0; nIndex < 3; nIndex++)
	{
		// Cojo los valores de las coordenadas cartesianas X,Y y Z.
		strBuffer.TrimLeft();
		for (nIndex2 = 0; strBuffer.GetAt(nIndex2) != ' '; nIndex2++);
		strDatos = strBuffer.Left(nIndex2);
		strBuffer = strBuffer.Right(strBuffer.GetLength() - nIndex2);

		// Los almaceno en la instancia Vector adecuada.
		//dPunto[nIndex] =  _tstof(strDatos);
		dPunto[nIndex] = atof(strDatos);

		//AfxMessageBox(strDatos + _T("\n") + strBuffer,MB_ICONSTOP|MB_OK);
	}
	m_pPunto->x = dPunto[0];
	m_pPunto->y = dPunto[1];
	m_pPunto->z = dPunto[2];

	TRACE(_T("DATOS CAPTURADOS.\n"));
}


////////////////////////////////////////////////////////////////////////////
//
//
////////////////////////////////////////////////////////////////////////////
void CPuertoSerie::InicializarElementosEscritura()
{
	// Inicializamos la CRITICAL_SECTION que nos permite manejar las peticiones de escritura
	// sin que existan condiciones de carrera entre el hilo escritor y el método que añade
```

```
        // peticiones de escritura en la lista enlazada.
        InitializeCriticalSection(&m_csHeapEscritura);

        // Creamos una memoria de apilamiento (HEAP) para las peticiones de escritura.
        SYSTEM_INFO      sysInfo;
        GetSystemInfo(&sysInfo);
        m_hHeapEscritura = HeapCreate(0, sysInfo.dwPageSize*2, sysInfo.dwPageSize*8);
        if (m_hHeapEscritura == NULL)
                //AVISAR DEL ERROR Y ABORTAR
                TRACE(_T("ERROR en InicializarElementosEscritura(): HeapEscritura no creada.\n"));

        // Inicializamos la lista enlazada de peticiones de escritura.
        m_pCabezaHeapEscritura = (PPETICION_ESCRITURA)HeapAlloc(m_hHeapEscritura, HEAP_ZERO_MEMORY,
sizeof(PETICION_ESCRITURA));
        if (m_pCabezaHeapEscritura == NULL)
                //AVISAR DEL ERROR Y ABORTAR
                TRACE(_T("ERROR en InicializarElementosEscritura(): CabezaHeapEscritura no
creada.\n"));

        m_pColaHeapEscritura = (PPETICION_ESCRITURA)HeapAlloc(m_hHeapEscritura, HEAP_ZERO_MEMORY,
sizeof(PETICION_ESCRITURA));
        if (m_pColaHeapEscritura == NULL)
                //AVISAR DEL ERROR Y ABORTAR
                TRACE(_T("ERROR en InicializarElementosEscritura(): ColaHeapEscritura no
creada.\n"));

        m_pCabezaHeapEscritura->pNodoAnterior = m_pCabezaHeapEscritura->pNodoSiguiente =
m_pColaHeapEscritura;
        m_pColaHeapEscritura->pNodoAnterior = m_pColaHeapEscritura->pNodoSiguiente =
m_pCabezaHeapEscritura;

        // Creamos un objeto de sincronización de tipo semáforo para indicar que existe una petición
        // de escritura. Este evento se activa cuando se añade un objeto a la lista enlazada de
peticiones,
        // es decir, cada vez que se pretende enviar un comando por el puerto serie.
        m_hSemafPeticionEscritura = CreateSemaphore(NULL, 0, 200, NULL);
        if (m_hSemafPeticionEscritura == NULL)
                //AVISAR DEL ERROR Y ABORTAR
                TRACE(_T("ERROR en InicializarElementosEscritura(): SemafPeticionEscritura no
creado.\n"));

        // Creamos un evento de sincronización para indicar que existe permiso para escribir. Se
activa después
        // de recibir un caracter '.'. Creamos el evento con auto-reset: el SO resetea
automáticamente el evento
        // (lo pone a 0) después de que un único hilo que estaba esperando deja de hacerlo. Si en
este momento
        // hay algo para escribir (el semáforo vale más que cero) se escribe por el puerto serie.
//      m_hEventoPermisoEscritura = CreateEvent(NULL,FALSE,FALSE,NULL);
//      if (m_hEventoPermisoEscritura == NULL)
//              //AVISAR DEL ERROR Y ABORTAR
//              TRACE(_T("ERROR en InicializarElementosEscritura(): EventoPermisoEscritura no
creado.\n"));
}
```

```
////////////////////////////////////////////////////////////////////////////
//
//
////////////////////////////////////////////////////////////////////////////
void CPuertoSerie::FinalizarElementosEscritura()
{
        // Cerramos los manejadores de eventos.
//      BOOL bSinError = CloseHandle(m_hEventoPermisoEscritura);
//      m_hEventoPermisoEscritura = INVALID_HANDLE_VALUE;
//      if (!bSinError)
//              TRACE(_T("ERROR en FinalizarElementosEscritura(): Manejador de
EventoPermisoEscritura no cerrado. GetLastError(): %d.\n"), GetLastError());

        BOOL bSinError = CloseHandle(m_hSemafPeticionEscritura);
        m_hSemafPeticionEscritura = INVALID_HANDLE_VALUE;
        if (!bSinError)
                TRACE(_T("ERROR en FinalizarElementosEscritura(): Manejador de
SemafPeticionEscritura no cerrado. GetLastError(): %d.\n"), GetLastError());

        // Destruimos la HEAP para las peticiones de escritura.
        HeapDestroy(m_hHeapEscritura);

        // Borramos la CRITICAL_SECTION
        DeleteCriticalSection(&m_csHeapEscritura);
}

////////////////////////////////////////////////////////////////////////////
//
//
////////////////////////////////////////////////////////////////////////////
void CPuertoSerie::ProcesarPeticionEscritura()
{
        PPETICION_ESCRITURA pNodo;

        pNodo = m_pCabezaHeapEscritura->pNodoSiguiente;

        // Hay que comprobar que el nodo no sea la cola.
        // Si no es la cola, se escribe su contenido por el puerto serie y se elimina el nodo.
        // Si es la cola no se hace nada.
        if (pNodo != m_pColaHeapEscritura)
        {
                Escribir(pNodo);
                EliminarNodo(pNodo);
        }
        else
                TRACE(_T("ERROR en ProcesarPeticionEscritura(): No hay peticiones.\n"));
}

/*
//NON-OVERLAPPED
DWORD CPuertoSerie::Escribir(PPETICION_ESCRITURA pNodo)
{
        ASSERT(EstaAbierto());

        DWORD dwBytesEscritos = 0;
```

```
    //   Realizamos la escritura.
         if (!WriteFile(m_hComm, pNodo->chBuffer, sizeof(pNodo->chBuffer), &dwBytesEscritos, NULL))
         {
                  //AVISAR DEL ERROR Y ABORTAR
                  TRACE(_T("ERROR en Escribir(): Fallo al llamar a WriteFile. GetLastError(): %d.\n"),
GetLastError());
                  AfxLanzaExcepcionPuertoSerie();
                  return 0;
         }

         return dwBytesEscritos;
}
*/

////////////////////////////////////////////////////////////////////////////
//
//
////////////////////////////////////////////////////////////////////////////
void CPuertoSerie::Escribir(PPETICION_ESCRITURA pNodo)
{
         ASSERT(EstaAbierto());

         OVERLAPPED osEscritor = {0};
         DWORD dwBytesEscritos = 0;

         // Creamos el evento para la estructura OVERLAPPED.
         osEscritor.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
         if (osEscritor.hEvent == NULL)
                  //AVISAR DEL ERROR Y ABORTAR
                  TRACE(_T("ERROR en Escribir(): Fallo al intentar crear el evento para la estructura
OVERLAPPED.\n"));


    //   Realizamos la escritura.
         if (!WriteFile(m_hComm, pNodo->chBuffer, sizeof(pNodo->chBuffer), &dwBytesEscritos,
&osEscritor))
         {
                  if (GetLastError() != ERROR_IO_PENDING)            // Writefile falló, pero no se ha
pausado.
                 {
                           //AVISAR DEL ERROR Y ABORTAR
                           TRACE(_T("ERROR en Escribir(): Fallo al llamar a WriteFile.\n"));
                           AfxLanzaExcepcionPuertoSerie();
                  }

                  else     // Writefile se ha pausado
                  {
                           if (!GetOverlappedResult(m_hComm, &osEscritor, &dwBytesEscritos, TRUE))
                           {
                                    //AVISAR DEL ERROR Y ABORTAR
                                    TRACE(_T("ERROR en Escribir(): Fallo en la espera de WriteFile().
GetLastError(): %d.\n"), GetLastError());
                           }
                           //else
```

```
                        //      TRACE(_T("Escribir(): Tras espera hemos escrito:
%d.\n"),dwBytesEscritos);

/*                      // Un código equivalente es:
                        DWORD dwRes = WaitForMultipleObjects(2, hEventos, FALSE, INFINITE);
            switch(dwRes)
            {
                                case WAIT_OBJECT_0:
// Evento de estructura OVERLAPPED marcado.
                                        //SetLastError(ERROR_SUCCESS);
                                        if (!GetOverlappedResult(m_hComm, &osEscritor,
&dwBytesEscritos, FALSE))
                                        {
                                                //AVISAR DEL ERROR Y ABORTAR
                                                TRACE(_T("ERROR en Escribir(): Fallo en la espera
de WriteFile(). GetLastError(): %d.\n"), GetLastError());
                                        }
                                        //if (dwWritten != dwToWrite)
                                        //{
                                        //      if ((GetLastError() == ERROR_SUCCESS) &&
SHOWTIMEOUTS(TTYInfo))
                                        //              UpdateStatus("Write timed out.
(overlapped)\r\n");
                                        //      else
                                        //              ErrorReporter("Error writing data to port
(overlapped)");
                                        //}
                    break;

                                case WAIT_OBJECT_0 + 1:          // EventoTerminaHilos marcado.
                                        break;

                case WAIT_TIMEOUT:
                    TRACE(_T("Wait Timeout en la espera de WriteFile().\n"));
                    break;

                case WAIT_FAILED:
                default:
                                        //AVISAR DEL ERROR Y ABORTAR
                                        TRACE(_T("ERROR en Escribir(): Fallo en la espera de
WriteFile.\n"));
                                        AfxLanzaExcepcionPuertoSerie();
                                        break;
            }
*/
        }

    }
    else         // WriteFile terminó inmediantamente.
        {
        if (dwBytesEscritos != sizeof(pNodo->chBuffer))
            TRACE(_T("Wait Timeout en WriteFile() (cuando termina inmediatamente.\n"));
    }

        BOOL bSinError = CloseHandle(osEscritor.hEvent);
```

```
        osEscritor.hEvent = INVALID_HANDLE_VALUE;
        if (!bSinError)
                TRACE(_T("ERROR en Escribir(): Fallo al intentar cerrar manajedor del evento
OVERLAPPED. GetLastError(): %d.\n"), GetLastError());
}

//////////////////////////////////////////////////////////////////////////
//
//
//////////////////////////////////////////////////////////////////////////
void CPuertoSerie::ObtenerEstadoPuertoSerie(DCB& dcb)
{
        ASSERT(EstaAbierto());

        if (!GetCommState(m_hComm, &dcb))
        {
                TRACE(_T("ERROR en ObtenerEstadoPuertoSerie(): Fallo al llamar a
GetCommState().\n"));
                AfxLanzaExcepcionPuertoSerie();
        }
}

//////////////////////////////////////////////////////////////////////////
//
//
//////////////////////////////////////////////////////////////////////////
void CPuertoSerie::FijarEstadoPuertoSerie(DCB& dcb)
{
        ASSERT(EstaAbierto());

        if (!SetCommState(m_hComm, &dcb))
        {
                TRACE(_T("ERROR en FijarEstadoPuertoSerie(): Fallo al llamar a SetCommState().\n"));
                AfxLanzaExcepcionPuertoSerie();
        }
}

//////////////////////////////////////////////////////////////////////////
//
//
//////////////////////////////////////////////////////////////////////////
void CPuertoSerie::ObtenerMascaraEventosPuertoSerie(DWORD& dwMascara)
{
        ASSERT(EstaAbierto());

        if (!GetCommMask(m_hComm, &dwMascara))
        {
                TRACE(_T("ERROR en ObtenerMascaraEventosPuertoSerie(): Fallo al llamar a
GetCommMask().\n"));
                AfxLanzaExcepcionPuertoSerie();
        }
}

//////////////////////////////////////////////////////////////////////////
//
```

```
//
//////////////////////////////////////////////////////////////////////
void CPuertoSerie::FijarMascaraEventosPuertoSerie(DWORD dwMascara)
{
        ASSERT(EstaAbierto());

        if (!SetCommMask(m_hComm, dwMascara))
        {
            TRACE(_T("ERROR en FijarMascaraEventosPuertoSerie(): Fallo al llamar a
SetCommMask().\n"));
                AfxLanzaExcepcionPuertoSerie();
        }
}

//////////////////////////////////////////////////////////////////////
//
//
//////////////////////////////////////////////////////////////////////
void CPuertoSerie::ObtenerTimeouts(COMMTIMEOUTS& timeouts)
{
        ASSERT(EstaAbierto());

        if (!GetCommTimeouts(m_hComm, &timeouts))
        {
                TRACE(_T("ERROR en ObtenerTimeouts(): Fallo al llamar a GetCommTimeouts().\n"));
                AfxLanzaExcepcionPuertoSerie();
        }
}

//////////////////////////////////////////////////////////////////////
//
//
//////////////////////////////////////////////////////////////////////
void CPuertoSerie::FijarTimeouts(COMMTIMEOUTS& timeouts)
{
        ASSERT(EstaAbierto());

        if (!SetCommTimeouts(m_hComm, &timeouts))
        {
                TRACE(_T("ERROR en FijarTimeouts(): Fallo al llamar a SetCommTimeouts().\n"));
                AfxLanzaExcepcionPuertoSerie();
        }
}

//////////////////////////////////////////////////////////////////////
//
//
//////////////////////////////////////////////////////////////////////
void CPuertoSerie::FijarNoTimeouts()
{
        COMMTIMEOUTS Timeouts;
        ZeroMemory(&Timeouts, sizeof(COMMTIMEOUTS));
        //Timeouts.ReadIntervalTimeout = MAXDWORD;
        FijarTimeouts(Timeouts);
}
```

```
////////////////////////////////////////////////////////////////////////////
//
//
////////////////////////////////////////////////////////////////////////////
void CPuertoSerie::LimpiarBuffers()
{
        ASSERT(EstaAbierto());

        DWORD dwFlags = PURGE_TXABORT | PURGE_RXABORT | PURGE_TXCLEAR | PURGE_RXCLEAR;

        // PURGE_TXABORT: Terminates all outstanding overlapped write operations and returns
immediately, even if the write operations have not been completed.
        // PURGE_RXABORT: Terminates all outstanding overlapped read operations and returns
immediately, even if the read operations have not been completed.
        // PURGE_TXCLEAR: Clears the output buffer (if the device driver has one).
        // PURGE_RXCLEAR: Clears the input buffer (if the device driver has one).

        if (!PurgeComm(m_hComm, dwFlags))
        {
                TRACE(_T("ERROR en LimpiarBuffers(): Fallo al llamar a PurgeComm().\n"));
            AfxLanzaExcepcionPuertoSerie();
        }

        // If a thread uses PurgeComm to flush an output buffer, the deleted characters are not
transmitted. To empty the output buffer while ensuring that the contents are transmitted, call the
FlushFileBuffers function (a synchronous operation). Note, however, that FlushFileBuffers is subject
to flow control but not to write time-outs, and it will not return until all pending write operations
have been transmitted
}

/*----------------------------------------------------------------------------

FUNCTION: WriterAddNewNode(DWORD, DWORD, char, char *, HANDLE, HWND)

PURPOSE: Adds a new write request packet

PARAMETERS:
    dwRequestType - write request packet request type
    dwSize        - size of write request
    ch            - character to write
    lpBuf         - address of buffer to write
    hHeap         - heap handle of data buffer
    hProgress     - hwnd of transfer progress bar

RETURN:
    TRUE if node is added to linked list
    FALSE if node can't be allocated.

COMMENTS: Allocates a new packet and fills it based on the
          parameters passed in.

HISTORY:   Date:      Author:     Comment:
           10/27/95   AllenD      Wrote it
```

```
------------------------------------------------------------------------------*/

//////////////////////////////////////////////////////////////////////////////
//
//
//////////////////////////////////////////////////////////////////////////////
void CPuertoSerie::HacerPeticionEscritura(LPCSTR lpszBuffer)
{
        // Reservamos memoria para el nuevo nodo.
        PPETICION_ESCRITURA pNuevoNodo = (PPETICION_ESCRITURA)HeapAlloc(m_hHeapEscritura,
HEAP_ZERO_MEMORY, sizeof(PETICION_ESCRITURA));
        if (pNuevoNodo == NULL)
                //AVISAR DEL ERROR Y ABORTAR
                TRACE(_T("ERROR en HacerPeticionEscritura(): NuevoNodo no creado.\n"));

        // Rellenamos los campos de la estructura PETICION_ESCRITURA.
//      pNuevoNodo->lpBuffer = lpszBuffer;
        strcpy(pNuevoNodo->chBuffer, lpszBuffer);

        // Añadimos el nuevo nodo a la lista enlazada de peticiones de escritura.
        PonerNodo(pNuevoNodo);

        // Notificamos al hilo escritor que se ha añadido una petición de escritura.
        //if (!SetEvent(m_hEventoPeticionEscritura))
        if (!ReleaseSemaphore(m_hSemafPeticionEscritura, 1, NULL))
                //AVISAR DEL ERROR Y ABORTAR
                TRACE(_T("ERROR en HacerPeticionEscritura(): Fallo al notificar
EventoPeticionEscritura.\n"));
        else
                TRACE(_T("HacerPeticionEscritura(): EventoPeticionEscritura activado.\n"));
}

/*------------------------------------------------------------------------------

FUNCTION: AddToLinkedList(PWRITEREQUEST)

PURPOSE: Adds a node to the write request linked list

PARAMETERS:
    pNode - pointer to write request packet to add to linked list

HISTORY:   Date:      Author:     Comment:
           10/27/95   AllenD      Wrote it

------------------------------------------------------------------------------*/

//////////////////////////////////////////////////////////////////////////////
//
//
//////////////////////////////////////////////////////////////////////////////
void CPuertoSerie::PonerNodo(PPETICION_ESCRITURA pNodo)
{
        PPETICION_ESCRITURA pAntiguoUltimoNodo;

        EnterCriticalSection(&m_csHeapEscritura);
```

```
        pAntiguoUltimoNodo = m_pColaHeapEscritura->pNodoAnterior;

        pNodo->pNodoSiguiente = m_pColaHeapEscritura;
        pNodo->pNodoAnterior = pAntiguoUltimoNodo;

        pAntiguoUltimoNodo->pNodoSiguiente = pNodo;
        m_pColaHeapEscritura->pNodoAnterior = pNodo;

        LeaveCriticalSection(&m_csHeapEscritura);
}

/*-------------------------------------------------------------------------

FUNCTION: RemoveFromLinkedList(PWRITEREQUEST)

PURPOSE: Deallocates the head node and makes the passed in node
         the new head node.
         Sets the head node point to node just after the passed in node.
         Returns the node pointed to by the head node.

PARAMETERS:
    pNode - pointer to node to make the new head

RETURN:
    Pointer to next node.  This will be NULL if there are no
    more nodes in the list.

HISTORY:   Date:      Author:     Comment:
           10/27/95   AllenD      Wrote it

-------------------------------------------------------------------------*/
//////////////////////////////////////////////////////////////////////////
//
//
//////////////////////////////////////////////////////////////////////////
void CPuertoSerie::EliminarNodo(PPETICION_ESCRITURA pNodoAEliminar)
{
    PPETICION_ESCRITURA pNodoProximo;
        PPETICION_ESCRITURA pNodoPrevio;
    BOOL fRes;

    EnterCriticalSection(&m_csHeapEscritura);

    pNodoProximo = pNodoAEliminar->pNodoSiguiente;
    pNodoPrevio = pNodoAEliminar->pNodoAnterior;

    fRes = HeapFree(m_hHeapEscritura, 0, pNodoAEliminar);

    pNodoPrevio->pNodoSiguiente = pNodoProximo;
    pNodoProximo->pNodoAnterior = pNodoPrevio;

    LeaveCriticalSection(&m_csHeapEscritura);

    if (!fRes)
```

```
        //AVISAR DEL ERROR Y ABORTAR
                TRACE(_T("ERROR en EliminarNodo(): Fallo al llamar a HeapFree.\n"));

    //return pNodoProximo;     // return the freed node's pNext (maybe the tail)
}

/*
void CPuertoSerie::GetOverlappedResult(OVERLAPPED& overlapped, DWORD& dwBytesTransferred, BOOL bWait)
{
  ASSERT(EstaAbierto());
 // ASSERT(m_bOverlapped);

  if (!::GetOverlappedResult(m_hComm, &overlapped, &dwBytesTransferred, bWait))
  {
    if (GetLastError() != ERROR_IO_PENDING)
    {
      TRACE(_T("Failed in call to GetOverlappedResult\n"));
      AfxLanzaExcepcionPuertoSerie();
    }
  }
}
*/

void CPuertoSerie::_OnCompletion(DWORD dwErrorCode, DWORD dwCount, LPOVERLAPPED lpOverlapped)
{
  //Validate our parameters
  ASSERT(lpOverlapped);

  //Convert back to the C++ world
  CPuertoSerie* pSerialPort = (CPuertoSerie*) lpOverlapped->hEvent;
  ASSERT(pSerialPort);
  ASSERT(pSerialPort->IsKindOf(RUNTIME_CLASS(CPuertoSerie)));

  //Call the C++ function
  pSerialPort->OnCompletion(dwErrorCode, dwCount, lpOverlapped);
}

void CPuertoSerie::OnCompletion(DWORD /*dwErrorCode*/, DWORD /*dwCount*/, LPOVERLAPPED lpOverlapped)
{
  //Just free up the memory which was previously allocated for the OVERLAPPED structure
  delete lpOverlapped;

  //Your derived classes can do something useful in OnCompletion, but don't forget to
  //call CSerialPort::OnCompletion to ensure the memory is freed up
}

void CPuertoSerie::CancelIo()
{
        ASSERT(EstaAbierto());

        if (_SerialPortData.m_lpfnCancelIo == NULL)
        {
                TRACE(_T("CancelIo function is not supported on this OS. You need to be running at
least NT 4 or Win 98 to use this function\n"));
                AfxLanzaExcepcionPuertoSerie(ERROR_CALL_NOT_IMPLEMENTED);
```

```
        }

        if (!::_SerialPortData.m_lpfnCancelIo(m_hComm))
        {
            TRACE(_T("Failed in call to CancelIO\n"));
            AfxLanzaExcepcionPuertoSerie();
        }
}

/*
DWORD CSerialPort::Read(void* lpBuf, DWORD dwCount)
{
  ASSERT(EstaAbierto());
  ASSERT(!m_bOverlapped);

  DWORD dwBytesRead = 0;
  if (!ReadFile(m_hComm, lpBuf, dwCount, &dwBytesRead, NULL))
  {
    TRACE(_T("Failed in call to ReadFile\n"));
    AfxThrowSerialException();
  }

  return dwBytesRead;
}

BOOL CPuertoSerie::Leer(void* lpBuf, DWORD dwCount, OVERLAPPED& overlapped, DWORD* pBytesRead)
{
  ASSERT(EstaAbierto());
 // ASSERT(m_bOverlapped);

  DWORD dwBytesRead = 0;
  BOOL bSinError = ReadFile(m_hComm, lpBuf, dwCount, &dwBytesRead, &overlapped);
  if (!bSinError)
  {
    if (GetLastError() != ERROR_IO_PENDING)
    {
      TRACE(_T("Failed in call to ReadFile\n"));
      AfxLanzaExcepcionPuertoSerie();
    }
  }
  else
  {
    if (pBytesRead)
      *pBytesRead = dwBytesRead;
  }
  return bSinError;
}
*/

void CPuertoSerie::CapturarPuntoInicial(Joint *m_pAngulos)
{
        CString strOrden;

        // Enviamos un comando HERE para que el controlador nos devuelva la posición del extremo.
        CCorte3DApp* pAplicacion = (CCorte3DApp*)AfxGetApp();
```

```
        strOrden.Format(_T("POINT #P"));
        pAplicacion->EnviarOrden((LPCTSTR)strOrden);

        m_pPuntoInicial = m_pAngulos;
        m_fCapturarPuntoInicial = TRUE;

}
```

B.1.7. CPuntoTrayectoria.

B.1.7.1. PuntoTrayectoria.h

```
////////////////////////////////////////////////////////////////////////////
//                                                                        //
//      Módulo : PuntoTrayectoria.h                                       //
//                      Definición de la clase CPuntoTrayectoria.         //
//                                                                        //
//      Propósito:      Esta clase contiene las estructuras y operaciones //
//                      necesarias para trabajar con los puntos de las    //
//                      trayectorias: permite almacenar los puntos en     //
//                      diversos formatos (XYZvector, XYZeulerZYZ, theta[i]//
//                      {i=1..6},matriz de transf. homogénea), realizar   //
//                      conversiones entre       ellos y ejecutar las operaciones //
//                      necesarias para cada formato.                     //
//                                                                        //
//      Creado: Manuel Gómez Langley / 2003-2004                          //
//      Modificado:     Alejandro Guija Rodríguez 2006-2007               //
//                                                                        //
////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_PUNTOTRAYECTORIA_H__A2D3E6B5_D0C6_47DD_B3F7_C3A4D9C4A6D8__INCLUDED_)
#define AFX_PUNTOTRAYECTORIA_H__A2D3E6B5_D0C6_47DD_B3F7_C3A4D9C4A6D8__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

// Estructura que se utiliza para almacenar las coordenadas cartesianas de un
// punto.
struct Vector
{
        double x;
        double y;
        double z;
};

//struct Joint
//{
//      double j1;
//      double j2;
//      double j3;
//      double j4;
//      double j5;
```

```
//      double j6;
//};

//////////////////////////// Clases //////////////////////////////////////////
/*

//AÑADIR CLASES VECTOR Y MATRIZ, en este mismo fichero valdría.

class CVector
{
//////////////////////////////////////////////////////////////////////////////
// Enumeraciones
//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////
// Variables miembro
//////////////////////////////////////////////////////////////////////////////
protected:
        double X;
        double Y;
        double Z;

//////////////////////////////////////////////////////////////////////////////
// Constructor/Destructor
//////////////////////////////////////////////////////////////////////////////

public:
        CVector();
        virtual ~CVector();

//////////////////////////////////////////////////////////////////////////////
// Métodos miembro
//////////////////////////////////////////////////////////////////////////////

        double Modulo();
};

class CMatriz
{

//////////////////////////////////////////////////////////////////////////////
// Enumeraciones
//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////
// Variables miembro
//////////////////////////////////////////////////////////////////////////////
protected:
        // Doble array que almacena la localización y orientación del extremo
        // del robot en forma de matriz de transformación homogénea (4x4).
        // Se utiliza para realizar la traslación de cada punto de la
        // trayectoria virtual respecto del punto inicial donde se encuentra la
        // herramienta en la realidad para obtener la trayectoria real.
        double m_dMatriz[4][4];
```

```
////////////////////////////////////////////////////////////////////////////
Constructor/Destructor
////////////////////////////////////////////////////////////////////////////

public:
        CMatriz();
        virtual ~CMatriz();

////////////////////////////////////////////////////////////////////////////
// Métodos miembro
////////////////////////////////////////////////////////////////////////////

        void Invertir();
        CMatriz& Premultiplicar(const CMatriz& matriz);
        SetMatriz(CVector* puntos);
};
*/

class CPuntoTrayectoria
{
////////////////////////////////////////////////////////////////////////////
// Enumeraciones
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
// Variables miembro
////////////////////////////////////////////////////////////////////////////

protected:

        // Estructura que almacena un punto en un formato X,Y,Z,vector, que es
        // el que contienen los ficheros APT.
        // En este formato, los 3 primeros valores son las coordenadas del
        // extremo de la garra en el espacio, mientras que los otros 3 valores
        // nos informan de la orientación de la herramienta a través de un
        // vector unitario.
        struct coordXYZvector
        {
                double X;
                double Y;
                double Z;
                double i;
                double j;
                double k;
        } m_sCoordXYZvector;

        // Estructura que almacena un punto en un formato X,Y,Z,euler-ZYZ, que
        // es el que utiliza V_TRAJSIG.
        // En este formato, los 3 primeros valores son las coordenadas del
        // extremo de la garra en el espacio, mientras que los otros 3 valores
        // nos informan de la orientación de la garra a través de un sistema de
        // referencia.
        // Los ángulos ZYZ de Euler definen la orientación mediante 3 giros
        // consecutivos respecto a los ejes ZYZ de un sistema de referencia {B}
        // solidario al cuerpo. Se comienza con {B} coincidente con el sistema
```

```
        // de referencia {A}. Sucesivamente se realizan 3 giros:
        //      -       Primero se produce una rotación de {B} un ángulo ALFA
        //              alrededor del eje Zb, que en este caso es coincidente con
        //              el eje Z origen (Za).
        //      -       Después se produce una rotación de un ángulo BETA alrededor
        //              del nuevo eje Yb.
        //      -       Finalmente se produce una rotación de un ángulo GAMMA
        //              alrededor del nuevo eje Zb.
        struct coordXYZeulerZYZ
        {
                double X;
                double Y;
                double Z;
                double eulerZalfa;
                double eulerYbeta;
                double eulerZgamma;
        } m_sCoordXYZeulerZYZ;

        // Variable estática que se utiliza para almacenar el valor del ángulo
        // alfa de EulerZYZ del punto anterior de la trayectoria.
        static double m_dEulerZalfaPuntoAnterior;

        // Estructura estática que se utiliza para almacenar el valor de los
        // parámetros de Denavit-Hartemberg del robot RX-90.
        struct parametrosDHRobot
        {
                double a2;
                double a3;
                double d3;
                double d4;
                double d6;
        } static m_sParam;

        // Constante estática que almacena el valor de PI.
//      const double PI = 3.14159265358979310862446 8;
        // Constante estática que almacena el valor de la precision para 0.
//      const double CERO = 0.00001;

////////////////////////////////////////////////////////////////////////////
// Constructor/Destructor
////////////////////////////////////////////////////////////////////////////

public:
        CPuntoTrayectoria();
        virtual ~CPuntoTrayectoria();

////////////////////////////////////////////////////////////////////////////
// Métodos miembro
////////////////////////////////////////////////////////////////////////////

public:
        double LeerCoordXYZeulerZYZ(int j);
        void CrearMatrizDesplazada(Vector* vPuntos);

        // Doble array que almacena la localización y orientación del extremo
```

```
        // del robot en forma de matriz de transformación homogénea (4x4).
        // Se utiliza para realizar la traslación de cada punto de la
        // trayectoria virtual respecto del punto inicial donde se encuentra la
        // herramienta en la realidad para obtener la trayectoria real.
        double m_dMatriz[4][4];

        void XYZeulerZYZ2Matriz();
//      void XYZvector2Matriz(double* coordAPT);
// Seria ideal, pero no lo puedo hacer directamente, sino q lo tng q hace en 2 pasos.
        void XYZvector2XYZeulerZYZ();                                   // Es la
función zyz() de M.Vargas.
        void XYZvector2Matriz();
        void Matriz2XYZeulerZYZ();

        void Componer(const CPuntoTrayectoria& Inicio);
// Definimos asi la función xq es mas efectivo pasar objetos por referencia
// (no se llaman constructores ni copy constructors... Definimos el parámetro
// const para que no se modifique dentro de la función. Así tenemos un objeto
// localmente constante dentro de la función.

        void ResolverModeloCinematicoInverso(int nBrazo, int nCodo, int nMuneca);
//      void ResolverModeloCinematicoDirecto();
        BOOL EsAlcanzable();

        void SetXYZvector(double* dCoordenadas);
        void SetXYZeulerZYZ(double* dCoordenadas);
        void GetXYZeulerZYZ(double* dCoordenadas);

        void CrearMatriz(Vector* vPuntos);
        //void CrearMatriz(Vector vPunto0, VectorvPunto1, Vector vPunto2);
        void InvertirMatriz();

//      void GetXYZeulerZYZ(double* &datos);
//      void SetXYZeulerZYZ(double* datos);                             // El
programa sería más eficiente si se inicializaran las variables en el cosntructor.
//      void SetXYZvector(double* dCoordenadas);
// Como ambos constructores necesitarian un double[6], habria que añadir un
// nuevo paramétro para poder discrminar qué variables habría que inicializar.
// Por ejemplo, un tipo enumerado creado a tal efecto.

// Como definimos un array de CPuntoTrayectoria con new, no podemos usar el
// constructor para inicializar, asi q usamos estas funciones para asignar los
// valores iniciales.
// Estructura que almacena la localización de un punto en coordenadas
// articulares del robot.
// Se utiliza al resolver el modelo cinemático inverso para comprobar si
// el punto es alcanzable.
        struct coordTheta
        {
                double theta[6];
        } m_sCoordTheta;
};

#endif // !defined(AFX_PUNTOTRAYECTORIA_H__A2D3E6B5_D0C6_47DD_B3F7_C3A4D9C4A6D8__INCLUDED_)
```

B.1.7.2. PuntoTrayectoria.cpp

```
//////////////////////////////////////////////////////////////////////
//                                                                  //
//      Módulo : PuntoTrayectoria.cpp                               //
//               Implementación de la clase CPuntoTrayectoria.      //
//                                                                  //
//      Creado: Manuel Gómez Langley / 2003-2004                    //
//      Modificado:      Alejandro Guija Rodríguez 2006-2007        //
//                                                                  //
//////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////
//      Incluimos el fichero de cabecera estándar y el fichero de   //
//      cabecera necesario para realizar operaciones matemáticas.   //
//////////////////////////////////////////////////////////////////////
#include "stdafx.h"
#include <math.h>
#include <fstream.h>
#include <iostream.h>

//////////////////////////////////////////////////////////////////////////////
// Incluimos los ficheros de cabecera del módulo y del programa principal
//////////////////////////////////////////////////////////////////////////////

#include "PuntoTrayectoria.h"
#include "Corte3D.h"

//////////////////////////////////////////////////////////////////////////////
// Habilitamos el 'debug memory manager'
//////////////////////////////////////////////////////////////////////////////

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////////////
// Definición e inicialización de miembros estáticos
//////////////////////////////////////////////////////////////////////////////
double CPuntoTrayectoria::m_dEulerZalfaPuntoAnterior = 0;
struct CPuntoTrayectoria::parametrosDHRobot CPuntoTrayectoria::m_sParam = {450, 0, 0, 450, 85};

//////////////////////////////////////////////////////////////////////////////
// Código
//////////////////////////////////////////////////////////////////////////////

CPuntoTrayectoria::CPuntoTrayectoria()//:
//      m_dMatriz[3][0](0),
//      m_dMatriz[3][1](0),
//      m_dMatriz[3][2](0),
//      m_dMatriz[3][3](1)
        // No puedo inicializarlos asi, no sé por qué!!
```

```
{
	m_dMatriz[0][0] = 0;
	m_dMatriz[0][1] = 0;
	m_dMatriz[0][2] = 0;
	m_dMatriz[0][3] = 0;
	m_dMatriz[1][0] = 0;
	m_dMatriz[1][1] = 0;
	m_dMatriz[1][2] = 0;
	m_dMatriz[1][3] = 0;
	m_dMatriz[2][0] = 0;
	m_dMatriz[2][1] = 0;
	m_dMatriz[2][2] = 0;
	m_dMatriz[2][3] = 0;
	m_dMatriz[3][0] = 0;
	m_dMatriz[3][1] = 0;
	m_dMatriz[3][2] = 0;
	m_dMatriz[3][3] = 1;
}


CPuntoTrayectoria::~CPuntoTrayectoria()
{
}

//////////////////////////////////////////////////////////////////////
//                                                                  //
//      Método miembro que sirve para calcular la matriz de         //
//      transformación homogénea de un punto a partir de las        //
//      coordenadas XYZeulerZYZ.                                    //
//                                                                  //
//////////////////////////////////////////////////////////////////////
void CPuntoTrayectoria::XYZeulerZYZ2Matriz()
{
	const double PI = 3.14159265358979310862446 8;

	// Calculamos el valor de las funciones trigonométricas una sola vez
	// para ahorrar tiempo de ejecución.
	double dSenoAlfa = sin(m_sCoordXYZeulerZYZ.eulerZalfa*PI/180);
	double dCosenoAlfa = cos(m_sCoordXYZeulerZYZ.eulerZalfa*PI/180);
	double dSenoBeta = sin(m_sCoordXYZeulerZYZ.eulerYbeta*PI/180);
	double dCosenoBeta = cos(m_sCoordXYZeulerZYZ.eulerYbeta*PI/180);
	double dSenoGamma = sin(m_sCoordXYZeulerZYZ.eulerZgamma*PI/180);
	double dCosenoGamma = cos(m_sCoordXYZeulerZYZ.eulerZgamma*PI/180);

	// Calculamos el valor de los elementos de la matriz de rotación.
	m_dMatriz[0][0] = dCosenoAlfa*dCosenoBeta*dCosenoGamma - dSenoAlfa*dSenoGamma;
	m_dMatriz[0][1] = -(dCosenoAlfa*dCosenoBeta*dSenoGamma) - dSenoAlfa*dCosenoGamma;
	m_dMatriz[0][2] = dCosenoAlfa*dSenoBeta;

	m_dMatriz[1][0] = dSenoAlfa*dCosenoBeta*dCosenoGamma + dCosenoAlfa*dSenoGamma;
	m_dMatriz[1][1] = -(dSenoAlfa*dCosenoBeta*dSenoGamma) + dCosenoAlfa*dCosenoGamma;
	m_dMatriz[1][2] = dSenoAlfa*dSenoBeta;

	m_dMatriz[2][0] = -(dSenoBeta*dCosenoGamma);
	m_dMatriz[2][1] = dSenoBeta*dSenoGamma;
```

```
        m_dMatriz[2][2] = dCosenoBeta;

        // Fijamos el valor de los elementos del vector de traslación.
        m_dMatriz[0][3] = m_sCoordXYZeulerZYZ.X;
        m_dMatriz[1][3] = m_sCoordXYZeulerZYZ.Y;
        m_dMatriz[2][3] = m_sCoordXYZeulerZYZ.Z;

//      TRACE(_T("Coordenadas XYZeulerZYZ: %f %f %f %f %f %f\n"),
//                      m_sCoordXYZeulerZYZ.X, m_sCoordXYZeulerZYZ.Y, m_sCoordXYZeulerZYZ.Z,
//                      m_sCoordXYZeulerZYZ.eulerZalfa, m_sCoordXYZeulerZYZ.eulerYbeta,
m_sCoordXYZeulerZYZ.eulerZgamma);

        TRACE(_T("Matriz de transformación homogénea:\n%f %f %f %f\n%f %f %f %f\n%f %f %f %f\n%f %f
%f %f\n"),
                                m_dMatriz[0][0],m_dMatriz[0][1],m_dMatriz[0][2],m_dMatriz[0][3],
                                m_dMatriz[1][0],m_dMatriz[1][1],m_dMatriz[1][2],m_dMatriz[1][3],
                                m_dMatriz[2][0],m_dMatriz[2][1],m_dMatriz[2][2],m_dMatriz[2][3],
                                m_dMatriz[3][0],m_dMatriz[3][1],m_dMatriz[3][2],m_dMatriz[3][3]);
}

////////////////////////////////////////////////////////////////////////////
//
// Método miembro que sirve para calcular las coordenadas XYZeulerZYZ de un
// punto a partir de las coordenadas XYZvector.
// Empleamos el método seguido en la función zyz() de M.Vargas.
//
////////////////////////////////////////////////////////////////////////////
void CPuntoTrayectoria::XYZvector2XYZeulerZYZ()
{
        double dAux, dSenoAux, dCosenoAux, dAux2, dDifAux, dDifAux2;
        const double PI = 3.14159265358979310862 4468;

        // Primero almacenamos el vector de translación.
        m_sCoordXYZeulerZYZ.X = m_sCoordXYZvector.X;
        m_sCoordXYZeulerZYZ.Y = m_sCoordXYZvector.Y;
        m_sCoordXYZeulerZYZ.Z = m_sCoordXYZvector.Z;

        // Ahora calculamos el valor de los ángulos que nos dan la orientación de la herramienta.
        dAux = atan2(m_sCoordXYZvector.j,m_sCoordXYZvector.i);
        dSenoAux = sin(dAux);
        dCosenoAux = cos(dAux);

        if (fabs(dSenoAux) > 0.5)
                dAux2 = m_sCoordXYZvector.j / dSenoAux;
        else
                dAux2 = m_sCoordXYZvector.i / dCosenoAux;

        m_sCoordXYZeulerZYZ.eulerZalfa = dAux * 180/PI;
        m_sCoordXYZeulerZYZ.eulerYbeta = (atan2(dAux2,m_sCoordXYZvector.k)) * 180/PI;
        m_sCoordXYZeulerZYZ.eulerZgamma = 0;

        dAux = m_sCoordXYZeulerZYZ.eulerZalfa;
        dAux2 = m_sCoordXYZeulerZYZ.eulerZalfa + 180;

        if (dAux2 > 180)
```

```
                dAux2 -= 360;

        dDifAux = dAux - m_dEulerZalfaPuntoAnterior;
        dDifAux2 = dAux2 - m_dEulerZalfaPuntoAnterior;

        if (dDifAux < -180)
                dDifAux += 360;
        if (dDifAux > 180)
                dDifAux -= 360;

        if (dDifAux2 < -180)
                dDifAux2 += 360;
        if (dDifAux2 > 180)
                dDifAux2 -= 360;

        if (fabs(dDifAux2) < fabs(dDifAux))
        {
                m_sCoordXYZeulerZYZ.eulerZalfa = dAux2;
                m_sCoordXYZeulerZYZ.eulerYbeta = -m_sCoordXYZeulerZYZ.eulerYbeta;
        }

        // Almacenamos el valor del ángulo alfa para ser utilizado por el punto siguiente.
        m_dEulerZalfaPuntoAnterior = m_sCoordXYZeulerZYZ.eulerZalfa;


//      TRACE(_T("Coordenadas XYZvector: %f %f %f %f %f %f\n"),
//                      m_sCoordXYZvector.X, m_sCoordXYZvector.Y, m_sCoordXYZvector.Z,
//                      m_sCoordXYZvector.i, m_sCoordXYZvector.j, m_sCoordXYZvector.k);

        TRACE(_T("Coordenadas XYZeulerZYZ: %f %f %f %f %f %f\n"),
                      m_sCoordXYZeulerZYZ.X, m_sCoordXYZeulerZYZ.Y, m_sCoordXYZeulerZYZ.Z,
                      m_sCoordXYZeulerZYZ.eulerZalfa, m_sCoordXYZeulerZYZ.eulerYbeta,
m_sCoordXYZeulerZYZ.eulerZgamma);
}

////////////////////////////////////////////////////////////////////////////
//
// Método miembro que sirve para calcular la matriz de transformación
// homogénea de un punto a partir de las coordenadas XYZvector.
//
////////////////////////////////////////////////////////////////////////////
void CPuntoTrayectoria::XYZvector2Matriz()
{
        XYZvector2XYZeulerZYZ();
        XYZeulerZYZ2Matriz();
}

////////////////////////////////////////////////////////////////////////////
//
// Método miembro que sirve para calcular las coordenadas XYZeulerZYZ de un
// punto a partir de su matriz de transformación homogénea.
//
// Utilizamos el método seguido en HEMERO (distinto al del libro de Robótica).
//      function euler = tr2eul(m)
//              euler = zeros(1,3);
```

```
//              if abs(m(2,3)) > eps & abs(m(1,3)) > eps,
//                      euler(1) = atan2(m(2,3), m(1,3));
//                      sp = sin(euler(1));
//                      cp = cos(euler(1));
//                      euler(2)=atan2(cp*m(1,3) + sp*m(2,3), m(3,3));
//                      euler(3)=atan2(-sp*m(1,1)+cp*m(2,1),-sp*m(1,2)+cp*m(2,2));
//              else,
//                      euler(1) = 0;
//                      euler(2) = atan2(m(1,3), m(3,3));
//                      euler(3) = atan2(m(2,1), m(2,2));
//      end
//
////////////////////////////////////////////////////////////////////////////
void CPuntoTrayectoria::Matriz2XYZeulerZYZ()
{
        // Definimos las constante que almacenan los valores de la precision para 0 y de pi.
        const double CERO = 0.0000001;
        const double PI = 3.14159265358979310862446 8;
        // El valor de diferencia se utiliza para salvar la prolongación del extremo en el
        // eje z del mismo debido a la utilización de la fresa.
        int diferencia=275;

        // Primero almacenamos el valor del vector de translación.
        m_sCoordXYZeulerZYZ.X = m_dMatriz[0][3]-diferencia*m_dMatriz[0][2];
        m_sCoordXYZeulerZYZ.Y = m_dMatriz[1][3]-diferencia*m_dMatriz[1][2];
        m_sCoordXYZeulerZYZ.Z = m_dMatriz[2][3]-diferencia*m_dMatriz[2][2];

        // Ahora calculamos los ángulos ZYZ de euler.
        if ((fabs(m_dMatriz[1][2]) > CERO) && (fabs(m_dMatriz[0][2]) > CERO))
        {
                m_sCoordXYZeulerZYZ.eulerZalfa = atan2(m_dMatriz[1][2], m_dMatriz[0][2]);
                double dSenoAlfa = sin(m_sCoordXYZeulerZYZ.eulerZalfa);
                double dCosenoAlfa = cos(m_sCoordXYZeulerZYZ.eulerZalfa);

                m_sCoordXYZeulerZYZ.eulerYbeta = atan2(dCosenoAlfa*m_dMatriz[0][2] +
dSenoAlfa*m_dMatriz[1][2], m_dMatriz[2][2]);
                m_sCoordXYZeulerZYZ.eulerZgamma = atan2(-dSenoAlfa*m_dMatriz[0][0] +
dCosenoAlfa*m_dMatriz[1][0], -dSenoAlfa*m_dMatriz[0][1] + dCosenoAlfa*m_dMatriz[1][1]);
        }
        else
        {
                m_sCoordXYZeulerZYZ.eulerZalfa = 0;
                m_sCoordXYZeulerZYZ.eulerYbeta = atan2(m_dMatriz[0][2], m_dMatriz[2][2]);
                m_sCoordXYZeulerZYZ.eulerZgamma = atan2(m_dMatriz[1][0], m_dMatriz[1][1]);
        }

        // Pasamos los ángulos a grados.
        m_sCoordXYZeulerZYZ.eulerZalfa *= (180/PI);
        m_sCoordXYZeulerZYZ.eulerYbeta *= (180/PI);
        m_sCoordXYZeulerZYZ.eulerZgamma *= (180/PI);


        TRACE(_T("Coordenadas XYZeulerZYZ: %f %f %f %f %f %f\n"),
                        m_sCoordXYZeulerZYZ.X, m_sCoordXYZeulerZYZ.Y, m_sCoordXYZeulerZYZ.Z,
```

```
                        m_sCoordXYZeulerZYZ.eulerZalfa, m_sCoordXYZeulerZYZ.eulerYbeta,
m_sCoordXYZeulerZYZ.eulerZgamma);
}

////////////////////////////////////////////////////////////////////////////
//
// Método miembro que sirve para componer un punto respecto a otro.
// Para ello hace una multiplicación de las matrices de transformación
// homogénea de los 2 puntos.
// Debemos premultiplicar la matriz de transformación homogénea de esta
// instancia de la clase CPuntoTrayectoria por la matriz de la instancia
// pasada como parámetro. Lo hacemos elemento a elemento.
//
// Parámetros:
//       - Referencia constante a CPuntoTrayectoria que contiene el punto
//         respecto al cual queremos componer el nuestro.
//
////////////////////////////////////////////////////////////////////////////
void CPuntoTrayectoria::Componer(const CPuntoTrayectoria& Inicio)
{
        double dMatrizAux[4][4];
        int nIndex, nIndex2;

        for (nIndex = 0; nIndex < 4; nIndex++)
                for (nIndex2 = 0; nIndex2 < 4; nIndex2++)
                        dMatrizAux[nIndex][nIndex2] =
Inicio.m_dMatriz[nIndex][0]*m_dMatriz[0][nIndex2]
                                                                                +
Inicio.m_dMatriz[nIndex][1]*m_dMatriz[1][nIndex2]
                                                                                +
Inicio.m_dMatriz[nIndex][2]*m_dMatriz[2][nIndex2]
                                                                                +
Inicio.m_dMatriz[nIndex][3]*m_dMatriz[3][nIndex2];

        for (nIndex = 0; nIndex < 4; nIndex++)
                for (nIndex2 = 0; nIndex2 < 4; nIndex2++)
                        m_dMatriz[nIndex][nIndex2] = dMatrizAux[nIndex][nIndex2];


        TRACE(_T("Matriz de transformación homogénea compuesta:\n%f %f %f %f\n%f %f %f %f\n%f %f %f
%f\n%f %f %f %f\n"),
                                m_dMatriz[0][0],m_dMatriz[0][1],m_dMatriz[0][2],m_dMatriz[0][3],
                                m_dMatriz[1][0],m_dMatriz[1][1],m_dMatriz[1][2],m_dMatriz[1][3],
                                m_dMatriz[2][0],m_dMatriz[2][1],m_dMatriz[2][2],m_dMatriz[2][3],
                                m_dMatriz[3][0],m_dMatriz[3][1],m_dMatriz[3][2],m_dMatriz[3][3]);
}

////////////////////////////////////////////////////////////////////////////
//
// Método que sirve para realizar la resolución del problema cinemático
// inverso para el robot Stäubli RX90 en el punto almacenado.
// Para ello utiliza los valores almacenados en la matriz de transformación
// homogénea y devuelve el resultado en coordenadas articulares.
//
// Parámetros:
```

```
//  - nBrazo: variable que especifica brazo izquierdo o derecho.
//  - nCodo: variable que especifica codo arriba o abajo.
//  - nMuneca: variable que especifica giro de muñeca en sentido horario o
//        antihorario.
//
////////////////////////////////////////////////////////////////////////////
void CPuntoTrayectoria::ResolverModeloCinematicoInverso(int nBrazo, int nCodo, int nMuneca)
{
        double dX = m_dMatriz[0][3];
        double dY = m_dMatriz[1][3];
        double dZ = m_dMatriz[2][3];

        double dSeno[6] = {0,0,0,0,0,0};
        double dCoseno[6] = {0,0,0,0,0,0};
        double theta23 = 0;
        double dSeno23 = 0;
        double dCoseno23= 0;

        double dK, dAux1, dAux2, dAux3, dAux4, dAux5;

        const double PI = 3.14159265358979310862 4468;

        //      El parámetro d6 lo hemos metido con un valor aproximado
        //  Según el proyecto de Manolo es de unos 275 mm + 85 mm;
        //  De todas formas, para las pruebas (como se hacen sin la fresa)
        //      pongo la medida del extremo con la garra.
        m_sParam.d6=360;
//      m_sParam.d1=420;
//      m_sParam.d6=210;
        m_sParam.a2=450;
        m_sParam.d4=450;
        m_sParam.a3=0;
        m_sParam.d3=0;

        // Deducción del tamaño de la herramienta                          // Si quito esto no
funciona el método simplificado. Desactivado aki xq ya lo hacemos en main.
        dX = dX - m_sParam.d6*m_dMatriz[0][2];
        dY = dY - m_sParam.d6*m_dMatriz[1][2];
        dZ = dZ - m_sParam.d6*m_dMatriz[2][2];

        dK=(dX*dX + dY*dY + dZ*dZ - m_sParam.a2*m_sParam.a2 - m_sParam.a3*m_sParam.a3 -
m_sParam.d3*m_sParam.d3 - m_sParam.d4*m_sParam.d4)/(2*m_sParam.a2);

        // Cálculo de ángulos de las articulaciones

        // Calculamos theta1
//      m_sCoordTheta.theta[0] = atan2(-nBrazo*dY,-nBrazo*dX);
        m_sCoordTheta.theta[0] = atan2(dY,dX);
        dSeno[0] = sin(m_sCoordTheta.theta[0]);
        dCoseno[0] = cos(m_sCoordTheta.theta[0]);


        // Calculamos theta2
        dAux1=sqrt(dX*dX+dY*dY);
        dAux2=dX*dX+dY*dY+dZ*dZ;
```

```
        dAux3=atan2(dZ,dAux1);
        dAux4=acos((m_sParam.d4*m_sParam.d4-dAux2-m_sParam.a2*m_sParam.a2)/(-
2*sqrt(dAux2)*m_sParam.a2));
        m_sCoordTheta.theta[1]=-(dAux3+dAux4);
        dSeno[1]=sin(m_sCoordTheta.theta[1]);
        dCoseno[1]=cos(m_sCoordTheta.theta[1]);

        // Calculamos theta3
        m_sCoordTheta.theta[2]=PI/2+2*dAux4;
        dSeno[2] = sin(m_sCoordTheta.theta[2]);
        dCoseno[2] = cos(m_sCoordTheta.theta[2]);

        // Calculamos el seno y el coseno de (theta2+theta3)
        dSeno23 = sin(m_sCoordTheta.theta[1] + m_sCoordTheta.theta[2]);
        dCoseno23 = cos(m_sCoordTheta.theta[1] + m_sCoordTheta.theta[2]);

        // Calculamos theta4
// La matriz 4T0 se multiplicaría por el vector de orientacion
// de la fresa m_dMatriz[i][2] para obtener los nuevos vectores de
// orientacion en el sistema de referencia de la articulacion 4 y 5.
// se multulicaría R*P, siendo wRm=-(mRw)t
// Luego:  (el signo negativo no sé por qué no lo pone
//                       |c1c23 s1c23 -s23| |m_dMatriz[0][2]|
//                       | -s1    c1          0 |*|m_dMatriz[1][2]|
//                       |c1s23 s1s23  c23| |m_dMatriz[2][2]|

        dAux1 = nMuneca*(dCoseno[0]*m_dMatriz[1][2] - dSeno[0]*m_dMatriz[0][2]);
        dAux2 = nMuneca*(dCoseno[0]*dCoseno23*m_dMatriz[0][2] + dSeno[0]*dCoseno23*m_dMatriz[1][2] -
dSeno23*m_dMatriz[2][2]);

        m_sCoordTheta.theta[3]=atan2(dAux1, dAux2);
        dSeno[3] = sin(m_sCoordTheta.theta[3]);
        dCoseno[3] = cos(m_sCoordTheta.theta[3]);

        // Calculamos theta5
        dAux1 = m_dMatriz[0][2]*(dCoseno[0]*dCoseno23*dCoseno[3] - dSeno[0]*dSeno[3]) +
m_dMatriz[1][2]*(dSeno[0]*dCoseno23*dCoseno[3] + dCoseno[0]*dSeno[3]) -
m_dMatriz[2][2]*dCoseno[3]*dSeno23;
        dAux2 = m_dMatriz[0][2]*dCoseno[0]*dSeno23 + m_dMatriz[1][2]*dSeno[0]*dSeno23 +
m_dMatriz[2][2]*dCoseno23;

        m_sCoordTheta.theta[4]=atan2(dAux1, dAux2);
        dSeno[4] = sin(m_sCoordTheta.theta[4]);
        dCoseno[4] = cos(m_sCoordTheta.theta[4]);

        // Calculamos theta6
        dAux1 = -dCoseno[0]*dCoseno23*dSeno[3] - dSeno[0]*dCoseno[3];
        dAux2 = -dSeno[0]*dCoseno23*dSeno[3] + dCoseno[0]*dCoseno[3];
        dAux3 = dSeno[3]*dSeno23;
        dAux4 = m_dMatriz[0][0]*dAux1 + m_dMatriz[1][0]*dAux2 + m_dMatriz[2][0]*dAux3;
        dAux5 = m_dMatriz[0][1]*dAux1 + m_dMatriz[1][1]*dAux2 + m_dMatriz[2][1]*dAux3;

        m_sCoordTheta.theta[5]=atan2(dAux4, dAux5);
        //dSeno[5] = sin(m_sCoordTheta.theta[5]);
        //dCoseno[5] = cos(m_sCoordTheta.theta[5]);
```

```
        // Pasamos los ángulos a grados, dentro del rango [0,360).
        int nIndex;
        for (nIndex = 0; nIndex < 5; nIndex++)
        {
                m_sCoordTheta.theta[nIndex] *= (180/PI);

                while (m_sCoordTheta.theta[nIndex] >= 360)
                        m_sCoordTheta.theta[nIndex] -= 360;
        }

//      CString strMsg;
//      strMsg.Format("Coordenadas articulares:
\ntheta[0]=%f\ntheta[1]=%f\ntheta[2]=%f\ntheta[3]=%f\ntheta[4]=%f\ntheta[5]=%f",m_sCoordTheta.theta[0
],m_sCoordTheta.theta[1],m_sCoordTheta.theta[2],m_sCoordTheta.theta[3],m_sCoordTheta.theta[4],m_sCoor
dTheta.theta[5]);
//      AfxMessageBox(strMsg);

        TRACE(_T("Coordenadas articulares: %f %f %f %f %f %f\n"), m_sCoordTheta.theta[0],
m_sCoordTheta.theta[1],
                                m_sCoordTheta.theta[2], m_sCoordTheta.theta[3],
m_sCoordTheta.theta[4], m_sCoordTheta.theta[5]);
}

////////////////////////////////////////////////////////////////////////////
//
// Método que sirve para comprobar que el punto es alcanzable.
// Para ello comprueba que el valor de las coordenadas articulares está dentro
// del rango alcanzable de cada articulación.
//      theta1 = theta[0] = (-160,160)
//      theta2 = theta[1] = (-200,35)                   (El rango teórico es
//      -(137.5,137.5), pero el origen es -90 y no 0. En la práctica, he llegado
//      a (-207.75,41.96) hasta la base y a (-220.15,43.05) hasta el suelo)
//      theta3 = theta[2] = (-52.5,232.5)         (El rango teórico es
//      (-142.5,142.5), pero el origen es 90 y no 0)
//      theta4 = theta[3] = (-270,270)
//      theta1 = theta[4] = (-105,120)
//      theta1 = theta[5] = (-270,270)
//
// Devuelve TRUE si el punto es alcanzable y FALSE si no lo es (si el valor de
// alguna coordenada articular está fuera de rango).
//
////////////////////////////////////////////////////////////////////////////
BOOL CPuntoTrayectoria::EsAlcanzable()
{
        BOOL fRes = TRUE;

        // Comprobamos que el valor de todas las variables articulares esté dentro
        // del rango alcanzable:
        if ((m_sCoordTheta.theta[0] < -160) || (m_sCoordTheta.theta[0] > 160))
        {
                fRes = FALSE;
                TRACE(_T("ERROR: La coordenada articular theta1 = %f no está dentro del rango [-
160,160]\n"), m_sCoordTheta.theta[0]);
        }
```

```
        else if ((m_sCoordTheta.theta[1] < -200) || (m_sCoordTheta.theta[1] > 35))

        {
                fRes = FALSE;
                TRACE(_T("ERROR: La coordenada articular theta2 = %f no está dentro del rango [-
200,35]\n"), m_sCoordTheta.theta[1]);
        }
        else if ((m_sCoordTheta.theta[2] < -52.5) || (m_sCoordTheta.theta[2] > 232.5))
        {
                fRes = FALSE;
                TRACE(_T("ERROR: La coordenada articular theta3 = %f no está dentro del rango [-
52.5,232.5]\n"), m_sCoordTheta.theta[2]);
        }
        else if ((m_sCoordTheta.theta[3] < -270) || (m_sCoordTheta.theta[3] > 270))
        {
                fRes = FALSE;
                TRACE(_T("ERROR: La coordenada articular theta4 = %f no está dentro del rango [-
270,270]\n"), m_sCoordTheta.theta[3]);
        }
        else if ((m_sCoordTheta.theta[4] < -105) || (m_sCoordTheta.theta[4] > 120))
        {
                fRes = FALSE;
                TRACE(_T("ERROR: La coordenada articular theta5 = %f no está dentro del rango [-
105,120]\n"), m_sCoordTheta.theta[4]);
        }
        else if ((m_sCoordTheta.theta[5] < -270) || (m_sCoordTheta.theta[5] > 270))
        {
                fRes = FALSE;
                TRACE(_T("ERROR: La coordenada articular theta6 = %f no está dentro del rango [-
270,270]\n"), m_sCoordTheta.theta[5]);
        }

        return fRes;
}


////////////////////////////////////////////////////////////////////////////
//
// Método que sirve para fijar las coordenadas XYZeulerZYZ del punto.
//
// Parámetros:
//      - Array double[6] donde están almacenadas las coordenadas a guardar.
//
//      QUIZAS MEJOR const double* &datos
//
////////////////////////////////////////////////////////////////////////////
void CPuntoTrayectoria::SetXYZvector(double* dCoordenadas)
{
        m_sCoordXYZvector.X = dCoordenadas[0];
        m_sCoordXYZvector.Y = dCoordenadas[1];
        m_sCoordXYZvector.Z = dCoordenadas[2];
        m_sCoordXYZvector.i = dCoordenadas[3];
        m_sCoordXYZvector.j = dCoordenadas[4];
        m_sCoordXYZvector.k = dCoordenadas[5];
```

```
        TRACE(_T("Coordenadas XYZvector: %f %f %f %f %f %f\n"),
                        m_sCoordXYZvector.X, m_sCoordXYZvector.Y, m_sCoordXYZvector.Z,
                        m_sCoordXYZvector.i, m_sCoordXYZvector.j, m_sCoordXYZvector.k);
}

////////////////////////////////////////////////////////////////////////////
//
// Método que sirve para fijar las coordenadas XYZeulerZYZ del punto.
//
// Parámetros:
//       - Array double[6] donde están almacenadas las coordenadas a guardar.
//
////////////////////////////////////////////////////////////////////////////
void CPuntoTrayectoria::SetXYZeulerZYZ(double* dCoordenadas)
{
        m_sCoordXYZeulerZYZ.X = dCoordenadas[0];
        m_sCoordXYZeulerZYZ.Y = dCoordenadas[1];
        m_sCoordXYZeulerZYZ.Z = dCoordenadas[2];
        m_sCoordXYZeulerZYZ.eulerZalfa = dCoordenadas[3];
        m_sCoordXYZeulerZYZ.eulerYbeta = dCoordenadas[4];
        m_sCoordXYZeulerZYZ.eulerZgamma = dCoordenadas[5];
}

////////////////////////////////////////////////////////////////////////////
//
// Método que sirve para obtener las coordenadas XYZeulerZYZ del punto.
//
// Parámetros:
//       - Array double[6] donde se van a almacenar las coordenadas requeridas.
//
//       QUIZAS MEJOR double* &datos
//
////////////////////////////////////////////////////////////////////////////
void CPuntoTrayectoria::GetXYZeulerZYZ(double* dCoordenadas)                        //Quizas mejor
double* &datos
{
        dCoordenadas[0] = m_sCoordXYZeulerZYZ.X;
        dCoordenadas[1] = m_sCoordXYZeulerZYZ.Y;
        dCoordenadas[2] = m_sCoordXYZeulerZYZ.Z;
        dCoordenadas[3] = m_sCoordXYZeulerZYZ.eulerZalfa;
        dCoordenadas[4] = m_sCoordXYZeulerZYZ.eulerYbeta;
        dCoordenadas[5] = m_sCoordXYZeulerZYZ.eulerZgamma;
}

////////////////////////////////////////////////////////////////////////////
//
// Método que sirve para crear la matriz de transformación homogénea a partir
// de 3 puntos en coordenadas cartesianas (p0,p1,p2).
// p0 = Esquina superior iqda.
// p1 = Esquina superior derecha.
// p2 = Esquina inferior derecha.
//
// Parámetros:
//       - Array double[3][3] donde están almacenados las coordenadas cartesianas
//         de los 3 puntos.
```

```
//
//      QUIZAS MEJOR double* &datos
//
////////////////////////////////////////////////////////////////////////////
void CPuntoTrayectoria::CrearMatriz(Vector* vPuntos)
{
        Vector vAux, vU, vV;
        double dModulo;

//      Definir clases CMatriz y CVector.
//      http://www.vjuegos.org/modules.php?name=Content&pa=showpage&pid=39

        // La matriz de transformación homogénea será de la forma:
        // [ u   v   w : p0 ]
        // [.............]
        // [ 0   0   0 :  1 ]


//Rellenamos la 1ª columna con el vector unitario diferencia entre p0 y p1(u).
        vAux.x = vPuntos[1].x - vPuntos[0].x;
        vAux.y = vPuntos[1].y - vPuntos[0].y;
        vAux.z = vPuntos[1].z - vPuntos[0].z;

        dModulo = sqrt(vAux.x*vAux.x + vAux.y*vAux.y + vAux.z*vAux.z);

        m_dMatriz[0][0] = vU.x = vAux.x/dModulo;
        m_dMatriz[1][0] = vU.y = vAux.y/dModulo;
        m_dMatriz[2][0] = vU.z = vAux.z/dModulo;


        // Rellenamos la 2ª columna con el vector unitario diferencia entre p0 y p2 (v).
        vAux.x = vPuntos[2].x - vPuntos[0].x;
        vAux.y = vPuntos[2].y - vPuntos[0].y;
        vAux.z = vPuntos[2].z - vPuntos[0].z;

        dModulo = sqrt(vAux.x*vAux.x + vAux.y*vAux.y + vAux.z*vAux.z);

        m_dMatriz[0][1] = vV.x = vAux.x/dModulo;
        m_dMatriz[1][1] = vV.y = vAux.y/dModulo;
        m_dMatriz[2][1] = vV.z = vAux.z/dModulo;


        // Rellenamos la 3ª columna con el vector unitario que completa la base ortonormal (w).
        vAux.x = vU.y*vV.z - vU.z*vV.y;
        vAux.y = vU.z*vV.x - vU.x*vV.z;
        vAux.z = vU.x*vV.y - vU.y*vV.x;

        dModulo = sqrt(vAux.x*vAux.x + vAux.y*vAux.y + vAux.z*vAux.z);

        m_dMatriz[0][2] = vAux.x/dModulo;
        m_dMatriz[1][2] = vAux.y/dModulo;
        m_dMatriz[2][2] = vAux.z/dModulo;

        // Rellenamos la 4ª columna con el punto p0.
        m_dMatriz[0][3] = vPuntos[0].x;
```

```
        m_dMatriz[1][3] = vPuntos[0].y;
        m_dMatriz[2][3] = vPuntos[0].z;


        TRACE(_T("Matriz de transformación homogénea:\n%f %f %f %f\n%f %f %f %f\n%f %f %f %f\n%f %f
%f %f\n"),
                        m_dMatriz[0][0],m_dMatriz[0][1],m_dMatriz[0][2],m_dMatriz[0][3],
                        m_dMatriz[1][0],m_dMatriz[1][1],m_dMatriz[1][2],m_dMatriz[1][3],
                        m_dMatriz[2][0],m_dMatriz[2][1],m_dMatriz[2][2],m_dMatriz[2][3],
                        m_dMatriz[3][0],m_dMatriz[3][1],m_dMatriz[3][2],m_dMatriz[3][3]);
}


////////////////////////////////////////////////////////////////////////////
//
// Método que sirve para invertir la matriz de transformación homogénea (4x4)
// por el método de Gauss-Jordan.
//
// Realizada por Óscar Collazo, adaptada por Manuel Vargas. Luego por Manuel Gómez.
//
// Escribe la matriz inversa en m_dMatriz.
//
////////////////////////////////////////////////////////////////////////////
void CPuntoTrayectoria::InvertirMatriz()
{
        double dMatrizInv[4][4];
        double dMatrizAux[4][4];
        int nIndex, nIndex2;
        int i, j, f;
        int nEncontrePivote;
        double dPivote1, dPivote2, dAux;

        // Inicialmente, la matriz inversa será la identidad:
        for (nIndex = 0; nIndex < 4; nIndex++)
        {
                for (nIndex2 = 0; nIndex2 < 4; nIndex2++)
                {
                        if (nIndex == nIndex2)
                                dMatrizInv[nIndex][nIndex2] = 1;
                        else
                                dMatrizInv[nIndex][nIndex2] = 0;
                }
        }

        // La matriz dMatrizAux es inicialmente igual a la original.
        for (nIndex = 0; nIndex < 4; nIndex++)
                for (nIndex2 = 0; nIndex2 < 4; nIndex2++)
                        dMatrizAux[nIndex][nIndex2] = m_dMatriz[nIndex][nIndex2];

        // Diagonalización inferior.
        for (f=0; f<4; f++)
        {
      // Tomamos el elemento de la diagonal correspondiente a la fila en la que estamos:
                dPivote1 = dMatrizAux[f][f];
```

```
      // Si el pivote es prácticamente cero, debemos intercambiar la fila actual por una
      // de las inferiores que no tenga ese problema.

/*        if (_Despreciable(dPivote1))
         // Un nro. real lo consideraremos despreciablemente pequeño si se cumple esa
         // condición. Realmente, ésta es una cota para los float pero para los double
         // podría ser bastante menor(1e-307 )
         // Una división en la que aparecezca un nro. de este orden en el denominador
         // es arriesgada (aunque todo depende del numerador, por supuesto).
         #define _Despreciable(x) (fabs(x) < 1e-37)
*/
                if (fabs(dPivote1) < 1e-50 )
                {
                        nEncontrePivote = 0;
                for (i=f+1; i<4; i++)
                        {
                        dPivote1 = dMatrizAux[i][f];

                                if(! (fabs(dPivote1) < 1e-50))
                                {
                // Hemos encontrado un elemento no nulo, intercambiamos filas:
                                        nEncontrePivote = 1;
                                        for (j=0; j<4; j++)
                                        {
                // Intercambio los elementos de ambas filas:
                                                dAux = dMatrizInv[i][j];
                                                dMatrizInv[i][j] = dMatrizInv[f][j];
                                                dMatrizInv[f][j] = dAux;
                                                if (j >= f)
                                                {
                                                dAux = dMatrizAux[i][j];
                                                        dMatrizAux[i][j] = dMatrizAux[f][j];
                                                        dMatrizAux[f][j] = dAux;
                                                }
                                        }

                        // Salimos del bucle, para conservar en dPivote1 el valor.
                   break;
                }
             }
             // Si todos los elementos inferiores de esa columna son cero => la matriz no es invertible:
             if (!nEncontrePivote)
                return;//return(1);
      }


                // En primer lugar dividimos la fila actual por su pivote:
                for (i=0; i<4; i++)
                {
                        dMatrizInv[f][i] /= dPivote1;
                        if (i >= f)
// En el caso de la matriz que se está llevando a la identidad, todos los
// elementos previos al de la diagonal ya deben de ser cero.
                        dMatrizAux[f][i] /= dPivote1;
                }
```

```
                // Ahora se diagonaliza la parte inferior a la diagonal principal
                for (i=f+1; i<4; i++)
                {
                        dPivote2 = dMatrizAux[i][f];
                        for (j=0; j<4; j++)
                        {
                                dMatrizInv[i][j] -= dMatrizInv[f][j] * dPivote2;
                                if (j >= f)
                                        dMatrizAux[i][j] -= dMatrizAux[f][j] * dPivote2;
                        }
                }
        }

        // Ahora comienza la diagonalización superior:
   for (f=3; f>=0; f--)
        {
                for (i=f-1; i>=0; i--)
                {
                dPivote2 = dMatrizAux[i][f];
                        for (j=3; j>=0; j--)
                        {
                                dMatrizInv[i][j] -= dMatrizInv[f][j] * dPivote2;
                                if (j <= f)
                                        dMatrizAux[i][j] -= dMatrizAux[f][j] * dPivote2;
                        }
                }
        }

   // Si hemos llegado hasta aquí, no ha habido error. Pasamos los valores de
   // dMatrizInv a la matriz mienbro.
        for (nIndex = 0; nIndex < 4; nIndex++)
                for (nIndex2 = 0; nIndex2 < 4; nIndex2++)
                        m_dMatriz[nIndex][nIndex2] = dMatrizInv[nIndex][nIndex2];



        TRACE(_T("Matriz de transformación homogénea invertida:\n%f %f %f %f\n%f %f %f %f\n%f %f %f
%f\n%f %f %f %f\n"),
                        m_dMatriz[0][0],m_dMatriz[0][1],m_dMatriz[0][2],m_dMatriz[0][3],
                        m_dMatriz[1][0],m_dMatriz[1][1],m_dMatriz[1][2],m_dMatriz[1][3],
                        m_dMatriz[2][0],m_dMatriz[2][1],m_dMatriz[2][2],m_dMatriz[2][3],
                        m_dMatriz[3][0],m_dMatriz[3][1],m_dMatriz[3][2],m_dMatriz[3][3]);
}

//////////////////////////////////////////////////////////////////
//                                                              //
//      Método similar a CrearMatriz, con la misma finalidad para   //
//      crear una matriz de transformación homogénea a partir de    //
//      los tres puntos que se dan como iniciales.              //
//                                                              //
//      Esta función, además, tiene en cuenta que los puntos leídos  //
//      del robot real son válidos para un extremo de 85mm de       //
//      longitud. Pero la fresa con la fresa mide en realidad 360mm  //
//      Esta matriz resultante se desplaza 275mm en la dirección    //
```

```
//      del eje Z del efector final.                                        //
//                                                                          //
//////////////////////////////////////////////////////////////////////////

void CPuntoTrayectoria::CrearMatrizDesplazada(Vector *vPuntos)
{
        Vector vAux, vU, vV;
        double dModulo;

        int diferencia=275;
        int escala=150;
//      Definir clases CMatriz y CVector.
//      http://www.vjuegos.org/modules.php?name=Content&pa=showpage&pid=39

        // La matriz de transformación homogénea será de la forma:
        // [ u   v   w : p0 ]
        // [·····················]
        // [ 0   0   0 :  1 ]


        // Rellenamos la 1ª columna con el vector unitario diferencia entre p0 y p1(u).
        vAux.x = vPuntos[1].x - vPuntos[0].x;
        vAux.y = vPuntos[1].y - vPuntos[0].y;
        vAux.z = vPuntos[1].z - vPuntos[0].z;

        dModulo = sqrt(vAux.x*vAux.x + vAux.y*vAux.y + vAux.z*vAux.z);

        m_dMatriz[0][0] = vU.x = vAux.x/dModulo;
        m_dMatriz[1][0] = vU.y = vAux.y/dModulo;
        m_dMatriz[2][0] = vU.z = vAux.z/dModulo;


        // Rellenamos la 2ª columna con el vector unitario diferencia entre p0 y p2 (v).
        vAux.x = vPuntos[2].x - vPuntos[0].x;
        vAux.y = vPuntos[2].y - vPuntos[0].y;
        vAux.z = vPuntos[2].z - vPuntos[0].z;

        dModulo = sqrt(vAux.x*vAux.x + vAux.y*vAux.y + vAux.z*vAux.z);

        m_dMatriz[0][1] = vV.x = vAux.x/dModulo;
        m_dMatriz[1][1] = vV.y = vAux.y/dModulo;
        m_dMatriz[2][1] = vV.z = vAux.z/dModulo;


        // Rellenamos la 3ª columna con el vector unitario que completa la base ortonormal (w).
        vAux.x = vU.y*vV.z - vU.z*vV.y;
        vAux.y = vU.z*vV.x - vU.x*vV.z;
        vAux.z = vU.x*vV.y - vU.y*vV.x;

        dModulo = sqrt(vAux.x*vAux.x + vAux.y*vAux.y + vAux.z*vAux.z);

        m_dMatriz[0][2] = vAux.x/dModulo;
        m_dMatriz[1][2] = vAux.y/dModulo;
        m_dMatriz[2][2] = vAux.z/dModulo;
```

```
	// Rellenamos la 4ª columna con el punto p0.
	m_dMatriz[0][3] = vPuntos[0].x+diferencia*m_dMatriz[0][2];
	m_dMatriz[1][3] = vPuntos[0].y+diferencia*m_dMatriz[1][2];
	m_dMatriz[2][3] = vPuntos[0].z+diferencia*m_dMatriz[2][2];

	vPuntos[0].x = vPuntos[0].x+(diferencia-escala)*m_dMatriz[0][2];
	vPuntos[0].y = vPuntos[0].y+(diferencia-escala)*m_dMatriz[1][2];
	vPuntos[0].z = vPuntos[0].z+(diferencia-escala)*m_dMatriz[2][2];

	vPuntos[1].x = vPuntos[1].x+(diferencia-escala)*m_dMatriz[0][2];
	vPuntos[1].y = vPuntos[1].y+(diferencia-escala)*m_dMatriz[1][2];
	vPuntos[1].z = vPuntos[1].z+(diferencia-escala)*m_dMatriz[2][2];

	vPuntos[2].x = vPuntos[2].x+(diferencia-escala)*m_dMatriz[0][2];
	vPuntos[2].y = vPuntos[2].y+(diferencia-escala)*m_dMatriz[1][2];
	vPuntos[2].z = vPuntos[2].z+(diferencia-escala)*m_dMatriz[2][2];

	ofstream cout("puntocentral.txt");
	for (int i=0;i<3;i++)
	{
		cout << vPuntos[i].x << " " << vPuntos[i].y << " " << vPuntos[i].z;
		cout << "\n";
	}
	cout.close();

	ofstream mat("matriz.txt");
	for (i=0;i<4;i++)
	{
		for (int j=0;j<4;j++)
			mat << m_dMatriz[i][j] << " ";
		mat << ";" << "\n";
	}
	mat.close();


	TRACE(_T("Matriz de transformación homogénea:\n%f %f %f %f\n%f %f %f %f\n%f %f %f %f\n%f %f
%f %f\n"),
			m_dMatriz[0][0],m_dMatriz[0][1],m_dMatriz[0][2],m_dMatriz[0][3],
			m_dMatriz[1][0],m_dMatriz[1][1],m_dMatriz[1][2],m_dMatriz[1][3],
			m_dMatriz[2][0],m_dMatriz[2][1],m_dMatriz[2][2],m_dMatriz[2][3],
			m_dMatriz[3][0],m_dMatriz[3][1],m_dMatriz[3][2],m_dMatriz[3][3]);
}

//////////////////////////////////////////////////////////////
//                                                          //
//      Método que permite a la clase CMainFrame leer las   //
//      cooedenadas XYZeulerZYZ a pesar de ser atributos    //
//      privados.                                           //
//                                                          //
//////////////////////////////////////////////////////////////
double CPuntoTrayectoria::LeerCoordXYZeulerZYZ(int j)
{
	double coord;

	switch (j)
```

```
        {
        case 0:
                coord = m_sCoordXYZeulerZYZ.X;
                break;
        case 1:
                coord = m_sCoordXYZeulerZYZ.Y;
                break;
        case 2:
                coord = m_sCoordXYZeulerZYZ.Z;
                break;
        case 3:
                coord = m_sCoordXYZeulerZYZ.eulerZalfa;
                break;
        case 4:
                coord = m_sCoordXYZeulerZYZ.eulerYbeta;
                break;
        case 5:
                coord = m_sCoordXYZeulerZYZ.eulerZgamma;
                break;
        }
        return coord;
}
```

B.1.8. CDlgSimulacion.

B.1.8.1. DlgSimulacion.h

```
////////////////////////////////////////////////////////////////////
//                                                                //
//      DlgSimulacion.h: Defición de la clase CDlgSimulacion.     //
//                                                                //
//      Clase que implementa un cuadro de diálogo para            //
//      preguntar al usuario si desea acceder a la aplicación     //
//      de simulación de la trayectoria.                          //
//                                                                //
//      Creado:  Alejandro Guija Rodríguez.                       //
//                                                                //
////////////////////////////////////////////////////////////////////
#if !defined(AFX_DLGSIMULACION_H__50FECF71_F997_4E79_838B_5DC1334FC1F9__INCLUDED_)
#define AFX_DLGSIMULACION_H__50FECF71_F997_4E79_838B_5DC1334FC1F9__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// DlgSimulacion.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CDlgSimulacion dialog

class CCorte3DApp;

class CDlgSimulacion : public CDialog
{
```

```
// Construction
public:
        CDlgSimulacion(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(CDlgSimulacion)
        enum { IDD = IDD_SIMULACION };
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CDlgSimulacion)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CDlgSimulacion)
        virtual void OnOK();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_DLGSIMULACION_H__50FECF71_F997_4E79_838B_5DC1334FC1F9__INCLUDED_)
```

B.1.8.2. DlgSimulacion.cpp

```
// DlgSimulacion.cpp : implementation file
//

#include "stdafx.h"
#include "Corte3D.h"
#include "DlgSimulacion.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CDlgSimulacion dialog


CDlgSimulacion::CDlgSimulacion(CWnd* pParent /*=NULL*/)
        : CDialog(CDlgSimulacion::IDD, pParent)
{
```

```
	//{{AFX_DATA_INIT(CDlgSimulacion)
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
}


void CDlgSimulacion::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CDlgSimulacion)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CDlgSimulacion, CDialog)
	//{{AFX_MSG_MAP(CDlgSimulacion)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDlgSimulacion message handlers

void CDlgSimulacion::OnOK()
{
	// TODO: Add extra validation here

// Para llamar a Rx90.exe, debe estar en la carpeta c:\windows\system32\
// pero los archivos rx90.x, pieza.x y sad.bmp, que son cargados desde el
// ejecutable Rx90.exe deben estar en la misma carpeta del ejecutable de
// Corte3d.exe

	system("Rx90.exe");

	CDialog::OnOK();
}
```

B.2. APLICACIÓN 'RX90'.

B.2.1. CRx90App.

B.2.1.1. Rx90.h

```
/////////////////////////////////////////////////////////////
//                                                         //
//      Rx90.h: Archivo de cabecera de la clase principal  //
//      CRx90App.                                          //
//                                                         //
//      Creado por F. Javier Rueda 2001                    //
//      Modificado por Alejandro Guija Rodríguez           //
//                                                         //
/////////////////////////////////////////////////////////////

#if !defined(AFX_RX90_H__6EF5082E_838B_11D1_B61B_444553540000__INCLUDED_)
#define AFX_RX90_H__6EF5082E_838B_11D1_B61B_444553540000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
        #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // main symbols
#include "robot.h"
#include "PuntoTrayectoria.h"
#include "Captura.h"

class CMainFrame;
//class CPuntoTrayectoria;

/////////////////////////////////////////////////////////////////////////////
// CRx90App:
// See Rx90.cpp for the implementation of this class
//

class CRx90App : public CWinApp
{
public:
        CRx90App();
        CMainFrame *TheFrame;
        CRobot RX90Robot;

        CCaptura* m_pPuntosTrayectoria;
        CPuntoTrayectoria* m_pPuntoTrayectoria;
        CPuntoTrayectoria m_MundoReal;
        CPuntoTrayectoria m_MundoVirtual;
//      CPuntoTrayectoria m_Transformacion;
```

```
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CRx90App)
        public:
        virtual BOOL InitInstance();
        virtual BOOL OnIdle(LONG lCount);
        //}}AFX_VIRTUAL

// Implementation
        //{{AFX_MSG(CRx90App)
        afx_msg void OnAppAbout();
        // NOTE - the ClassWizard will add and remove member functions here.
        //    DO NOT EDIT what you see in these blocks of generated code !
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before //the previous line.

#endif // !defined(AFX_RX90_H__6EF5082E_838B_11D1_B61B_444553540000__INCLUDED_)
```

B.2.1.2. Rx90.cpp

```
////////////////////////////////////////////////////////////
//                                                        //
// Rx90.cpp : Archivo que implementa las funciones que    //
// inicializan algunas variables, punteros y el sistema   //
// DirectX.                                               //
//                                                        //
////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Rx90.h"

#include "MainFrm.h"
#include "Rx90Doc.h"
#include "Rx90View.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CRx90App

BEGIN_MESSAGE_MAP(CRx90App, CWinApp)
        //{{AFX_MSG_MAP(CRx90App)
        ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
        //}}AFX_MSG_MAP
```

```
        // Standard file based document commands
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CRx90App construction

CRx90App::CRx90App()
{
        // TODO: add construction code here,
        // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CRx90App object
CRx90App theApp;

/////////////////////////////////////////////////////////////////////////////
// CRx90App initialization

BOOL CRx90App::InitInstance()
{
        AfxEnableControlContainer();

        // Standard initialization
        // If you are not using these features and wish to reduce the size
        //  of your final executable, you should remove from the following
        //  the specific initialization routines you do not need.

#ifdef _AFXDLL
        Enable3dControls();                     // Call this when using MFC in a shared DLL
#else
        Enable3dControlsStatic();// Call this when linking to MFC statically
#endif

        // Change the registry key under which our settings are stored.
        // TODO: You should modify this string to be something appropriate
        // such as the name of your company or organization.
        SetRegistryKey(_T("Local AppWizard-Generated Applications"));

        LoadStdProfileSettings();  // Load standard INI file options (including MRU)

        // Register the application's document templates.  Document templates
        //  serve as the connection between documents, frame windows and views.

        // The one and only window has been initialized, so show and update it.

        TheFrame=new(CMainFrame);
        BOOL Status= TheFrame->Create (AfxRegisterWndClass(CS_HREDRAW |
CS_VREDRAW,0,(HBRUSH)(COLOR_WINDOW + 1),0),
                                                        "Simulador de Robot Industrial
RX90",
                                                         WS_OVERLAPPEDWINDOW,
                                                         CFrameWnd::rectDefault,

NULL,MAKEINTRESOURCE(IDR_MAINFRAME) ,0);
```

```
        // Inicializamos ventana para DirectX

        m_pMainWnd=(CWnd*)TheFrame;

        TheFrame->D3DInit();

        m_pMainWnd->ShowWindow(SW_SHOW);
        m_pMainWnd->UpdateWindow();

        // Inicializamos algunos punteros.
        TheFrame->m_pRobot=&RX90Robot;
        return TRUE;
}


/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
        CAboutDlg();

// Dialog Data
        //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
        //}}AFX_DATA

        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CAboutDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:
        //{{AFX_MSG(CAboutDlg)
                // No message handlers
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
        //{{AFX_DATA_INIT(CAboutDlg)
        //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAboutDlg)
        //}}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
        //{{AFX_MSG_MAP(CAboutDlg)
                // No message handlers
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CRx90App::OnAppAbout()
{
        CAboutDlg aboutDlg;
        aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// CRx90App message handlers

BOOL CRx90App::OnIdle(LONG lCount)
{
        CWinApp::OnIdle(lCount);
        if(TheFrame->bInitialized == TRUE)
        {
                TheFrame->Render();
                Sleep(50);
        }
        return TRUE;
}
```

B.2.2. CRx90Doc.

B.2.2.1. Rx90Doc.h

```
////////////////////////////////////////////////////////////
//                                                        //
//  Rx90Doc.h: Definición de la clase CRx90Doc.           //
//                                                        //
//      Creado por Microsoft Developer Studio.            //
//                                                        //
////////////////////////////////////////////////////////////

#if !defined(AFX_RX90DOC_H__6EF50834_838B_11D1_B61B_444553540000__INCLUDED_)
#define AFX_RX90DOC_H__6EF50834_838B_11D1_B61B_444553540000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CRx90Doc : public CDocument
{
protected: // create from serialization only
        CRx90Doc();
        DECLARE_DYNCREATE(CRx90Doc)
```

```
// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CRx90Doc)
        public:
        virtual BOOL OnNewDocument();
        virtual void Serialize(CArchive& ar);
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CRx90Doc();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(CRx90Doc)
        // NOTE - the ClassWizard will add and remove member functions here.
        //    DO NOT EDIT what you see in these blocks of generated code !
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_RX90DOC_H__6EF50834_838B_11D1_B61B_444553540000__INCLUDED_)
```

B.2.2.2. Rx90Doc.cpp

```
/////////////////////////////////////////////////////////////
//                                                         //
// Rx90Doc.cpp : implementación de la clase CRx90Doc.      //
//                                                         //
/////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Rx90.h"

#include "Rx90Doc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
```

```
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CRx90Doc

IMPLEMENT_DYNCREATE(CRx90Doc, CDocument)

BEGIN_MESSAGE_MAP(CRx90Doc, CDocument)
        //{{AFX_MSG_MAP(CRx90Doc)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CRx90Doc construction/destruction

CRx90Doc::CRx90Doc()
{
        // TODO: add one-time construction code here
}

CRx90Doc::~CRx90Doc()
{
}

BOOL CRx90Doc::OnNewDocument()
{
        if (!CDocument::OnNewDocument())
                return FALSE;

        // TODO: add reinitialization code here
        // (SDI documents will reuse this document)

        return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CRx90Doc serialization

void CRx90Doc::Serialize(CArchive& ar)
{
        if (ar.IsStoring())
        {
                // TODO: add storing code here
        }
        else
        {
                // TODO: add loading code here
        }
}

/////////////////////////////////////////////////////////////////////////////
```

```
// CRx90Doc diagnostics

#ifdef _DEBUG
void CRx90Doc::AssertValid() const
{
        CDocument::AssertValid();
}

void CRx90Doc::Dump(CDumpContext& dc) const
{
        CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CRx90Doc commands
```

B.2.3. CRx90View.

B.2.3.1. Rx90View.h

```
/////////////////////////////////////////////////////
//                                                 //
//  Rx90View.h: Definición de la clase CRx90View.  //
//                                                 //
//      Creado por Microsoft Developer Studio.     //
//                                                 //
/////////////////////////////////////////////////////

#if !defined(AFX_RX90VIEW_H__6EF50836_838B_11D1_B61B_444553540000__INCLUDED_)
#define AFX_RX90VIEW_H__6EF50836_838B_11D1_B61B_444553540000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CRx90View : public CView
{
protected: // create from serialization only
        CRx90View();
        DECLARE_DYNCREATE(CRx90View)

// Attributes
public:
        CRx90Doc* GetDocument();

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CRx90View)
        public:
```

```
        virtual void OnDraw(CDC* pDC);  // overridden to draw this view
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        protected:
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CRx90View();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(CRx90View)
        // NOTE - the ClassWizard will add and remove member functions here.
        //    DO NOT EDIT what you see in these blocks of generated code !
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in Rx90View.cpp
inline CRx90Doc* CRx90View::GetDocument()
   { return (CRx90Doc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_RX90VIEW_H__6EF50836_838B_11D1_B61B_444553540000__INCLUDED_)
```

B.2.3.2. Rx90View.cpp

```
/////////////////////////////////////////////////////////////
//                                                         //
// Rx90View.cpp : implementación de la clase CRx90View     //
//                                                         //
/////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Rx90.h"

#include "Rx90Doc.h"
#include "Rx90View.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
/////////////////////////////////////////////////////////////////////////////
// CRx90View

IMPLEMENT_DYNCREATE(CRx90View, CView)

BEGIN_MESSAGE_MAP(CRx90View, CView)
	//{{AFX_MSG_MAP(CRx90View)
	// NOTE - the ClassWizard will add and remove mapping macros here.
	//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CRx90View construction/destruction

CRx90View::CRx90View()
{
	// TODO: add construction code here
}

CRx90View::~CRx90View()
{
}

BOOL CRx90View::PreCreateWindow(CREATESTRUCT& cs)
{
	// TODO: Modify the Window class or styles here by modifying
	//  the CREATESTRUCT cs

	return CView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CRx90View drawing

void CRx90View::OnDraw(CDC* pDC)
{
	CRx90Doc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);
	// TODO: add draw code for native data here
}

/////////////////////////////////////////////////////////////////////////////
// CRx90View diagnostics

#ifdef _DEBUG
void CRx90View::AssertValid() const
{
	CView::AssertValid();
}

void CRx90View::Dump(CDumpContext& dc) const
{
	CView::Dump(dc);
```

```
}

CRx90Doc* CRx90View::GetDocument() // non-debug version is inline
{
        ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CRx90Doc)));
        return (CRx90Doc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CRx90View message handlers
```

B.2.4. CMainFrame.

B.2.4.1. MainFrm.h

```
//////////////////////////////////////////////////////////
//                                                      //
//      MainFrm.h : Definición de la clase CMainFrame.  //
//                                                      //
//      Clase que configura la escena creando los frames //
//      para las luces, la cámara, los elementos del robot //
//      y la pieza objetivo; los posiciona, los orienta y //
//      los relaciona unos con otros para que el usuario //
//      pueda interactuar con el robot.                 //
//                                                      //
//      Creado por F. Javier Rueda        2001          //
//      Modificado por Alejandro Guija Rodríguez 2006-2007 //
//                                                      //
//////////////////////////////////////////////////////////

#if !defined(AFX_MAINFRM_H__6EF50832_838B_11D1_B61B_444553540000__INCLUDED_)
#define AFX_MAINFRM_H__6EF50832_838B_11D1_B61B_444553540000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define  INITGUID
#include <d3drmwin.h>
#include "Captura.h"      // Added by ClassView
#include "DlgCamara.h"
#include "PuntoTrayectoria.h"
#include "DlgCoordArt.h"
#include "DlgCoordCart.h"

#define MAX_DRIVERS 5
#define elementos 11

class CRobot;
class CCaptura;
class CDlgCamara;
class CPuntoTrayectoria;
class CDlgCoordArt;
class CDlgCoordCart;
```

```
class CMainFrame : public CFrameWnd
{

protected: // create from serialization only
//       DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:
        CMainFrame();

        // Variables para Inicializacion Direct3D
        LPDIRECTDRAW lpDD;                                  // Puntero a objeto DirectDraw
        LPDIRECT3DRM FAR lpD3DRM;
        LPDIRECTDRAWCLIPPER lpDDClipper;  // DirectDrawClipper object
        BOOL bInitialized;                          // All D3DRM objects are initialized
        LPDIRECT3DRMFRAME lpObject_frame[elementos];       // Vector de sistemas de coordenadas
        LPDIRECT3DRMFRAME lpWorld_frame;

        CRobot* m_pRobot;

// Operations
public:
 BOOL CreateDevAndView(LPDIRECTDRAWCLIPPER lpDDClipper, int driver,
    int width, int height);
 BOOL SetRenderState(void);
 BOOL MyScene(LPDIRECT3DRMDEVICE dev, LPDIRECT3DRMVIEWPORT view,
    LPDIRECT3DRMFRAME lpScene, LPDIRECT3DRMFRAME lpCamera);
 void MakeMyFrames(LPDIRECT3DRMFRAME lpScene, LPDIRECT3DRMFRAME lpCamera,
          LPDIRECT3DRMFRAME * lplpLightFrame1, LPDIRECT3DRMFRAME * lplpLightFrame2,
          LPDIRECT3DRMFRAME * lplpWorld_frame, LPDIRECT3DRMFRAME * lplpObject_frame);

 void MakeMyLights(LPDIRECT3DRMFRAME lpScene, LPDIRECT3DRMFRAME lpCamera,
      LPDIRECT3DRMFRAME lpLightFrame1, LPDIRECT3DRMFRAME lpLightFrame2,
      LPDIRECT3DRMLIGHT * lplpLight1, LPDIRECT3DRMLIGHT * lplpLight2,
          LPDIRECT3DRMLIGHT * lplpLight3);

 void SetMyPositions(LPDIRECT3DRMFRAME lpScene,
      LPDIRECT3DRMFRAME lpCamera, LPDIRECT3DRMFRAME lpLightFrame1,
          LPDIRECT3DRMFRAME lpLightFrame2,LPDIRECT3DRMFRAME lpWorld_frame,
          LPDIRECT3DRMFRAME * lplpObject_frame);

 void MakeMyMesh(LPDIRECT3DRMMESHBUILDER * lplpbase_builder,
                                  LPDIRECT3DRMMESHBUILDER * lplpObject_builder);
 void MakeMyWrap(LPDIRECT3DRMMESHBUILDER base_builder,
                 LPDIRECT3DRMWRAP * lpWrap);
 void AddMyTexture(LPDIRECT3DRMMESHBUILDER lpbase_builder,
                   LPDIRECT3DRMTEXTURE * lplpTex);

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMainFrame)
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        //}}AFX_VIRTUAL
```

```
// Implementation
public:
        Vector InclinacionPieza();
        Vector Punto[3];
        Vector LeePuntoCentral();
        void LeeNombrePieza(char NombrePieza);
        int pos_ant[3];
        int pos[3];
        int theta_ant[6];
        int mhand;
        int Yant;
        int Pant;
        int move;
//  double articulacion[6];
        void CamaraStart(void);
        CDlgCamara *pDlgCamara;
        CDlgCoordArt *pDlgCoordArt;
        CDlgCoordCart *pDlgCoordCart;

        CCaptura Cp;
        CCaptura* pCaptura;
        void SetCamera(double pitch, double yaw);
        void SetPosition(double pos, int eje);
        void CleanUp(void);
        BOOL RenderLoop(void);
        GUID* GetGUID(void);
        BOOL SetMode(void);
        COLORREF D3DCOLOR_2_COLORREF(D3DCOLOR d3dclr);
        BOOL Render(void);
        BOOL EnumDrivers(void);
        D3DInit();
        virtual ~CMainFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
        CStatusBar  m_wndStatusBar;
        CToolBar    m_wndToolBar;

// Generated message map functions
protected:
        //{{AFX_MSG(CMainFrame)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        afx_msg void OnPaint();
        afx_msg BOOL OnEraseBkgnd(CDC* pDC);
        afx_msg void OnActivate(UINT nState, CWnd* pWndOther, BOOL bMinimized);
        afx_msg void OnDestroy();
        afx_msg void OnTimer(UINT nIDEvent);
        afx_msg void OnVerPuntodevista();
        afx_msg void OnMovimientoStart();
        afx_msg void OnMovimientoStop();
        afx_msg void OnMovimientoReset();
        afx_msg void OnMovimientoManualArticular();
```

```
        afx_msg void OnMovimientoManualCartesiana();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_MAINFRM_H__6EF50832_838B_11D1_B61B_444553540000__INCLUDED_)
```

B.2.4.2. MainFrm.cpp

```
/////////////////////////////////////////////////////////////////////////
//                                                                     //
// MainFrm.cpp : Archivo que implementa la clase CMainFrame.           //
//                                                                     //
/////////////////////////////////////////////////////////////////////////

#include <AfxWin.h>
#include <AfxExt.h>
#include <AfxTempl.h>

#include "mmsystem.h"
#include "Rx90.h"
#include "MainFrm.h"
#include "D3dcaps.h"
#include "math.h"

#include "Captura.h"
#include "Resource.h"
#include "PuntoTrayectoria.h"

#include <iostream>
#include <stdlib.h>
#include <string>

#include <iostream>
#include <fstream.h>

//#include "atlbase.h"
//#include "atlstr.h"
#include "comutil.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#define elementos 11
#define PI 3.14159

#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
/////////////////////////////////////////////////////////////////////////////
// CMainFrame

//IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
        //{{AFX_MSG_MAP(CMainFrame)
        ON_WM_CREATE()
        ON_WM_PAINT()
        ON_WM_ERASEBKGND()
        ON_WM_ACTIVATE()
        ON_WM_DESTROY()
        ON_WM_TIMER()
        ON_COMMAND(ID_VER_PUNTODEVISTA, OnVerPuntodevista)
        ON_COMMAND(ID_MOVIMIENTO_START, OnMovimientoStart)
        ON_COMMAND(ID_MOVIMIENTO_STOP, OnMovimientoStop)
        ON_COMMAND(ID_MOVIMIENTO_RESET, OnMovimientoReset)
        ON_COMMAND(ID_MOVIMIENTO_MANUAL_ARTICULAR, OnMovimientoManualArticular)
        ON_COMMAND(ID_MOVIMIENTO_MANUAL_CARTESIANA, OnMovimientoManualCartesiana)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

struct _myglobs {
        LPDIRECT3DRMDEVICE dev;     // Direct3DRM device
    LPDIRECT3DRMVIEWPORT view;  // Direct3DRM viewport through which
                                // the scene is viewed
    LPDIRECT3DRMFRAME scene;    // Master frame in which others are
                                // placed
    LPDIRECT3DRMFRAME camera;   // Frame describing the user's POV
    GUID DriverGUID[MAX_DRIVERS];     // GUIDs of available D3D drivers
    char DriverName[MAX_DRIVERS][50]; // Names of available D3D drivers
    int  NumDrivers;                  // Number of available D3D drivers
    int  CurrDriver;                  // Number of D3D driver currently
                                      // being used
    BOOL bQuit;                 // Program is about to terminate
    BOOL bMinimized;            // Window is minimized
    int BPP;                    // Bit depth of the current display mode

} myglobs;

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

//////////////////////////////////////////////////////////////////////
//                                                                  //
//      Función constructor estándar que inializa algunas variables //
//      y punteros.                                                 //
//                                                                  //
//////////////////////////////////////////////////////////////////////

CMainFrame::CMainFrame()
{
        mhand = 0;
        Pant = 0;
```

```
        Yant = 0;

        move = 0;
        myglobs.dev    = NULL;
        myglobs.view   = NULL;
    myglobs.scene  = NULL;
    myglobs.camera = NULL;

        lpD3DRM = NULL;
        lpDDClipper = NULL;
    bInitialized = FALSE;

        for(int i=0;i<elementos;i++)
                lpObject_frame[i]=NULL;

        lpWorld_frame = NULL;

        //Inicializamos la clase CCaptura
        Cp.m_pFrame=this;
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
        if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
                return -1;

        return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
        if( !CFrameWnd::PreCreateWindow(cs) )
                return FALSE;
        cs.style &= ~(WS_MAXIMIZEBOX);

        return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
        CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
        CFrameWnd::Dump(dc);
}
```

```
#endif //_DEBUG

////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers

/////////////////////////////////////////////////////////////////////
//                                                                 //
//      Función incializadora de las herramientas de DirectX.      //
//                                                                 //
/////////////////////////////////////////////////////////////////////

CMainFrame::D3DInit()
{
        // Inicializamos el sistema DirectX.
    lpD3DRM = NULL;
        CDC *dc;
        HRESULT res;
    RECT rc;

        // Record the current display bits-per-pixel.
        dc = GetDC();
    myglobs.BPP = dc->GetDeviceCaps( BITSPIXEL);
        ReleaseDC( dc);

        // Enumera los dispositivos y crea un objeto DirectDraw
        if (!EnumDrivers())
                return FALSE;

        // Creamos un objeto IDirect3D
        HRESULT Status = Direct3DRMCreate(&lpD3DRM);
    if(Status!=D3DRM_OK)
                AfxMessageBox("Error al crear objeto 3DRM");

        // Create the master scene frame and camera frame.
        res=lpD3DRM->CreateFrame(NULL, &myglobs.scene);
        if(res!=D3DRM_OK)
                AfxMessageBox("Error al CreateFrame");

        res=lpD3DRM->CreateFrame(myglobs.scene, &myglobs.camera);
        if(res!=D3DRM_OK)
                AfxMessageBox("Error al CreateFrame");
        res=myglobs.camera->SetPosition(myglobs.scene,
       D3DVAL(0.0), D3DVAL(35), D3DVAL(-45));
        if(res!=D3DRM_OK)
                AfxMessageBox("Error al CreateFrame");
        res=myglobs.camera->SetOrientation(myglobs.scene,0,0,1,0,1,0);
        if(res!=D3DRM_OK)
                AfxMessageBox("Error al CreateFrame");

        // Create a DirectDrawClipper object and associate the
        // window with it.
        res=DirectDrawCreateClipper(0, &lpDDClipper, NULL);
        if(res!=D3DRM_OK)
                AfxMessageBox("Error al CreateClipper");
```

```
        res=lpDDClipper->SetHWnd( 0, m_hWnd);
        if(res!=D3DRM_OK)
                AfxMessageBox("Error al asignar ventana");

        // Create the D3DRM device by using the selected D3D driver.
        GetClientRect( &rc);
        if (!CreateDevAndView(lpDDClipper,
                              myglobs.CurrDriver, rc.right,
                        rc.bottom))
        {
           return FALSE;
        }

        // Creamos la escena
    if (!MyScene(myglobs.dev, myglobs.view, myglobs.scene,
            myglobs.camera))
                 return FALSE;

        bInitialized = TRUE;  // Initialization completed

        return(1);
}
//////////////////////////////////////////////////////////////////////
//
// SetRenderState
// Set the render quality and shade information.
//
//////////////////////////////////////////////////////////////////////

BOOL CMainFrame::SetRenderState(void)
{
    HRESULT rval;

    // Set the render quality (light toggle, fill mode, shade mode).

    rval = myglobs.dev->SetQuality(
        D3DRMLIGHT_ON | D3DRMFILL_SOLID | D3DRMSHADE_GOURAUD);
    if (rval != D3DRM_OK) {
        return FALSE;
    }

    // If you want to change the dithering mode, call SetDither here.

    // If you want a texture quality other than D3DRMTEXTURE_NEAREST
    // (the default value), call SetTextureQuality here.

    // Set shade information based on current bits-per-pixel.

    switch (myglobs.BPP)
        {
        case 1:
            if (FAILED(myglobs.dev->SetShades(4)))
                goto shades_error;
            if (FAILED(lpD3DRM->SetDefaultTextureShades(4)))
                goto shades_error;
```

```
            break;
        case 16:
            if (FAILED(myglobs.dev->SetShades(32)))
                goto shades_error;
            if (FAILED(lpD3DRM->SetDefaultTextureColors(64)))
                goto shades_error;
            if (FAILED(lpD3DRM->SetDefaultTextureShades(32)))
                goto shades_error;
            break;
        case 24:
        case 32:
            if (FAILED(myglobs.dev->SetShades(256)))
                goto shades_error;
            if (FAILED(lpD3DRM->SetDefaultTextureColors(64)))
                goto shades_error;
            if (FAILED(lpD3DRM->SetDefaultTextureShades(256)))
                goto shades_error;
            break;
    }
    return TRUE;
shades_error:
    return FALSE;
}

//////////////////////////////////////////////////////////////////////
//
// MyScene
// Calls the functions that create the frames, lights, mesh, and
// texture. Releases all interfaces on completion.
//
//////////////////////////////////////////////////////////////////////

BOOL CMainFrame::MyScene(LPDIRECT3DRMDEVICE dev, LPDIRECT3DRMVIEWPORT view,
    LPDIRECT3DRMFRAME lpScene, LPDIRECT3DRMFRAME lpCamera)
{
        HRESULT res;
    LPDIRECT3DRMFRAME lpLightframe1 = NULL;
    LPDIRECT3DRMFRAME lpLightframe2 = NULL;
    LPDIRECT3DRMLIGHT lpLight1      = NULL;
    LPDIRECT3DRMLIGHT lpLight2      = NULL;
    LPDIRECT3DRMLIGHT lpLight3      = NULL;
    LPDIRECT3DRMTEXTURE lpTex       = NULL;
    LPDIRECT3DRMWRAP lpWrap         = NULL;
    LPDIRECT3DRMMESHBUILDER lpbase_builder = NULL;

/*********  Para 8 elementos:  **************
    LPDIRECT3DRMMESHBUILDER lpObject_builder[elementos] = { NULL,NULL,NULL,NULL,

NULL,NULL,NULL,NULL};
// Para cambiar el número de elementos hay que cambiar el valor
// del "define elementos" en MainFrm.cpp y en MainFrm.h.
//               - Hay que modificar el número de elementos en la declaración
//                 de lpObject_builder (justo a continuación).
//               - En MakeMyMesh hay que modificar los elementos de
//                 char *ObjectName[elementos]. Y cuidado al escalar las
```

```
//                piezas y darles color.
//              - En SetMyPosition hemos de cambiar también la dimensión de
//                D3DVALUE Object_Position[elementos][3].

/*********  Para 11 elementos: ****************/
    LPDIRECT3DRMMESHBUILDER lpObject_builder[elementos] = { NULL,NULL,NULL,NULL,

NULL,NULL,NULL,NULL,NULL,NULL,NULL};
/*****************************************/

        // Creamos los distintos componentes de la escena
    MakeMyFrames(lpScene, lpCamera, &lpLightframe1, &lpLightframe2,
                &lpWorld_frame, lpObject_frame);
    MakeMyLights(lpScene, lpCamera, lpLightframe1, lpLightframe2,
                &lpLight1, &lpLight2, &lpLight3);
    SetMyPositions(lpScene, lpCamera, lpLightframe1, lpLightframe2,
                lpWorld_frame, lpObject_frame);
    MakeMyMesh(&lpbase_builder,lpObject_builder);
    MakeMyWrap(lpbase_builder, &lpWrap);
    AddMyTexture(lpbase_builder, &lpTex);

    // If you need to create a material (for example, to create
    // a shiny surface), call CreateMaterial and SetMaterial here.

    // Now that the visual object has been created, add it
    // to the world frame.

    res=lpWorld_frame->AddVisual((LPDIRECT3DRMVISUAL)lpbase_builder);
        if(res!=D3DRM_OK)
                AfxMessageBox("Error en AddVisual");

        // Hay que asociar el resto de objetos a cada frame
        // y con esto tendre el robot completo
    for(int i=0;i<elementos;i++)
        {
        res=lpObject_frame[i]->AddVisual((LPDIRECT3DRMVISUAL)lpObject_builder[i]);
        if(res!=D3DRM_OK)
                AfxMessageBox("Error en AddVisual");
        }

    lpLightframe1->Release();
    lpWorld_frame->Release();
    lpbase_builder->Release();
        for(i=0; i<elementos; i++)
                lpObject_builder[i]->Release();
    lpLight1->Release();
    lpLight2->Release();
    lpTex->Release();
    lpWrap->Release();

    return TRUE;
}

//////////////////////////////////////////////////////////////////////
// MakeMyFrames// Create frames used in the scene.//
```

```
//////////////////////////////////////////////////////////////////////
void CMainFrame::MakeMyFrames(LPDIRECT3DRMFRAME lpScene, LPDIRECT3DRMFRAME lpCamera,
    LPDIRECT3DRMFRAME * lplpLightFrame1, LPDIRECT3DRMFRAME * lplpLightFrame2,
    LPDIRECT3DRMFRAME * lplpWorld_frame, LPDIRECT3DRMFRAME * lplpObject_frame)
{
        HRESULT res;

        // Creamos dos frame para la iluminacion.
        res=lpD3DRM->CreateFrame(lpScene, lplpLightFrame1);
        if(res!=D3DRM_OK)
                AfxMessageBox("Error en CreateFrame");

        res=lpD3DRM->CreateFrame(lpScene, lplpLightFrame2);
        if(res!=D3DRM_OK)
                AfxMessageBox("Error en CreateFrame");

        // Creamos el frame global.
    res=lpD3DRM->CreateFrame(lpScene, lplpWorld_frame);
        if(res!=D3DRM_OK)
                AfxMessageBox("Error en CreateFrame");

        LPDIRECT3DRMFRAME lpParent=*lplpWorld_frame;
        // Creamos 11 Frames para elemento del robot.


        for(int i=0; i<elementos; i++)
        {
                // Creamos el frame
// Para la pieza a cortar, el padre el sistema world.
                if (i==(elementos-1))
                        lpParent=*lplpWorld_frame;

                res=lpD3DRM->CreateFrame(lpParent, &lplpObject_frame[i]);
                if(res!=D3DRM_OK)
                        AfxMessageBox("Error en CreateFrame de Objetos");

                // Creamos la jerarquia
                res=lpParent->AddChild(lplpObject_frame[i]);
                if(res!=D3DRM_OK)
                        AfxMessageBox("Error en AddChild de Objetos");

                lpParent=lplpObject_frame[i];
        }
}

//////////////////////////////////////////////////////////////////////
// MakeMyLights// Create lights used in the scene.//
//////////////////////////////////////////////////////////////////////
void CMainFrame::MakeMyLights(LPDIRECT3DRMFRAME lpScene, LPDIRECT3DRMFRAME lpCamera,
    LPDIRECT3DRMFRAME lpLightFrame1, LPDIRECT3DRMFRAME lpLightFrame2,
    LPDIRECT3DRMLIGHT * lplpLight1, LPDIRECT3DRMLIGHT * lplpLight2,
        LPDIRECT3DRMLIGHT * lplpLight3)
{
        HRESULT res;
    res=lpD3DRM->CreateLightRGB( D3DRMLIGHT_DIRECTIONAL,
```

```
        D3DVAL(0.7), D3DVAL(0.7), D3DVAL(0.7), lplpLight1);
    if(res!=D3DRM_OK)
                AfxMessageBox("Error en CreateLight");

    res=lpLightFrame1->AddLight(*lplpLight1);
        if(res!=D3DRM_OK)
                AfxMessageBox("Error en AddLight");

    res=lpD3DRM->CreateLightRGB(D3DRMLIGHT_AMBIENT,
        D3DVAL(0.5), D3DVAL(0.5), D3DVAL(0.5), lplpLight2);
        if(res!=D3DRM_OK)
                AfxMessageBox("Error en CreateFrame");

    res=lpD3DRM->CreateLightRGB( D3DRMLIGHT_DIRECTIONAL,
        D3DVAL(0.7), D3DVAL(0.7), D3DVAL(0.7), lplpLight3);
        if(res!=D3DRM_OK)
                AfxMessageBox("Error en CreateFrame");

    res=lpScene->AddLight(*lplpLight2);
        if(res!=D3DRM_OK)
                AfxMessageBox("Error en AddLight");

    res=lpLightFrame2->AddLight(*lplpLight3);
        if(res!=D3DRM_OK)
                AfxMessageBox("Error en AddLight");

}

//////////////////////////////////////////////////////////////////////
//
// SetMyPositions
// Set the positions and orientations of the light, camera, and
// world frames. Establish a rotation for the globe.
//
//////////////////////////////////////////////////////////////////////

void CMainFrame::SetMyPositions(LPDIRECT3DRMFRAME lpScene,
    LPDIRECT3DRMFRAME lpCamera, LPDIRECT3DRMFRAME lpLightFrame1,
    LPDIRECT3DRMFRAME lpLightFrame2, LPDIRECT3DRMFRAME lpWorld_frame,
        LPDIRECT3DRMFRAME * lplpObject_frame)
{
        HRESULT res;
        Vector PosicionCentroPieza;
        Vector pieza;
        PosicionCentroPieza=LeePuntoCentral();

//      Hay que escalar los valores
        pieza.x=PosicionCentroPieza.x-30;
        pieza.y=PosicionCentroPieza.y+300-10;
//      pieza.z=PosicionCentroPieza.z-250;
        pieza.z=PosicionCentroPieza.z+420-10;

        // Posiciones para los objetos.
        D3DVALUE Object_Position[elementos][3]=
        {{D3DVAL(0.0), D3DVAL(260.0), D3DVAL(0.0)},
```

```
        {D3DVAL(0.0), D3DVAL(80.0), D3DVAL(-155.0)},
        {D3DVAL(0.0), D3DVAL(690.0), D3DVAL(0.0)},
        {D3DVAL(-520.0), D3DVAL(0.0), D3DVAL(120.0)},
        {D3DVAL(-60.0), D3DVAL(2.0), D3DVAL(22.0)},
        {D3DVAL(-280.0), D3DVAL(0.0), D3DVAL(-20.0)},
        {D3DVAL(-2.0), D3DVAL(-87.0), D3DVAL(0.0)},
        {D3DVAL(0.0), D3DVAL(-20.0), D3DVAL(0.0)},
        {D3DVAL(0.0), D3DVAL(-25.0), D3DVAL(0.0)},
        {D3DVAL(0.0), D3DVAL(-70.0), D3DVAL(0.0)},
        {D3DVAL(-pieza.y), D3DVAL(pieza.z), D3DVAL(pieza.x)}};
//      {D3DVAL(eje -y), D3DVAL(eje +z), D3DVAL(eje +x)}};

        // Luces
        res=lpLightFrame1->SetPosition(lpScene,
       D3DVAL(2), D3DVAL(0.0), D3DVAL(22));
        if(res!=D3DRM_OK)
        AfxMessageBox("Error en SetPosition");
        res=lpLightFrame1->SetOrientation(lpScene,
                D3DVAL(0.0), D3DVAL(0.0), D3DVAL(1.0),
                D3DVAL(0.0), D3DVAL(1.0), D3DVAL(0.0));
        if(res!=D3DRM_OK)
                AfxMessageBox("Error en SetOrientation Luces");

        res=lpLightFrame1->SetPosition(lpScene,
        D3DVAL(2), D3DVAL(0.0), D3DVAL(-22));
        if(res!=D3DRM_OK)
                AfxMessageBox("Error en SetPosition Luces");
        res=lpLightFrame1->SetOrientation(lpScene,
                D3DVAL(0.0), D3DVAL(0.0), D3DVAL(-1.0),
                D3DVAL(0.0), D3DVAL(1.0), D3DVAL(0.0));
        if(res!=D3DRM_OK)
                AfxMessageBox("Error en SetOrientation Luces");

        // Camara
        res=lpCamera->SetPosition(lpScene,
        D3DVAL(0.0), D3DVAL(900.0), D3DVAL(3500.0));
//              D3DVAL(0.0), D3DVAL(900.0), D3DVAL(0.0));
//              D3DVAL(-880.0), D3DVAL(1050.0), D3DVAL(500.0));
//              D3DVAL(-650.0), D3DVAL(1100.0), D3DVAL(-500.0));

        if(res!=D3DRM_OK)
                AfxMessageBox("Error en SetPosition");

        Pant=0;
        Yant=0;

        res=lpCamera->SetOrientation(lpScene,
        D3DVAL(0.0), D3DVAL(0.0), D3DVAL(-1.0),
        D3DVAL(0.0), D3DVAL(1.0), D3DVAL(0.0));
        if(res!=D3DRM_OK)
                AfxMessageBox("Error en SetOrientation");

        // Sistema de referencia global base
        res=lpWorld_frame->SetPosition(lpScene,
        D3DVAL(0.0), D3DVAL(0.0), D3DVAL(0.0));
```

```
        if(res!=D3DRM_OK)
                AfxMessageBox("Error en SetPosition");

        res=lpWorld_frame->SetOrientation(lpScene,
        D3DVAL(0.0), D3DVAL(0.0), D3DVAL(1.0),
        D3DVAL(0.0), D3DVAL(1.0), D3DVAL(0.0));
        if(res!=D3DRM_OK)
                AfxMessageBox("Error en SetOrientation");

        res=lpWorld_frame->SetRotation(lpScene,
        D3DVAL(1.0), D3DVAL(1.1), D3DVAL(0.0), D3DVAL(0.05));
        if(res!=D3DRM_OK)
                AfxMessageBox("Error en SetRotation");

        // Hay que situar todos los sistemas de coordenadas o
        // bien manualmente o intentando cargar las transformciones
        // del archivo .X
        LPDIRECT3DRMFRAME lpParent=lpWorld_frame;

        for(int i=0;i<elementos;i++)
        {
                if (i==(elementos-1))
                        lpParent=lpWorld_frame;
                res=lplpObject_frame[i]->SetPosition(lpParent,
                        Object_Position[i][0], Object_Position[i][1], Object_Position[i][2]);
                if(res!=D3DRM_OK)
                        AfxMessageBox("Error en SetPosition");
                lpParent=lplpObject_frame[i];
        }

//      Hemos de rotar la pieza según la colocación de la pieza real.
//      Estos ángulos los calculamos a partir de las tres posiciones
//      iniciales que se introdujeron al ejecutar "corte3d.exe".

        Vector angulos;
        angulos=InclinacionPieza();

        double roll=angulos.x;
        double pitch=angulos.y;
        double yaw=angulos.z;

        CString msg;
        msg.Format("roll=%f, yaw=%f, pitch=%f",roll,yaw,pitch);
//      AfxMessageBox(msg);

        double Delta;
        Delta=180*PI/180;
        res=lpObject_frame[elementos-1]->SetRotation(lpWorld_frame,
(float)D3DVAL(1.0),(float)D3DVAL(0.0),(float)D3DVAL(0.0),(float)Delta);
        res=lpObject_frame[elementos-1]->Move(D3DVALUE(1.0));
        // Probamos con el yaw
        res=lpObject_frame[elementos-1]->SetRotation(lpWorld_frame,
(float)D3DVAL(0.0),(float)D3DVAL(1.0),(float)D3DVAL(0.0),(float)yaw);
        res=lpObject_frame[elementos-1]->Move(D3DVALUE(1.0));
        // Probamos con el roll
```

```
        res=lpObject_frame[elementos-1]->SetRotation(lpWorld_frame,
(float)D3DVAL(0.0),(float)D3DVAL(0.0),(float)D3DVAL(1.0),(float)-roll+8*PI/180);
        res=lpObject_frame[elementos-1]->Move(D3DVALUE(1.0));
        // Probamos con el pitch
        res=lpObject_frame[elementos-1]->SetRotation(lpWorld_frame,
(float)D3DVAL(1.0),(float)D3DVAL(0.0),(float)D3DVAL(0.0),(float)pitch);
        res=lpObject_frame[elementos-1]->Move(D3DVALUE(1.0));

        res=lplpObject_frame[5]->SetOrientation(lpScene,
                D3DVAL(0.0), D3DVAL(0.0), D3DVAL(1.0),
                D3DVAL(1.0), D3DVAL(0.0), D3DVAL(0.0));
}

//////////////////////////////////////////////////////////////////////
// MakeMyMesh// Create MeshBuilder object, load, scale, and color the mesh.//
//////////////////////////////////////////////////////////////////////
void CMainFrame::MakeMyMesh(LPDIRECT3DRMMESHBUILDER * lplpbase_builder,
                                                    LPDIRECT3DRMMESHBUILDER *
lplpObject_builder)
{
    HRESULT res;

        //CString strNombrePieza;

        char *aux;
        char NombrePieza[50];

//  Abrimos el archivo donde se encuentra el nombre
//      de la pieza que hemos de cortar.

        LeeNombrePieza(NombrePieza[50]);

        ifstream cin("nombrepieza.txt");
        cin >> NombrePieza;
        cin.close();
        aux=NombrePieza;

        char *ObjectName[elementos]={"Solevado02","Caja01","Solevado03",
                        "Solevado04","Solevado08","Cilindro01",
         "Cilindro03","Cilindro04","Cilindro05",
"Cilindro06",aux};
        // Creamos la base
        res=lpD3DRM->CreateMeshBuilder(lplpbase_builder);
        if(res!=D3DRM_OK)
                AfxMessageBox("Error en CreateMeshBuilder");
        res=(*lplpbase_builder)->
                Load("Rx90.x", "Solevado01", D3DRMLOAD_FROMFILE | D3DRMLOAD_BYNAME  , NULL, NULL);
        if(res!=D3DRM_OK)
                AfxMessageBox("Error en LoadMesh pie");
        if(res==D3DRMERR_NOTFOUND )
                AfxMessageBox("Error en MeshNotFound");
        res=(*lplpbase_builder)->
                Scale(D3DVAL(1), D3DVAL(1), D3DVAL(1));
        if(res!=D3DRM_OK)
                AfxMessageBox("Error en Scale .x");
```

```
    // Set sphere to white to avoid unexpected texture-blending results.
        res=(*lplpbase_builder)->
                SetColorRGB(D3DVAL(10.0), D3DVAL(50.0), D3DVAL(0.5));
        if(res!=D3DRM_OK)
                AfxMessageBox("Error en SetColor .x");

        ofstream cout("nombrepieza.txt");
        cout << NombrePieza << ".x";
        cout.close();

        char NombreArchivo[50];
        char *NameFile;

        ifstream cin2("nombrepieza.txt");
        cin2 >> NombreArchivo;
        cin2.close();

        NameFile=NombreArchivo;

        // Hay que cargar el resto de objetos mesh.
        for(int i=0; i<elementos; i++)
        {
                //Carga de los demás elementos
                res=lpD3DRM->CreateMeshBuilder(&lplpObject_builder[i]);
                if(res!=D3DRM_OK)
                        AfxMessageBox("Error en CreateMeshBuilder");
                if (i==(elementos-1))
                        res=(lplpObject_builder[i])->
                                Load(NameFile, ObjectName[i], D3DRMLOAD_FROMFILE | D3DRMLOAD_BYNAME
                                , NULL, NULL);
                else
                        res=(lplpObject_builder[i])->
                                Load("Rx90.x", ObjectName[i], D3DRMLOAD_FROMFILE | D3DRMLOAD_BYNAME
                                , NULL, NULL);
                        if(res!=D3DRM_OK)
                                AfxMessageBox("Error en LoadMesh .x");
                        if(res==D3DRMERR_NOTFOUND )
                                AfxMessageBox("Error en MeshNotFound");
//      Dimensiones de la fresa:
                        if (i==5 || i==6 || i==7)
                                res=(lplpObject_builder[i])->
                                        Scale(D3DVAL(0.7), D3DVAL(0.7), D3DVAL(1.0));
                        else if (i==9)
                                res=(lplpObject_builder[i])->
                                        Scale(D3DVAL(1), D3DVAL(2.5), D3DVAL(1));
                        else if (i==(elementos-1))
                                res=(lplpObject_builder[i])->
                                        Scale(D3DVAL(1.4), D3DVAL(1), D3DVAL(1.4));
                        else
                                res=(lplpObject_builder[i])->
                                        Scale(D3DVAL(1), D3DVAL(1), D3DVAL(1));

                        if(res!=D3DRM_OK)
                                AfxMessageBox("Error en Scale .x");
```

```
//                      if (i==5 || i==6 || i==7)
//                              res(lplpObject_builder[i]->
//                                      SetRotation(lpScene,
D3DVAL(0.0),D3DVAL(0.0),D3DVAL(1.0),D3DVAL(1.0)));
// Lo anterior no funciona porque los objetos "builder" no pueden
// utilizar métodos "frame", "SetRotation" es una función de método
// "frame"

                //Darle color
                if(i==4 || i==5)
                {
                    res=(lplpObject_builder[i])->
                                SetColorRGB(D3DVAL(0.0), D3DVAL(100.0), D3DVAL(50.0));
                        if(res!=D3DRM_OK)
                                AfxMessageBox("Error en SetColor .x");
                }
                else if (i<4)
                {
                        res=(lplpObject_builder[i])->
                                SetColorRGB(D3DVAL(10.0), D3DVAL(50.0), D3DVAL(0.50));
                        if(res!=D3DRM_OK)
                                AfxMessageBox("Error en SetColor .x");
                }
                else if (i==10)
                {
                        res=(lplpObject_builder[i])->
                                SetColorRGB(D3DVAL(255.0), D3DVAL(255.0), D3DVAL(255.0));
                        if(res!=D3DRM_OK)
                                AfxMessageBox("Error en SetColor .x");
                }
                else
                {
                        res=(lplpObject_builder[i])->
                                SetColorRGB(D3DVAL(20.5), D3DVAL(0.5), D3DVAL(0.5));
                        if(res!=D3DRM_OK)
                                AfxMessageBox("Error en SetColor .x");
                }
        }
}

//////////////////////////////////////////////////////////////////////
//
// MakeMyWrap
// Creates and applies wrap for texture.
//
//////////////////////////////////////////////////////////////////////

void CMainFrame::MakeMyWrap(LPDIRECT3DRMMESHBUILDER base_builder,
                LPDIRECT3DRMWRAP * lpWrap)
{
    HRESULT res;
        D3DVALUE miny, maxy, height;
    D3DRMBOX box;
```

```
// Guardamos en box los parámetros de la base del robot.

    base_builder->GetBox(&box);

    maxy = box.max.y;
    miny = box.min.y;
    height = maxy - miny;

    res = lpD3DRM->CreateWrap
        (D3DRMWRAP_CYLINDER, NULL,
        D3DVAL(0.0), D3DVAL(0.0), D3DVAL(0.0),
        D3DVAL(0.0), D3DVAL(1.0), D3DVAL(0.0),
        D3DVAL(0.0), D3DVAL(0.0), D3DVAL(1.0),
        D3DVAL(0.0), D3DDivide(miny, height),
        D3DVAL(1.0), D3DDivide(-D3DVAL(1.0), height),
        lpWrap);
        if(res!=D3DRM_OK)
                AfxMessageBox("Error en CreateWrap");

        res = (*lpWrap)->Apply((LPDIRECT3DRMOBJECT)base_builder);
        if(res!=D3DRM_OK)
                AfxMessageBox("Error en ApplyCreateWrap");
}

////////////////////////////////////////////////////////////////////
//
// AddMyTexture
// Creates and applies wrap for texture.
//
////////////////////////////////////////////////////////////////////

void CMainFrame::AddMyTexture(LPDIRECT3DRMMESHBUILDER lpbase_builder,
    LPDIRECT3DRMTEXTURE * lplpTex)
{
    HRESULT res = lpD3DRM->LoadTexture("Sad.bmp", lplpTex);
        if(res!=D3DRM_OK)
                AfxMessageBox("Error en LoadTexture");
    // If you need a color depth other than the default (16),
    // call IDirect3DRMTexture::SetShades here.

        res = lpbase_builder->SetTexture(*lplpTex);
//      if(res!=D3DRM_OK)
//              AfxMessageBox("Error en SetTexture");
}

////////////////////////////////////////////////////////////////////
// CreateDevAndView
// Create the D3DRM device and viewport with the given D3D driver and
// with the specified size.//
////////////////////////////////////////////////////////////////////
BOOL CMainFrame:: CreateDevAndView(LPDIRECTDRAWCLIPPER lpDDClipper, int driver,
    int width, int height)
{
        HRESULT rval;
    // Create the D3DRM device from this window by using the specified
```

```
    // D3D driver.
    rval=lpD3DRM->CreateDeviceFromClipper( lpDDClipper,
        &myglobs.DriverGUID[driver], width, height, &myglobs.dev);
    if(rval!=D3DRM_OK)
                AfxMessageBox("Error en CreateDeviceFromClipper");

         // Create the D3DRM viewport by using the camera frame.  Set the
    // background depth to a large number. The width and height
    // might have been slightly adjusted, so get them from the device.
    width = myglobs.dev->GetWidth();
    height = myglobs.dev->GetHeight();
    rval = lpD3DRM->CreateViewport( myglobs.dev,
        myglobs.camera, 0, 0, width, height, &myglobs.view);
    if (rval != D3DRM_OK)
         {
                 myglobs.dev->Release();
         return FALSE;
         }
    rval = myglobs.view->SetBack( D3DVAL(5000.0));
    if (rval != D3DRM_OK)
         {
                 myglobs.dev->Release();
         myglobs.view->Release();
                 return FALSE;
         }

    // Set the render quality, fill mode, lighting state,
    // and color shade info.
         if (!SetRenderState())
                 return FALSE;
    return TRUE;
}

//////////////////////////////////////////////////////////////////////
//
// BPPToDDBD
// Converts bits-per-pixel to a DirectDraw bit-depth flag.
//
//////////////////////////////////////////////////////////////////////

static DWORD
BPPToDDBD(int bpp)
{
    switch(bpp) {
        case 1:
            return DDBD_1;
        case 2:
            return DDBD_2;
        case 4:
            return DDBD_4;
        case 8:
            return DDBD_8;
        case 16:
            return DDBD_16;
        case 24:
```

```
            return DDBD_24;
        case 32:
            return DDBD_32;
        default:
            return 0;
    }
}

//////////////////////////////////////////////////////////////////////
// enumDeviceFunc
// Callback function that records each usable D3D driver's name
// and GUID. Chooses a driver and sets *lpContext to this driver.//
//////////////////////////////////////////////////////////////////////

HRESULT FAR PASCAL enumDeviceFunc(LPGUID lpGuid, LPSTR lpDeviceDescription,
    LPSTR lpDeviceName, LPD3DDEVICEDESC lpHWDesc,
    LPD3DDEVICEDESC lpHELDesc, LPVOID lpContext)
{
   static BOOL hardware = FALSE; // Current start driver is hardware
   static BOOL mono = FALSE;     // Current start driver is mono light
   LPD3DDEVICEDESC lpDesc;
   int *lpStartDriver = (int *)lpContext;

   // Decide which device description should be consulted.
   lpDesc = lpHWDesc->dcmColorModel ? lpHWDesc : lpHELDesc;

   // If this driver cannot render in the current display bit-depth,
   // skip it and continue with the enumeration.

   if (!(lpDesc->dwDeviceRenderBitDepth & BPPToDDBD(myglobs.BPP)))
      return D3DENUMRET_OK;

   // Record this driver's name and GUID.
   memcpy(&myglobs.DriverGUID[myglobs.NumDrivers],lpGuid,sizeof(GUID));
   lstrcpy(&myglobs.DriverName[myglobs.NumDrivers][0],lpDeviceName);

   // Choose hardware over software, RGB lights over mono lights.
   if (*lpStartDriver == -1)
   {
   // This is the first valid driver.
        *lpStartDriver = myglobs.NumDrivers;
        hardware = lpDesc == lpHWDesc ? TRUE : FALSE;
        mono = lpDesc->dcmColorModel & D3DCOLOR_MONO ? TRUE : FALSE;
   }
   else if (lpDesc == lpHWDesc && !hardware)
   {
        // This driver is hardware and the start driver is not.
        *lpStartDriver = myglobs.NumDrivers;
        hardware = lpDesc == lpHWDesc ? TRUE : FALSE;
        mono = lpDesc->dcmColorModel & D3DCOLOR_MONO ? TRUE : FALSE;
   }
   else if ((lpDesc == lpHWDesc && hardware ) ||
           (lpDesc == lpHELDesc && !hardware))
   {
        if (lpDesc->dcmColorModel == D3DCOLOR_MONO && !mono)
```

```
                {
            // This driver and the start driver are the same type, and
            // this driver is mono whereas the start driver is not.
            *lpStartDriver = myglobs.NumDrivers;
            hardware = lpDesc == lpHWDesc ? TRUE : FALSE;
            mono = lpDesc->dcmColorModel & D3DCOLOR_MONO ? TRUE : FALSE;
        }
    }
    myglobs.NumDrivers++;
    if (myglobs.NumDrivers == MAX_DRIVERS)
            return (D3DENUMRET_CANCEL);
    return (D3DENUMRET_OK);
}

BOOL CMainFrame::EnumDrivers(void)
{
        LPDIRECT3D lpD3D;
        HRESULT rval;

    // Create a DirectDraw object and query for the Direct3D interface
    // to use to enumerate the drivers.
        rval = DirectDrawCreate(NULL, &lpDD, NULL);
        if (rval != DD_OK)
        {
                AfxMessageBox("No se pudo crear objeto DirectDraw");
                return FALSE;
        }

        rval = lpDD->QueryInterface( IID_IDirect3D, (void**) &lpD3D);
        if (rval != DD_OK)
        {
        lpDD->Release();
        return FALSE;
        }

        lpD3D->EnumDevices( enumDeviceFunc,&myglobs.CurrDriver);
    // Ensure at least one valid driver was found.
        if (myglobs.NumDrivers == 0)
        {
                return FALSE;
        }

        lpD3D->Release();
        lpDD->Release();

        return TRUE;
}

BOOL CMainFrame::Render()
{
        HRESULT rval;

    // Clear the viewport.
        rval = myglobs.view->Clear();
        if (rval != D3DRM_OK)
```

```
        {
        return FALSE;
        }

        // Render the scene to the viewport.
        rval = myglobs.view->Render(myglobs.scene);
        if (rval != D3DRM_OK)
        {
                return FALSE;
        }

    // Update the window.
        rval = myglobs.dev->Update();
        if (rval != D3DRM_OK)
        {
        return FALSE;
        }
        return TRUE;
}

void CMainFrame::OnPaint()
{
        CPaintDC dc(this); // device context for painting

        if(bInitialized == TRUE)
        {
                if(myglobs.dev)
                {
                        LPDIRECT3DRMWINDEVICE windev;
                        PAINTSTRUCT ps;
                        BeginPaint(&ps);
                        if(myglobs.dev-
>QueryInterface(IID_IDirect3DRMWinDevice,(void**)&windev)==0)
                        {
                                if(windev->HandlePaint(ps.hdc)!=0)
                                        AfxMessageBox("HandlePaint failed");
                                windev->Release();
                        }
                        else
                                AfxMessageBox("failed to handle WM_PAINT");
                        EndPaint(&ps);
                }
        }
        // Do not call CFrameWnd::OnPaint() for painting messages
}

BOOL CMainFrame::OnEraseBkgnd(CDC* pDC)
{
        return CFrameWnd::OnEraseBkgnd(pDC);
}

void CMainFrame::OnActivate(UINT nState, CWnd* pWndOther, BOOL bMinimized)
{
        CFrameWnd::OnActivate(nState, pWndOther, bMinimized);
```

```
        LPDIRECT3DRMWINDEVICE lpD3DRMWinDev;
        if (!myglobs.dev)
                return;

        myglobs.dev->QueryInterface(IID_IDirect3DRMWinDevice,
                                    (void **) &lpD3DRMWinDev);
        lpD3DRMWinDev->HandleActivate((WORD) nState);
        lpD3DRMWinDev->Release();
}

COLORREF CMainFrame::D3DCOLOR_2_COLORREF(D3DCOLOR d3dclr)
{
        D3DVALUE r=D3DVALUE(255)*D3DRMColorGetRed(d3dclr);
        D3DVALUE g=D3DVALUE(255)*D3DRMColorGetGreen(d3dclr);
        D3DVALUE b=D3DVALUE(255)*D3DRMColorGetBlue(d3dclr);
        return RGB((int)r,(int)g,(int)b);
}

BOOL CMainFrame::SetMode()
{
        HRESULT hr;

        // Set DDSCL_NORMAL to use windowed mode
        hr = lpDD->SetCooperativeLevel(AfxGetMainWnd()->GetSafeHwnd(),
        DDSCL_NORMAL);
        if(hr != DD_OK)
        {
                AfxMessageBox("Error en SetCooperativeLevel");
                return FALSE;
        }

        return TRUE;
}
GUID* CMainFrame::GetGUID()
{
        static GUID* lpguid;
        HRESULT r;

        D3DFINDDEVICESEARCH searchdata;
        memset(&searchdata,0,sizeof(searchdata));
        searchdata.dwSize=sizeof(searchdata);
        searchdata.dwFlags=D3DFDS_COLORMODEL;
        searchdata.dcmColorModel=D3DCOLOR_MONO;

        static D3DFINDDEVICERESULT resultdata;
        memset(&resultdata,0,sizeof(resultdata));
        resultdata.dwSize=sizeof(resultdata);

        LPDIRECTDRAW  ddraw;
        r=DirectDrawCreate(NULL,&ddraw,NULL);
        if(r!=DD_OK)
        {
                TRACE("directdarwcreate failed \n");
                return NULL;
        }
```

```
	LPDIRECT3D d3d;
	r=ddraw->QueryInterface(IID_IDirect3D,(void**)&d3d);
	if(r!=D3DRM_OK)
	{
		TRACE("queryinterface failed \n");
		ddraw->Release();
		return NULL;
	}

	r=d3d->FindDevice(&searchdata,&resultdata);
	if(r==D3D_OK)lpguid=&resultdata.guid;
	else
	{
		TRACE("finddevice failed \n");
		lpguid=NULL;
	}

	d3d->Release();
	ddraw->Release();

	return lpguid;
}

BOOL CMainFrame::RenderLoop()
{
	return FALSE;
}

void CMainFrame::OnDestroy()
{
	CFrameWnd::OnDestroy();

	// Limpiamos todo lo creado
	CleanUp();
}

void CMainFrame::CleanUp()
{
	myglobs.scene->Release();
	myglobs.camera->Release();
	myglobs.view->Release();
	myglobs.dev->Release();
	lpD3DRM->Release();
	lpDDClipper->Release();

    bInitialized = FALSE;
}

//////////////////////////////////////////////////////////////////
//                                                              //
//      Función que desplaza un eje determinado un valor angular //
//      en el sentido de giro de dicho eje.                     //
//      Parámetros:                                             //
//              - double pos: Variable que contiene el nuevo valor //
```

```
//                angular que debe tener un eje.                          //
//              - int eje: Variable que indica el eje cuya variable       //
//                angular debe ser modificada.                            //
//                                                                        //
////////////////////////////////////////////////////////////////////////////

void CMainFrame::SetPosition(double pos, int eje)
{
        HRESULT res;
        double Delta;
        LPDIRECT3DRMFRAME lpParent;

//      En realidad sólo vale definir la orientacion para
//      los seis primeros elementos, que son las articulaciones
//      del robot, pero hay que definir el mismo número
//      de ejes que de objetos mesh que se cargan.

        D3DVALUE Orientacion_Ejes[elementos][3]=
        {{D3DVAL(0.0),    D3DVAL(-1.0),     D3DVAL(0.0)},
        {D3DVAL(0.0),     D3DVAL(0.0),     D3DVAL(1.0)},
        {D3DVAL(0.0),     D3DVAL(0.0),     D3DVAL(1.0)},
        {D3DVAL(1.0),      D3DVAL(0.0),     D3DVAL(0.0)},
        {D3DVAL(0.0),     D3DVAL(0.0),     D3DVAL(1.0)},
        {D3DVAL(1.0),      D3DVAL(0.0),      D3DVAL(0.0)},
        {D3DVAL(0.0),     D3DVAL(0.0),     D3DVAL(1.0)},
        {D3DVAL(0.0),     D3DVAL(0.0),     D3DVAL(1.0)},
        {D3DVAL(0.0),      D3DVAL(0.0),      D3DVAL(1.0)},
        {D3DVAL(0.0),     D3DVAL(0.0),     D3DVAL(1.0)},
        {D3DVAL(0.0),     D3DVAL(0.0),     D3DVAL(1.0)}};

        Delta=pos-m_pRobot->Posicionact.Artic[eje];

//  El siguiente "if" se usa para los casos en que hay un
//      cambio brusco en un ángulo. Suele suceder en la cuarta
//      articulación, que cambia de -170 a 170 (por ejemplo)
//      de forma que aunque en la realidad son 20° de diferencia
//      el robot virtual recorre los 340° y se ve un salto muy raro.
//      Se multiplica por (eje+1) porque para cada eje hay que escalar
//      los ángulos (ver la función where de la clase captura).
//      Para ver como funciona ponemos un ejemplo:
//                      pos=174;
//                      Posicionact.Artic[eje]=-167;
//                      Delta=174-(-167)=341; // que es igual a -19°;
//      Si (Delta > 330)
//              Delta =-(360 - Delta) = -19;
        if ((Delta*(eje+1))>310*PI/180)
                Delta= -(2*PI/(eje+1)-Delta);
        else if ((Delta*(eje+1))<-310*PI/180)
                Delta= 2*PI/(eje+1)+Delta;

//      Siguiendo con el ejemplo anterior, el valor de
//      Posicionact.Artic[eje] sí lo cambiamos a 174.

        m_pRobot->Posicionact.Artic[eje]=pos;
```

```
//      Pero a la hora de hacer la rotación gráfica, sólo se rota
//      un salto Delta = -19.

        if (eje==0)
                lpParent=lpWorld_frame;
        else
                lpParent=lpObject_frame[eje-1];
        res=lpObject_frame[eje]->SetRotation(lpParent,
(float)Orientacion_Ejes[eje][0],(float)Orientacion_Ejes[eje][1],(float)Orientacion_Ejes[eje][2],(floa
t)Delta);
        res=lpObject_frame[eje]->Move(D3DVALUE(1.0));
}

////////////////////////////////////////////////////////////////////
//                                                                //
//      Función que ubica la cámara según dos ángulos de rotación,  //
//      uno horizontal y otro vertical que recibe como parámetros.  //
//                                                                //
////////////////////////////////////////////////////////////////////

void CMainFrame::SetCamera(double pitch, double yaw)
{
        HRESULT res;

        // Camara
        res=myglobs.camera->SetOrientation(myglobs.scene,
                D3DVAL(-cos(pitch)*sin(yaw)), D3DVAL(sin(pitch)),
                D3DVAL(cos(pitch)*cos(yaw)), D3DVAL(sin(pitch)*sin(yaw)),
                D3DVAL(cos(pitch)), D3DVAL(-sin(pitch)*cos(yaw)));
        if(res!=D3DRM_OK)
                AfxMessageBox("Error en SetOrientation");

        res=myglobs.camera->SetPosition(myglobs.scene,
        D3DVAL(3500*sin(yaw)*cos(pitch)), D3DVAL(900.0*(1.0-
                4.5*sin(pitch))), D3DVAL(-3500.0*cos(yaw)*cos(pitch)));
        if(res!=D3DRM_OK)
                AfxMessageBox("Error en SetPosition");
}

////////////////////////////////////////////////////////////////////
//                                                              //
//      Función que salta cada vez que lo indica el temporizador,  //
//      llamando a la función que se ocupa de leer cada punto e    //
//      imponerlo en el robot.                                  //
//                                                              //
////////////////////////////////////////////////////////////////////

void CMainFrame::OnTimer(UINT nIDEvent)
{
        switch(nIDEvent)
        {
        case 1:
                //Obtención de las posiciones.
                Cp.where();
                if (Cp.iteracion>=Cp.TotalPuntosAPT)
```

```
                        OnMovimientoStop();
                break;
        }
                CFrameWnd::OnTimer(nIDEvent);
}

////////////////////////////////////////////////////////////////////////
//                                                                    //
//      Función que salta cuando se pulsa en el menú "ver punto de    //
//      vista". Su finalidad es abrir el cuadro de diálogo del punto  //
//      de vista.                                                     //
//                                                                    //
////////////////////////////////////////////////////////////////////////

void CMainFrame::OnVerPuntodevista()
{
        //Por compartir la función SetPosition.
        //Solo paramos la representación.
        KillTimer(1);

        //Creamos la instancia de la ventana.
        pDlgCamara=new CDlgCamara(this);
        pDlgCamara->m_pFrame=this;
        //La creamos.
        pDlgCamara->Create(IDD_DIALOGCAMARA,this);
        pDlgCamara->ShowWindow(TRUE);
}

////////////////////////////////////////////////////////////////////
//                                                                //
//      Función que llama a la clase CCaptura para comenzar la    //
//      captura de datos para la representación.                  //
//      Se activa también el temporizador.                        //
//                                                                //
////////////////////////////////////////////////////////////////////

void CMainFrame::OnMovimientoStart()
{
        if (move==0)
        {
                Cp.Capture();
                Cp.iteracion=0;
                if(Cp.func==1)
                {
                        move = 1;
                        Sleep(100);
                        SetTimer(1,400,NULL);
                }
        }
}

////////////////////////////////////////////////////////////////////
//                                                                //
//      Método que detiene el temporizador, y con él el movimiento //
//      del robot.                                                //
```

```
//                                                                    //
////////////////////////////////////////////////////////////////////////

void CMainFrame::OnMovimientoStop()
{
        KillTimer(1);
        if(move==1)
        {
                Sleep(200);
        }
        move = 0;
}

////////////////////////////////////////////////////////////////////////
//                                                                    //
//      Método que detiene el temporizador y, por tanto, el           //
//      movimiento del robot. Además devuelve el robot a su posición  //
//      inicial.                                                      //
//                                                                    //
////////////////////////////////////////////////////////////////////////
void CMainFrame::OnMovimientoReset()
{
        KillTimer(1);

        if(move==1)
        {
                Sleep(200);
        }

        //Volvemos a la posición original.
        //Dos veces.
        //mhand=0;
        move = 0;
        for(int j=0;j<2;j++)
        {
                for (int i=0;i<6;i++)
                        SetPosition(0.0,i);
        }
}

////////////////////////////////////////////////////////////////////
//                                                                //
//      Función que activa de nuevo el temporizador después de que //
//      se hubiese detenido para abrir el cuadro de diálogo "Ver  //
//      Punto de Vista".                                          //
//                                                                //
////////////////////////////////////////////////////////////////////

void CMainFrame::CamaraStart()
{
//Si se estaba moviendo volver a representar.
        if (move==1)
        {
                SetTimer(1,400,NULL);
        }
```

```
}


//////////////////////////////////////////////////////////////////////
//                                                                  //
//      Función que abre el cuadro de diálogo "Movimiento manual    //
//      articular". Detiene el temporizador si el robot se estaba   //
//      moviendo.                                                   //
//                                                                  //
//////////////////////////////////////////////////////////////////////

void CMainFrame::OnMovimientoManualArticular()
{
        // TODO: Add your command handler code here
        KillTimer(1);

        pDlgCoordArt=new CDlgCoordArt(this);
        pDlgCoordArt->m_pFrame=this;
        //La creamos.
        pDlgCoordArt->Create(IDD_DIALOGANGULO,this);
        pDlgCoordArt->ShowWindow(TRUE);
}

//////////////////////////////////////////////////////////////////////
//                                                                  //
//      Función que abre el cuadro de diálogo "Movimiento manual    //
//      cartesiano". Detiene el temporizador si el robot se estaba  //
//      moviendo.                                                   //
//                                                                  //
//////////////////////////////////////////////////////////////////////

void CMainFrame::OnMovimientoManualCartesiana()
{
        // TODO: Add your command handler code here
        KillTimer(1);

        pDlgCoordCart=new CDlgCoordCart(this);
        pDlgCoordCart->m_pFrame=this;

        //La creamos.
        pDlgCoordCart->Create(IDD_DIALOGPOS,this);
        pDlgCoordCart->ShowWindow(TRUE);
}

//////////////////////////////////////////////////////////////////////
//                                                                  //
//      Función que lee del archivo "nobrepieza.txt" el nombre de   //
//      la pieza que cargó el usuario en la aplicación 'Corte3D'.   //
//                                                                  //
//////////////////////////////////////////////////////////////////////

void CMainFrame::LeeNombrePieza(char Nombre)
{
        CString strContPieza, strBuffer, strNombrePieza;
```

```
        BOOL fRes=TRUE;
// Hemos de saber qué pieza se ha cargado en "corte3d.exe"
// para cargarla aquí.
// Abrimos el fichero "nombrepieza.txt"

        CFile file;
        try
        {
                file.Open("nombrepieza.txt", CFile::modeRead);
        }
        catch (CFileException *e)
        {
                CString strError;
                strError.Format("Error al abrir el archivo. Error número %d",e->m_cause);
                e->Delete();
                AfxMessageBox(strError);
                fRes=FALSE;
        }

        if (fRes)
        {
                int nFileLength = file.GetLength();
                char *lpBuffer = new char[nFileLength +1];
// Leemos todo el contenido del buffer del puerto serie y lo almacenamos en nuestro buffer temporal
// Cuando quitamos el fichero que ha de leer debería dar
// un error y saltar el "catch", pero no lo hace
// NO LO ENTIENDO.
                try
                {
                        file.Read(lpBuffer,nFileLength);
                }
                catch (CFileException *e)
                {
                        CString strError;
                        strError.Format("Error al leer el archivo. Error número %d",e->m_cause);
                        AfxMessageBox(strError);
                        e->Delete();
                        delete[] lpBuffer;
                        fRes=FALSE;
                }

                // Añadimos un carácter NULL para terminar la cadena.
                lpBuffer[nFileLength] = '\0';


                // Cerramos el fichero.
                file.Close();

                // Almacenamos el contenido del fichero en una instancia de la clase CString, más
fácil de usar.
                strContPieza.Format(_T("%s"),lpBuffer);

                // Vaciamos y eliminamos el buffer temporal.
                delete[] lpBuffer;
                int nLongitudLinea;
```

```
                nLongitudLinea = strContPieza.Find(_T('_'));
                for (int j=0; j<nLongitudLinea;j++)
                        strBuffer += strContPieza[j];

                strNombrePieza = strBuffer;
                strBuffer.Empty();
        }

// Reescribimos en el mismo fichero sólo la parte del nombre que
// nos interesa.

        ofstream cout("nombrepieza.txt");
        cout << strNombrePieza;
        cout.close();
}

//////////////////////////////////////////////////////////////////
//                                                              //
//      Función que lee los puntos que se almacenaron en el fichero     //
//      "puntocentral.txt", que son los tres puntos iniciales del       //
//      stock y el punto central de la trayectoria.             //
//                                                              //
//////////////////////////////////////////////////////////////////

Vector CMainFrame::LeePuntoCentral()
{
        Vector PuntoCentral;
//      Vector Punto[3];
        double dPuntoCentral[3];
        bool fRes=TRUE;

        PuntoCentral.x=0;
        PuntoCentral.y=0;
        PuntoCentral.z=0;

        CFile file;
        try
        {
                file.Open("puntocentral.txt", CFile::modeRead);
        }
        catch (CFileException *e)
        {
                CString strError;
                strError.Format("Error al abrir el archivo. Error número %d",e->m_cause);
                e->Delete();
                AfxMessageBox(strError);
                fRes=FALSE;
        }

        if (fRes)
        {
                int nFileLength = file.GetLength();
                char *lpBuffer = new char[nFileLength +1];
// Leemos todo el contenido del buffer del puerto serie y lo almacenamos en nuestro buffer temporal
```

```
// Cuando quitamos el fichero que ha de leer debería dar
// un error y saltar el "catch", pero no lo hace
// NO LO ENTIENDO.
                try
                {
                        file.Read(lpBuffer,nFileLength);
                }
                catch (CFileException *e)
                {
                        CString strError;
                        strError.Format("Error al leer el archivo. Error número %d",e->m_cause);
                        AfxMessageBox(strError);
                        e->Delete();
                        delete[] lpBuffer;
                        fRes=FALSE;
                }

                // Añadimos un carácter NULL para terminar la cadena.
                lpBuffer[nFileLength] = '\0';

                // Cerramos el fichero.
                file.Close();

                CString strContFichero, strBuffer;

                // Almacenamos el contenido del fichero en una instancia de la
                // clase CString, más fácil de usar.
                strContFichero.Format(_T("%s"),lpBuffer);

                // Vaciamos y eliminamos el buffer temporal.
                delete[] lpBuffer;

                int nIndex, nLongitud;

                for (int i=0; i<4; i++)
                {
                        nLongitud = strContFichero.Find(_T('\n'));

                        int nIndex2=0;
                        for (nIndex = 0; nIndex < nLongitud; nIndex++)
                        {

                // Recorremos la línea y vamos almacenado las coordenadas
                // en un array double[3].

                                if ((strContFichero[nIndex] != ' ') && (strContFichero[nIndex] !=
'\r'))
                                        strBuffer += strContFichero[nIndex];
                                else
                                {
                                // Estamos en un caracter delimitador (bien el
                                // caracter ' ' o bien '\r'). Por tanto, lo que
                                // tenemos hasta ahora es una coordenada completa
                                // y debemos convertirla en un double y pasarla
                                // al array temporal.
```

```
                                        dPuntoCentral[nIndex2] = _tcstod(strBuffer, '\0');
                                        strBuffer.Empty();
                                        nIndex2++;

                                }
                        }
                        if (i==3)
                        {
                                PuntoCentral.x=dPuntoCentral[0];
                                PuntoCentral.y=dPuntoCentral[1];
                                PuntoCentral.z=dPuntoCentral[2];
                        }
                        else
                        {
// Puntos iniciales del stock
                                Punto[i].x=dPuntoCentral[0];
                                Punto[i].y=dPuntoCentral[1];
                                Punto[i].z=dPuntoCentral[2];
                        }
                        strContFichero.Delete(0, nLongitud + 1);
                }
        }
        return PuntoCentral;
}

//////////////////////////////////////////////////////////////////////
//                                                                  //
//      Función que calcula la inclinación en el espacio del plano  //
//      constituido por los tres puntos iniciales leídos del robot  //
//      en la aplicación 'Corte3D'.                                 //
//                                                                  //
//////////////////////////////////////////////////////////////////////

Vector CMainFrame::InclinacionPieza()
{
        double modulo;
        double roll, yaw, pitch;

        Vector V12, Veuler;

        V12.x = Punto[1].x-Punto[2].x;
        V12.y = Punto[1].y-Punto[2].y;
        V12.z = Punto[1].z-Punto[2].z;

        modulo=sqrt(V12.x*V12.x+V12.y*V12.y+V12.z*V12.z);
//      roll=0;
        roll=acos(V12.z/modulo);
//      yaw=0;
        yaw=acos(V12.x/modulo);
        pitch=0;

        Veuler.x=roll;
        Veuler.y=pitch;
        Veuler.z=yaw;
```

```
        return Veuler;
}
```

B.2.5. CRobot.

B.2.5.1. Robot.h

```
//////////////////////////////////////////////////////////////
//                                                          //
//      Robot.h: Definición de la clase CRobot.             //
//                                                          //
//      Creado por F. Javier Rueda 2001                     //
//                                                          //
//////////////////////////////////////////////////////////////

#if !defined(AFX_ROBOT_H__6EF5083E_838B_11D1_B61B_444553540000__INCLUDED_)
#define AFX_ROBOT_H__6EF5083E_838B_11D1_B61B_444553540000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "Estructura.h"

class CRobot
{
public:
        CRobot();
        virtual ~CRobot();
        Rx90Pos Posicionact;
};

#endif // !defined(AFX_ROBOT_H__6EF5083E_838B_11D1_B61B_444553540000__INCLUDED_)
```

B.2.5.2. Robot.cpp

```
//////////////////////////////////////////////////////////////
//                                                          //
// Robot.cpp: archivo que implementa la clase CRobot.       //
//                                                          //
//////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Rx90.h"
#include "Robot.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
```

```
//////////////////////////////////////////////////////////////////////

CRobot::CRobot()
{
        for(int i=0;i<6;i++)
                Posicionact.Artic[i]=0.0;
}

CRobot::~CRobot()
{
}
```

B.2.6. CDlgCamara.

B.2.6.1. DlgCamara.h

```
//////////////////////////////////////////////////////////////////////
//                                                                  //
//      DlgCamara.h: Definición de la clase CDlgCamara.             //
//                                                                  //
//      Clase manejadora de los ángulos de rotación de la           //
//      cámara, que modifica con ellos el punto de vista de la      //
//      escena.                                                     //
//                                                                  //
//      Creado por F. Javier Rueda / 2001                           //
//      Modificado por Alejandro Guija Rodríguez /2006-2007         //
//                                                                  //
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_DLGCAMARA_H__1C2CBF22_E3B4_11D4_A351_810C1A248201__INCLUDED_)
#define AFX_DLGCAMARA_H__1C2CBF22_E3B4_11D4_A351_810C1A248201__INCLUDED_

#include "afxcmn.h"

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// DlgCamara.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CDlgCamara dialog

class CDlgCamara : public CDialog
{
// Construction
public:
        int nYaw;
        int nPitch;
        int YawAnt;
        int PitchAnt;
        CDlgCamara(CWnd* pParent = NULL);   // standard constructor
        CMainFrame* m_pFrame;

// Dialog Data
```

```
        //{{AFX_DATA(CDlgCamara)
        enum { IDD = IDD_DIALOGCAMARA };
        CSliderCtrl      m_YawSld;
        CSliderCtrl      m_PitchSld;
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CDlgCamara)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        virtual void PostNcDestroy();
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CDlgCamara)
        virtual BOOL OnInitDialog();
        afx_msg void OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
        virtual void OnCancel();
        virtual void OnOK();
        afx_msg void OnOutofmemorySlider1(NMHDR* pNMHDR, LRESULT* pResult);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_DLGCAMARA_H__1C2CBF22_E3B4_11D4_A351_810C1A248201__INCLUDED_)
```

B.2.6.2. DlgCamara.cpp

```
////////////////////////////////////////////////////////////
//                                                        //
// DlgCamara.cpp : archivo que implementa la clase        //
// CDlgCamara para el movimiento de la cámara             //
//                                                        //
////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Rx90.h"
#include "DlgCamara.h"
#include "MainFrm.h"

#define PI 3.14159

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
```

```
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CDlgCamara dialog


CDlgCamara::CDlgCamara(CWnd* pParent /*=NULL*/)
        : CDialog(CDlgCamara::IDD, pParent)
{
        //{{AFX_DATA_INIT(CDlgCamara)
        //}}AFX_DATA_INIT
}

void CDlgCamara::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CDlgCamara)
        DDX_Control(pDX, IDC_SLIDER2, m_YawSld);
        DDX_Control(pDX, IDC_SLIDER1, m_PitchSld);
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CDlgCamara, CDialog)
        //{{AFX_MSG_MAP(CDlgCamara)
        ON_WM_VSCROLL()
        ON_NOTIFY(NM_OUTOFMEMORY, IDC_SLIDER1, OnOutofmemorySlider1)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDlgCamara message handlers

void CDlgCamara::PostNcDestroy()
{
        delete (this);
        CDialog::PostNcDestroy();
}

//////////////////////////////////////////////////////////////////////
//                                                                  //
//      Rutina que salta cada vez que se quiere abrir el cuadro de  //
//      diálogo para mover el punto de vista de la cámara.          //
//                                                                  //
//////////////////////////////////////////////////////////////////////
BOOL CDlgCamara::OnInitDialog()
{
        CDialog::OnInitDialog();

        //Inicializamos barras de desplazamiento
        m_PitchSld.SetRange(-225, 45, TRUE);
        m_YawSld.SetRange(-180, 180, TRUE);

        //Reservamos la posicion inicial para el Cancel
        PitchAnt = m_pFrame->Pant;
```

```
        YawAnt = m_pFrame->Yant;

        //Asignamos la posición anterior después de un OK
        m_PitchSld.SetPos(m_pFrame->Pant);
        m_YawSld.SetPos(m_pFrame->Yant);

        //Asignamos la posición iniciar al cuadro de diálogo
        CString szPosvalP,szPosvalY;

        nPitch = m_PitchSld.GetPos();
        nYaw = m_YawSld.GetPos();

        szPosvalP.Format("%d",-nPitch);
        szPosvalY.Format("%d",-nYaw);

        SetDlgItemText(IDC_STATIC_PITCH, szPosvalP);
        SetDlgItemText(IDC_STATIC_YAW, szPosvalY);

        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}

//////////////////////////////////////////////////////////////////////
//                                                                  //
//      Método que controla el movimiento de los scroll verticales  //
//      aparecen en el cuadro de diálogod de "Ver punto de Vista"   //
//      Es llamada cuando se modifica la posición en alguna de las  //
//      barras.                                                     //
//                                                                  //
//////////////////////////////////////////////////////////////////////

void CDlgCamara::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
        CSliderCtrl *pSld=(CSliderCtrl*)(pScrollBar);
        int ID=pSld->GetDlgCtrlID();

        CString szPosvalP,szPosvalY;

        nPitch = m_PitchSld.GetPos();
        nYaw = m_YawSld.GetPos();

        szPosvalP.Format("%d",-nPitch);
        szPosvalY.Format("%d",-nYaw);

        SetDlgItemText(IDC_STATIC_PITCH, szPosvalP);
        SetDlgItemText(IDC_STATIC_YAW, szPosvalY);

// Le restamos a yaw 180° para que la visualización sea tal y como
// se ve el robot real en el laboratorio desde el PC.
        m_pFrame->SetCamera(m_PitchSld.GetPos()*PI/180.0,(m_YawSld.GetPos()-180)*PI/180.0);

        CDialog::OnVScroll(nSBCode, nPos, pScrollBar);
}

//////////////////////////////////////////////////////////////////////
```

```
//                                                                  //
//      Función que vuelve la escena a la vista original (a la      //
//      que había antes de abrir el cuadro de diálogo.              //
//                                                                  //
//////////////////////////////////////////////////////////////////////

void CDlgCamara::OnCancel()
{
        m_pFrame->Pant=PitchAnt;
        m_pFrame->Yant=YawAnt;

// Le restamos a yaw 180° para que la visualización sea tal y como
// se ve el robot real en el laboratorio desde el PC.

        m_pFrame->SetCamera(PitchAnt*PI/180.0,(YawAnt-180)*PI/180.0);
        if(m_pFrame->move != 0)
                m_pFrame->CamaraStart();

        CDialog::OnCancel();
}

//////////////////////////////////////////////////////////////////
//                                                              //
//      En el caso de que se pulse OK, nos quedamos con la      //
//      nueva vista de la escena.                               //
//                                                              //
//////////////////////////////////////////////////////////////////

void CDlgCamara::OnOK()
{
        m_pFrame->Pant = m_PitchSld.GetPos();
        m_pFrame->Yant = m_YawSld.GetPos();

        if(m_pFrame->move !=0)
                m_pFrame->CamaraStart();

        CDialog::OnOK();
}

void CDlgCamara::OnOutofmemorySlider1(NMHDR* pNMHDR, LRESULT* pResult)
{
        // TODO: Add your control notification handler code here

        *pResult = 0;
}
```

B.2.7. CDlgCoordArt.

B.2.7.1. DlgCoordArt.h

```
//////////////////////////////////////////////////////////////////
//                                                              //
//      DlgCoordArt.h: Definición de la clase CDlgCoordArt.     //
//                                                              //
//      Clase que permite al usuario modificar las variables    //
```

```
//      angulares del robot virtual moviéndolo en la escena.       //
//                                                                 //
//      Creado por Alejandro Guija Rodríguez.                      //
//                                                                 //
/////////////////////////////////////////////////////////////////////
#if !defined(AFX_DLGCOORDART_H__73DB54B3_B1E5_4F9D_8A25_264A26D805D9__INCLUDED_)
#define AFX_DLGCOORDART_H__73DB54B3_B1E5_4F9D_8A25_264A26D805D9__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// DlgCoordArt.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CDlgCoordArt dialog

class CMainFrame;

class CDlgCoordArt : public CDialog
{
// Construction
public:
        void ModeloCinematicoDirecto(double, double, double, double, double);
        int th[6];
        CDlgCoordArt(CWnd* pParent = NULL);   // standard constructor
        CMainFrame* m_pFrame;

// Dialog Data
        //{{AFX_DATA(CDlgCoordArt)
        enum { IDD = IDD_DIALOGANGULO };
        CSliderCtrl      m_artic6;
        CSliderCtrl      m_artic5;
        CSliderCtrl      m_artic4;
        CSliderCtrl      m_artic3;
        CSliderCtrl      m_artic2;
        CSliderCtrl      m_artic1;
        //}}AFX_DATA

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CDlgCoordArt)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CDlgCoordArt)
        virtual void OnCancel();
        virtual void OnOK();
        virtual BOOL OnInitDialog();
        afx_msg void OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
```

```
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_DLGCOORDART_H__73DB54B3_B1E5_4F9D_8A25_264A26D805D9__INCLUDED_)
```

B.2.7.2. DlgCoordArt.cpp

```
////////////////////////////////////////////////////////////
//                                                        //
// DlgCoordArt.cpp : archivo que implementa la clase      //
// CDlgCoordArt que abre el cuadro de diálogo para        //
// mover el robot mediante sus coordenadas articulares.  //
//                                                        //
////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Rx90.h"
#include "DlgCoordArt.h"
#include "MainFrm.h"
#include <math.h>

#define PI 3.1415926

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CDlgCoordArt dialog

CDlgCoordArt::CDlgCoordArt(CWnd* pParent /*=NULL*/)
        : CDialog(CDlgCoordArt::IDD, pParent)
{
        //{{AFX_DATA_INIT(CDlgCoordArt)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
}

void CDlgCoordArt::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CDlgCoordArt)
        DDX_Control(pDX, IDC_BARRA6, m_artic6);
        DDX_Control(pDX, IDC_BARRA5, m_artic5);
        DDX_Control(pDX, IDC_BARRA4, m_artic4);
        DDX_Control(pDX, IDC_BARRA3, m_artic3);
        DDX_Control(pDX, IDC_BARRA2, m_artic2);
        DDX_Control(pDX, IDC_BARRA1, m_artic1);
        //}}AFX_DATA_MAP
```

```
}

BEGIN_MESSAGE_MAP(CDlgCoordArt, CDialog)
        //{{AFX_MSG_MAP(CDlgCoordArt)
        ON_WM_HSCROLL()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDlgCoordArt message handlers

//////////////////////////////////////////////////////////////////
//                                                              //
//      Función que devuelve al robot su posición previa  a la  //
//      apertura de cuadro de diálogo "Movimiento manual articular.  //
//                                                              //
//////////////////////////////////////////////////////////////////

void CDlgCoordArt::OnCancel()
{
        // TODO: Add extra cleanup here
        int i;

// Cargamos la posición que se guardó al principio, al iniciar
// el cuadro de diálogo.

        for (i=0;i<6;i++)
                th[i]=m_pFrame->theta_ant[i];

//  Hacemos lo mismo dos veces porque a veces aunque la variable:
//  m_pRobot->PosicionAct.Artic se modifique correctamente, en la
//  imagen no se modifica bien la posicion.
//      Esto sucede cuando los cambios que ha de hacer el robot virtual
//  son muy bruscos.
//  Repitiendo el proceso dos veces nos aseguramos que el dibujo se
//  quede en la posicion que debe estar.

        for (i=0;i<2;i++)
        {
                m_pFrame->SetPosition(th[0]*PI/180-PI/2,0);
                m_pFrame->SetPosition((th[1]*PI/180+PI/2)/2,1);
                m_pFrame->SetPosition((th[2]*PI/180-PI)/3,2);
                m_pFrame->SetPosition(th[3]*PI/180/4,3);
                m_pFrame->SetPosition(th[4]*PI/180/5,4);
                m_pFrame->SetPosition(th[5]*PI/180/6,5);
        }

        CDialog::OnCancel();
}

//////////////////////////////////////////////////////////////////
//                                                              //
//      Fución que valida los cambios efectuados en las variables   //
//      angulares y que modifican la posición del robot.        //
//                                                              //
```

```
////////////////////////////////////////////////////////////////////

void CDlgCoordArt::OnOK()
{
        // TODO: Add extra validation here

        m_pFrame->theta_ant[0]=m_artic1.GetPos();
        m_pFrame->theta_ant[1]=m_artic2.GetPos();
        m_pFrame->theta_ant[2]=m_artic3.GetPos();
        m_pFrame->theta_ant[3]=m_artic4.GetPos();
        m_pFrame->theta_ant[4]=m_artic5.GetPos();
        m_pFrame->theta_ant[5]=m_artic6.GetPos();

        CDialog::OnOK();
}

////////////////////////////////////////////////////////////////////
//                                                                //
//      Funcion que inicializa el cuadro de diálogo que implementa //
//      el movimiento manual angular de cada articulación.        //
//                                                                //
////////////////////////////////////////////////////////////////////

BOOL CDlgCoordArt::OnInitDialog()
{
        CDialog::OnInitDialog();
        double pos[6];
        int i;

// TODO: Add extra initialization here
// Establecemos el rango de las barras de deslizamiento.

        m_artic1.SetRange(-160,160,TRUE);
        m_artic2.SetRange(-200,35,TRUE);
        m_artic3.SetRange(-52,232,TRUE);
        m_artic4.SetRange(-270,270,TRUE);
        m_artic5.SetRange(-105,120,TRUE);
        m_artic6.SetRange(-270,270,TRUE);

// Cargamos en pos[i] la coordenada actual de la articulacion i
// del robot virtual.

        for (i=0;i<6;i++)
        {
                pos[i]=m_pFrame->m_pRobot->Posicionact.Artic[i];
        }

// Lo pasamos a grados y escalamos la posición para que los ángulos
// crezcan igual que los del Rx90 real,

        pos[0]=pos[0]*180/PI+90;
        pos[1]=(pos[1]*180/PI)*2-90;
        pos[2]=(pos[2]*180/PI)*3+180;
        pos[3]=(pos[3]*180/PI)*4;
        pos[4]=(pos[4]*180/PI)*5;
```

```
        pos[5]=(pos[5]*180/PI)*6;

// Imponemos la posición inicial de las barras de deslizamiento
// según la posición actual del robot virtual.

        m_artic1.SetPos(pos[0]);
        m_artic2.SetPos(pos[1]);
        m_artic3.SetPos(pos[2]);
        m_artic4.SetPos(pos[3]);
        m_artic5.SetPos(pos[4]);
        m_artic6.SetPos(pos[5]);

// Guardamos la posición inicial por si se pulsa CANCEL

        for (i=0;i<6;i++)
                m_pFrame->theta_ant[i]=pos[i];

        CString angulo[6];

        for (i=0;i<6;i++)
                angulo[i].Format("%d",m_pFrame->theta_ant[i]);

        SetDlgItemText(IDC_ANGULO1,angulo[0]);
        SetDlgItemText(IDC_ANGULO2,angulo[1]);
        SetDlgItemText(IDC_ANGULO3,angulo[2]);
        SetDlgItemText(IDC_ANGULO4,angulo[3]);
        SetDlgItemText(IDC_ANGULO5,angulo[4]);
        SetDlgItemText(IDC_ANGULO6,angulo[5]);

// Llamamos a la siguiente funcion para aplicar el modelo directo
// y calcular la posición y la orientación del extremo.

        ModeloCinematicoDirecto(pos[0],pos[1],pos[2],pos[3],pos[4]);

        return TRUE;  // return TRUE unless you set the focus to a control                //
EXCEPTION: OCX Property Pages should return FALSE
}

//////////////////////////////////////////////////////////////////////
//                                                                  //
//       Función que controla los cambios producidos en las barras  //
//       deslizadoras del diálogo "Movimiento manual articular".    //
//                                                                  //
//////////////////////////////////////////////////////////////////////

void CDlgCoordArt::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
// TODO: Add your message handler code here and/or call default

        CSliderCtrl* slider = (CSliderCtrl*)pScrollBar;

// Cargamos la nueva posición de las barras de deslizamiento.

        th[0] = m_artic1.GetPos();
        th[1] = m_artic2.GetPos();
```

```
        th[2] = m_artic3.GetPos();
        th[3] = m_artic4.GetPos();
        th[4] = m_artic5.GetPos();
        th[5] = m_artic6.GetPos();

        CString angulo1,angulo2,angulo3,angulo4,angulo5,angulo6;

        angulo1.Format("%d",th[0]);
        angulo2.Format("%d",th[1]);
        angulo3.Format("%d",th[2]);
        angulo4.Format("%d",th[3]);
        angulo5.Format("%d",th[4]);
        angulo6.Format("%d",th[5]);

// Imprimimos ese nuevo valor de las barras de deslizamiento.

        SetDlgItemText(IDC_ANGULO1,angulo1);
        SetDlgItemText(IDC_ANGULO2,angulo2);
        SetDlgItemText(IDC_ANGULO3,angulo3);
        SetDlgItemText(IDC_ANGULO4,angulo4);
        SetDlgItemText(IDC_ANGULO5,angulo5);
        SetDlgItemText(IDC_ANGULO6,angulo6);

// Calculamos ahora la posición y la orientación del extremo
// para las nuevas coordenadas articulares.

        ModeloCinematicoDirecto(th[0],th[1],th[2],th[3],th[4]);

// Movemos el robot virtual pasando los valores a radianes y
// reescalandolos.

        m_pFrame->SetPosition(th[0]*PI/180-PI/2,0);
        m_pFrame->SetPosition((th[1]*PI/180+PI/2)/2,1);
        m_pFrame->SetPosition((th[2]*PI/180-PI)/3,2);
        m_pFrame->SetPosition(th[3]*PI/180/4,3);
        m_pFrame->SetPosition(th[4]*PI/180/5,4);
        m_pFrame->SetPosition(th[5]*PI/180/6,5);

        CDialog::OnHScroll(nSBCode, nPos, pScrollBar);
}

////////////////////////////////////////////////////////////////////////////
//                                                                        //
//      Función que calcula el modelo cinemático directo del              //
//      manipulador Rx-90 calculando a través de sus primeras 5           //
//      variables articulares (la sexta no afecta debido a la             //
//      radial en el eje Z de la fresa)   las coordenadas cartesianas     //
//      de su posición y los ángulos de su orientación.                   //
//                                                                        //
////////////////////////////////////////////////////////////////////////////

void CDlgCoordArt::ModeloCinematicoDirecto(double t1, double t2, double t3, double t4, double t5)
{

// Valores del Rx90 en mm.
```

```
        int a2=450;
        int a3=0;
        int d1=0;
        // Sería d1=420, pero no lo ponemos porque el sistema
        // de referencia World está encima de la base, no en
        // el suelo.
        int d3=0;
        int d4=450;
        int d6=360;      // MEDIDA DE LA FRESA = 275+85

        double cartX;
        double cartY;
        double cartZ;
        double orientX;
        double orientY;
        double orientZ;

// Para simplificar las expreiones declaramos los siguiente valores:
        double c1=cos(t1*PI/180);
        double c2=cos(t2*PI/180);
        double c3=cos(t3*PI/180);
        double c4=cos(t4*PI/180);
        double c5=cos(t5*PI/180);
        double s1=sin(t1*PI/180);
        double s2=sin(t2*PI/180);
        double s3=sin(t3*PI/180);
        double s4=sin(t4*PI/180);
        double s5=sin(t5*PI/180);
        double s23=sin((t2+t3)*PI/180);
        double c23=cos((t2+t3)*PI/180);

//  Posicion del extremo utilizando Hemero con parámetros DH

//               eje  | alpha(i-1) | a(i-1)  |  d(i)  |  theta(i)
//               ------------------------------------------------
//                 1  |      0     |    0    |   0    |    t1
//                 2  |    -90     |    0    |   0    |   t2
//                 3  |      0     |   a(2)  |   0    |    t3
//                 4  |     90°    |    0    |  d(4)  |    t4
//                 5  |    -90°    |    0    |   0    |    t5
//                 6  |     90°    |    0    |  d(6)  |    t6

// Aplicando el modelo directo, la matriz de transformación
// homogénea resulta:

//               [ n(x) s(x)  a(x) | X ]
//               [ n(y) s(y)  a(y) | Y ]
//       T=      [ n(z) s(z)  a(z) | Z ]
//               [-----------------|---]
//               [  0    0     0   | 1 ]

// Donde X, Y, Z es la posición del extremo según el sistema de referencia
// World, y las ecuaciones son las siguientes:
```

```
        cartX=((c1*c23*c4-s1*s4)*s5+c1*s23*c5)*d6+c1*s23*d4+c1*c2*a2;
        cartY=((s1*c23*c4+c1*s4)*s5+s1*s23*c5)*d6+s1*s23*d4+s1*c2*a2;
        cartZ=-(s23*c4*s5-c23*c5)*d6+c23*d4-s2*a2+d1;

        CString Xstring, Ystring, Zstring;

        Xstring.Format("%f",cartX);
        Ystring.Format("%f",cartY);
        Zstring.Format("%f",cartZ);

// Imprimimos las posiciones calculadas.

        SetDlgItemText(IDC_POSx,Xstring);
        SetDlgItemText(IDC_POSy,Ystring);
        SetDlgItemText(IDC_POSz,Zstring);

//  Calculamos la orientación de la fresa (tercera columna de la
//  matriz de transformación homogénea: a(x), a(y) y a(z)).

        orientX=(c1*c23*c4-s1*s4)*s5+c1*s23*c5;
        orientY=(s1*c23*c4+c1*s4)*s5+s1*s23*c5;
        orientZ=-s23*c4*s5+c23*c5;

        Xstring.Format("%f",orientX);
        Ystring.Format("%f",orientY);
        Zstring.Format("%f",orientZ);

// Imprimimos dichos valores de orientación.

        SetDlgItemText(IDC_YAW,Xstring);
        SetDlgItemText(IDC_PITCH,Ystring);
        SetDlgItemText(IDC_ROLL,Zstring);
}
```

B.2.8. CDlgCoordCart.

B.2.8.1. DlgCoordCart.h

```
////////////////////////////////////////////////////////////////////
//                                                                //
//      DlgCoordCart.h: Definición de la clase CDlgCoordCart.     //
//                                                                //
//      Clase que permite al usuario modificar la posición del    //
//      extremo del efector final del robot, moviéndolo según     //
//      los ejes del sistema de referencia world.                 //
//                                                                //
//      Creado por Alejandro Guija Rodríguez.                     //
//                                                                //
////////////////////////////////////////////////////////////////////

#if !defined(AFX_DLGCOORDCART_H__E43B492D_18A8_43DF_993F_A0A7B4A7229E__INCLUDED_)
#define AFX_DLGCOORDCART_H__E43B492D_18A8_43DF_993F_A0A7B4A7229E__INCLUDED_
```

```
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// DlgCoordCart.h : header file
//
class CRobot;
class CPuntoTrayectoria;

/////////////////////////////////////////////////////////////////////////////
// CDlgCoordCart dialog

class CDlgCoordCart : public CDialog
{
// Construction
public:
        void ModeloCinematicoInverso(int, int, int);
        void ModeloCinematicoDirecto(double, double, double, double, double);

        CDlgCoordCart(CWnd* pParent = NULL);   // standard constructor
        CMainFrame* m_pFrame;
//      CRobot* m_pRobot;

// Dialog Data
        //{{AFX_DATA(CDlgCoordCart)
        enum { IDD = IDD_DIALOGPOS };
        CSliderCtrl     m_posicz;
        CSliderCtrl     m_posicy;
        CSliderCtrl     m_posicx;
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CDlgCoordCart)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CDlgCoordCart)
        virtual void OnCancel();
        virtual BOOL OnInitDialog();
        afx_msg void OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_DLGCOORDCART_H__E43B492D_18A8_43DF_993F_A0A7B4A7229E__INCLUDED_)
```

B.2.8.2. DlgCoordCart.cpp

```
//////////////////////////////////////////////////////////////
//                                                          //
// DlgCoordCart.cpp : archivo que implementa la clase       //
// CDlgCoordCart que abre el cuadro de diálogo para         //
// mover el extremo del robot suiguiendo los ejes           //
// cartesianos.                                             //
//                                                          //
//////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Rx90.h"
#include "DlgCoordCart.h"
#include "MainFrm.h"
#include "Robot.h"
#include "Estructura.h"
#include <math.h>
#include "Resource.h"
#include "PuntoTrayectoria.h"


#define PI 3.14159265358979310862446 8
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CDlgCoordCart dialog

CDlgCoordCart::CDlgCoordCart(CWnd* pParent /*=NULL*/)
        : CDialog(CDlgCoordCart::IDD, pParent)
{
        //{{AFX_DATA_INIT(CDlgCoordCart)
        //}}AFX_DATA_INIT
}

void CDlgCoordCart::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CDlgCoordCart)
        DDX_Control(pDX, IDC_BARRA9, m_posicz);
        DDX_Control(pDX, IDC_BARRA8, m_posicy);
        DDX_Control(pDX, IDC_BARRA7, m_posicx);
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CDlgCoordCart, CDialog)
        //{{AFX_MSG_MAP(CDlgCoordCart)
        ON_WM_VSCROLL()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
////////////////////////////////////////////////////////////////////////////
// CDlgCoordCart message handlers

/////////////////////////////////////////////////////////////////////
//                                                                 //
//      Función que impone los valores angulares que tenían las    //
//      articulaciones del robot antes de iniciarse el actual      //
//      cuadro de diálogo.                                         //
//                                                                 //
/////////////////////////////////////////////////////////////////////

void CDlgCoordCart::OnCancel()
{
        // TODO: Add extra cleanup here

// Cargamos la posición que se guardó al principio, al iniciar
// el cuadro de diálogo.

//  Hacemos lo mismo dos veces porque a veces aunque la variable:
//  m_pRobot->PosicionAct.Artic se modifique correctamente, en la
//  imagen no se modifica bien la posicion.
//      Esto sucede cuando los cambios que ha de hacer el robot virtual
//  son muy bruscos.
//  Repitiendo el proceso dos veces nos aseguramos que el dibujo se
//  quede en la posicion que debe estar.

        for (int i=0;i<2;i++)
        {
                m_pFrame->SetPosition((m_pFrame->theta_ant[0]-90)*PI/180,0);
                m_pFrame->SetPosition(((m_pFrame->theta_ant[1]+90)*PI/180)/2,1);
                m_pFrame->SetPosition(((m_pFrame->theta_ant[2]-180)*PI/180)/3,2);
                m_pFrame->SetPosition(((m_pFrame->theta_ant[3])*PI/180)/4,3);
                m_pFrame->SetPosition(((m_pFrame->theta_ant[4])*PI/180)/5,4);
        }

        CDialog::OnCancel();
}

/////////////////////////////////////////////////////////////////////
//                                                                 //
//      Funcion que inicializa el cuadro de diálogo que implementa //
//      el movimiento manual cartesiano del extremo del robot.     //
//                                                                 //
/////////////////////////////////////////////////////////////////////

BOOL CDlgCoordCart::OnInitDialog()
{
        CDialog::OnInitDialog();

        CString teta[6];
        double angulo[6];
        int i;

// Establecemos el rango de las barras de deslizamiento.
```

```
        m_posicx.SetRange(-1300,1300,TRUE);
        m_posicy.SetRange(-1300,1300,TRUE);
        m_posicz.SetRange(-1300,1300,TRUE);

// Cargamos en angulo[i] la coordenada actual de la articulacion i
// del robot virtual.

        for (i=0;i<6;i++)
        {
                angulo[i]=m_pFrame->m_pRobot->Posicionact.Artic[i];
        }

// Lo pasamos a grados y escalamos la posición para que los ángulos
// crezcan igual que los del Rx90 real,

        angulo[0]=angulo[0]*180/PI+90;
        angulo[1]=(angulo[1]*180/PI)*2-90;
        angulo[2]=(angulo[2]*180/PI)*3+180;
        angulo[3]=(angulo[3]*180/PI)*4;
        angulo[4]=(angulo[4]*180/PI)*5;
        angulo[5]=(angulo[5]*180/PI)*6;

// Guardamos la posición inicial por si se pulsa CANCEL
        for (i=0;i<6;i++)
        {
                teta[i].Format("%f",angulo[i]);
                m_pFrame->theta_ant[i]=angulo[i];
        }

// Imprimimos el valor de cada coordenaadr articular.

        SetDlgItemText(IDC_THETA1,teta[0]);
        SetDlgItemText(IDC_THETA2,teta[1]);
        SetDlgItemText(IDC_THETA3,teta[2]);
        SetDlgItemText(IDC_THETA4,teta[3]);
        SetDlgItemText(IDC_THETA5,teta[4]);
        SetDlgItemText(IDC_THETA6,teta[5]);

// Calculamos con el método directo la posición y la orientación
// del extremo.

        ModeloCinematicoDirecto(angulo[0],angulo[1],angulo[2],angulo[3],angulo[4]);
        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}

////////////////////////////////////////////////////////////////////////
//                                                                    //
//      Función que calcula el modelo cinemático directo del          //
//      manipulador Rx-90 calculando a través de sus primeras 5       //
//      variables articulares (la sexta no afecta debido a la         //
//      radial en el eje Z de la fresa)   las coordenadas cartesianas //
//      de su posición y los ángulos de su orientación.               //
//                                                                    //
```

```
////////////////////////////////////////////////////////////////////////////

void CDlgCoordCart::ModeloCinematicoDirecto(double t1, double t2, double t3, double t4, double t5)
{

// Valores del Rx90 en mm.

        int a2=450;
        int a3=0;
        int d1=0;        // Sería d1=420, pero no lo ponemos porque el sistema
                         // de referencia World está encima de la base, no en
                         // el suelo.
        int d3=0;
        int d4=450;
        int d6=360;      // MEDIDA DE LA FRESA = 275+85

        int cartX;
        int cartY;
        int cartZ;
        double orientX;
        double orientY;
        double orientZ;

        double c1=cos(t1*PI/180);
        double c2=cos(t2*PI/180);
        double c3=cos(t3*PI/180);
        double c4=cos(t4*PI/180);
        double c5=cos(t5*PI/180);
        double s1=sin(t1*PI/180);
        double s2=sin(t2*PI/180);
        double s3=sin(t3*PI/180);
        double s4=sin(t4*PI/180);
        double s5=sin(t5*PI/180);
        double s23=sin((t2+t3)*PI/180);
        double c23=cos((t2+t3)*PI/180);

//  Posicion del extremo segun Hemero con parámetros DH de Manuel
//  cambiando algunos signos de alpha(i-1)

//              eje  | alpha(i-1) | a(i-1)  |  d(i)  |  theta(i)
//              ------------------------------------------------
//                1  |      0     |    0    |   0    |    t1
//                2  |    -90°    |    0    |   0    |    t2
//                3  |      0     |   a(2)  |   0    |    t3
//                4  |     90°    |    0    |  d(4)  |    t4
//                5  |    -90°    |    0    |   0    |    t5
//                6  |     90°    |    0    |  d(6)  |    t6

// Aplicando el modelo directo, la matriz de transformación
// homogénea resulta:

//              [ n(x) s(x)  a(x) | X ]
//              [ n(y) s(y)  a(y) | Y ]
//      T=      [ n(z) s(z)  a(z) | Z ]
//              [-----------------|---]
```

```
//              [  0    0     0   | 1 ]

// Donde X, Y, Z es la posición del extremo según el sistema de
// referencia world, y las ecuaciones son las siguientes:

        cartX=((c1*c23*c4-s1*s4)*s5+c1*s23*c5)*d6+c1*s23*d4+c1*c2*a2;
        cartY=((s1*c23*c4+c1*s4)*s5+s1*s23*c5)*d6+s1*s23*d4+s1*c2*a2;
        cartZ=-(s23*c4*s5-c23*c5)*d6+c23*d4-s2*a2+d1;

        CString Xstring, Ystring, Zstring;

        Xstring.Format("%d",cartX);
        Ystring.Format("%d",cartY);
        Zstring.Format("%d",cartZ);

// Imprimimos las posiciones calculadas.

        SetDlgItemText(IDC_XPOS,Xstring);
        SetDlgItemText(IDC_YPOS,Ystring);
        SetDlgItemText(IDC_ZPOS,Zstring);

// Imponemos la posición inicial de las barras de deslizamiento
// a las coordenadas calculadas.

        m_posicx.SetPos(cartX);
        m_posicy.SetPos(cartY);
        m_posicz.SetPos(cartZ);

        m_pFrame->pos[0]=m_posicx.GetPos();
        m_pFrame->pos[1]=m_posicy.GetPos();
        m_pFrame->pos[2]=m_posicz.GetPos();

//  Calculamos la orientación de la fresa (tercera columna de la
//  matriz de transformación homogénea: a(x), a(y) y a(z)).

        orientX=(c1*c23*c4-s1*s4)*s5+c1*s23*c5;
        orientY=(s1*c23*c4+c1*s4)*s5+s1*s23*c5;
        orientZ=-s23*c4*s5+c23*c5;

        Xstring.Format("%f",orientX);
        Ystring.Format("%f",orientY);
        Zstring.Format("%f",orientZ);

        SetDlgItemText(IDC_YAW,Xstring);
        SetDlgItemText(IDC_PITCH,Ystring);
        SetDlgItemText(IDC_ROLL,Zstring);

}

//////////////////////////////////////////////////////////////////
//                                                              //
//      Función que controla las barras deslizadoras que modifican  //
//      las coordenadas cartesianas de la posición del extremo de  //
//      la fresa sin cambiar la orientación de la misma.        //
//                                                              //
```

```
//////////////////////////////////////////////////////////////////////

void CDlgCoordCart::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
        // TODO: Add your message handler code here and/or call default

        CString ejex,ejey,ejez;
        BOOL fRes;
        int posaux[3];
        double angulo[6];

//      Leemos la nueva posición de los scrolls.
        posaux[0]=m_posicx.GetPos();
        posaux[1]=m_posicy.GetPos();
        posaux[2]=m_posicz.GetPos();

        ejex.Format("%d",posaux[0]);
        ejey.Format("%d",posaux[1]);
        ejez.Format("%d",posaux[2]);

// Imprimimos en los cuadros de texto adjuntos los valores leídos.
        SetDlgItemText(IDC_XPOS,ejex);
        SetDlgItemText(IDC_YPOS,ejey);
        SetDlgItemText(IDC_ZPOS,ejez);

        CRx90App* pAplicacion = (CRx90App*)AfxGetApp();
        pAplicacion->m_pPuntoTrayectoria = new CPuntoTrayectoria;

//      Para los nuevos valores de X, Y y Z llamamaos a la función
//      que va a calcular las variables angulares correspondientes.
        ModeloCinematicoInverso(posaux[0],posaux[1],posaux[2]);

//      Comprobamos que los ángulos obtenidos respetan laslimitaciones
//      mecánicas de los ejes del robot real.
        fRes = pAplicacion->m_pPuntoTrayectoria->EsAlcanzable();

        if (fRes == FALSE)
        {
//      Si ha habido error cargamos los valores que se guardaron
//      con la función OnInitDialog().
                ejex.Format("%d",m_pFrame->pos[0]);
                ejey.Format("%d",m_pFrame->pos[1]);
                ejez.Format("%d",m_pFrame->pos[2]);

                SetDlgItemText(IDC_XPOS,ejex);
                SetDlgItemText(IDC_YPOS,ejey);
                SetDlgItemText(IDC_ZPOS,ejez);

// Imponemos los valores anteriores a las barras deslizadoras.
                m_posicx.SetPos(m_pFrame->pos[0]);
                m_posicy.SetPos(m_pFrame->pos[1]);
                m_posicz.SetPos(m_pFrame->pos[2]);

// Indicamos el error que se ha producido mediante un mensaje.
                CString strError;
```

```
                strError.Format(_T("Se sale del espacio alcanzable."));
                AfxMessageBox(strError,MB_ICONEXCLAMATION|MB_OK);
        }
        else
        {
// Por el contrario si la nueva posición del robot entra dentro
// de los límites del robot se imponen en el mismo.
                CString ANGULO1, ANGULO2, ANGULO3, ANGULO4, ANGULO5;

                m_pFrame->pos[0]=posaux[0];
                m_pFrame->pos[1]=posaux[1];
                m_pFrame->pos[1]=posaux[2];

                angulo[0]=(pAplicacion->m_pPuntoTrayectoria->m_sCoordTheta.theta[0])*PI/180;
                angulo[1]=(pAplicacion->m_pPuntoTrayectoria->m_sCoordTheta.theta[1])*PI/180;
                angulo[2]=(pAplicacion->m_pPuntoTrayectoria->m_sCoordTheta.theta[2])*PI/180;
                angulo[3]=(pAplicacion->m_pPuntoTrayectoria->m_sCoordTheta.theta[3])*PI/180;
                angulo[4]=(pAplicacion->m_pPuntoTrayectoria->m_sCoordTheta.theta[4])*PI/180;

                ANGULO1.Format("%f",angulo[0]*180/PI);
                ANGULO2.Format("%f",angulo[1]*180/PI);
                ANGULO3.Format("%f",angulo[2]*180/PI);
                ANGULO4.Format("%f",angulo[3]*180/PI);
                ANGULO5.Format("%f",angulo[4]*180/PI);

                SetDlgItemText(IDC_THETA1,ANGULO1);
                SetDlgItemText(IDC_THETA2,ANGULO2);
                SetDlgItemText(IDC_THETA3,ANGULO3);
                SetDlgItemText(IDC_THETA4,ANGULO4);
                SetDlgItemText(IDC_THETA5,ANGULO5);

                ejex.Format("%d",posaux[0]);
                ejey.Format("%d",posaux[1]);
                ejez.Format("%d",posaux[2]);

                SetDlgItemText(IDC_XPOS,ejex);
                SetDlgItemText(IDC_YPOS,ejey);
                SetDlgItemText(IDC_ZPOS,ejez);

                m_pFrame->SetPosition(angulo[0]-PI/2,0);
                m_pFrame->SetPosition((angulo[1]+PI/2)/2,1);
                m_pFrame->SetPosition((angulo[2]-PI)/3,2);
                m_pFrame->SetPosition(angulo[3]/4,3);
                m_pFrame->SetPosition(angulo[4]/5,4);
                //m_pFrame->SetPosition(angulo[5]/6,5);
        }

        CDialog::OnVScroll(nSBCode, nPos, pScrollBar);
}

//////////////////////////////////////////////////////////////////////////
//                                                                      //
//      Función que a partir de la posición impuesta por el usuario     //
//      a través de los scrolls y de la orientación actual del          //
//      efector final, obtiene las variables articulares.               //
```

```
//                                                                          //
//////////////////////////////////////////////////////////////////////////////

void CDlgCoordCart::ModeloCinematicoInverso(int x, int y, int z)
{
        double angulo[6];
        double orient[3];
        int i;

        int d6=360;
        int d1=0;
        int a2=450;
        int a3=0;
        int d4=450;

        double dX, dY, dZ;
        double c1,c2,c3,c4,c5,s1,s2,s3,s4,s5,s23,c23;
        double dK, dAux1, dAux2, dAux3, dAux4;
        CRx90App* pAplicacion = (CRx90App*)AfxGetApp();

// Vemos primero cual es la orientación actual de la fresa.
// Para ello hemos de tener las coordenadas articulares de las que partimos.

        for (i=0;i<6;i++)
        {
                angulo[i]=m_pFrame->m_pRobot->Posicionact.Artic[i];
        }

        angulo[0]=angulo[0]+PI/2;
        angulo[1]=angulo[1]*2-PI/2;
        angulo[2]=angulo[2]*3+PI;
        angulo[3]=angulo[3]*4;
        angulo[4]=angulo[4]*5;
        angulo[5]=angulo[5]*6;

        c1=cos(angulo[0]);
        c2=cos(angulo[1]);
        c3=cos(angulo[2]);
        c4=cos(angulo[3]);
        c5=cos(angulo[4]);
        s1=sin(angulo[0]);
        s2=sin(angulo[1]);
        s3=sin(angulo[2]);
        s4=sin(angulo[3]);
        s5=sin(angulo[4]);
        s23=sin(angulo[1]+angulo[2]);
        c23=cos(angulo[1]+angulo[2]);

// A partir de los angulos iniciales, calculamos la orientacion,
// que es lo mismo que la 3ª columna de la matriz de transformacion
// del modelo directo.

        orient[0]=(c1*c23*c4-s1*s4)*s5+c1*s23*c5;
        orient[1]=(s1*c23*c4+c1*s4)*s5+s1*s23*c5;
        orient[2]=-s23*c4*s5+c23*c5;
```

```
// Deducción del tamaño de la herramienta

        dX = x - d6*orient[0];
        dY = y - d6*orient[1];
        dZ = z - d6*orient[2];

        dK=(dX*dX + dY*dY + dZ*dZ - a2*a2 - d4*d4)/(2*a2);

        // Cálculo de ángulos de las articulaciones

        // Calculamos theta1

        angulo[0] = atan2(dY,dX);
        pAplicacion->m_pPuntoTrayectoria->m_sCoordTheta.theta[0]=angulo[0]*180/PI;
        c1=cos(angulo[0]);
        s1=sin(angulo[0]);

        // Calculo de theta2

        dAux1=sqrt(dX*dX+dY*dY);
        dAux2=dX*dX+dY*dY+dZ*dZ;
        dAux3=atan2(dZ,dAux1);
        dAux4=acos((d4*d4-dAux2-a2*a2)/(-2*sqrt(dAux2)*a2));
        angulo[1]=-(dAux3+dAux4);
        pAplicacion->m_pPuntoTrayectoria->m_sCoordTheta.theta[1]=angulo[1]*180/PI;
        c2=cos(angulo[1]);
        s2=sin(angulo[1]);

        // Calculo de theta3

        angulo[2]=PI/2+2*dAux4;
        pAplicacion->m_pPuntoTrayectoria->m_sCoordTheta.theta[2]=angulo[2]*180/PI;
        c3=cos(angulo[2]);
        s2=cos(angulo[2]);

        s23=sin(angulo[1]+angulo[2]);
        c23=cos(angulo[1]+angulo[2]);

        // Calculo de theta4
        int nMuneca=-1;
        dAux1 = nMuneca*(c1*orient[1] - s1*orient[0]);
        dAux2 = nMuneca*(c1*c23*orient[0] + s1*c23*orient[1] - s23*orient[2]);

        angulo[3]=atan2(dAux1, dAux2);
        pAplicacion->m_pPuntoTrayectoria->m_sCoordTheta.theta[3]=angulo[3]*180/PI;
        s4 = sin(angulo[3]);
        c4 = cos(angulo[3]);

        // Calculamos theta5
        dAux1 = orient[0]*(c1*c23*c4 - s1*s4) + orient[1]*(s1*c23*c4 + c1*s4) - orient[2]*c4*s23;
        dAux2 = orient[0]*c1*s23 + orient[1]*s1*s23 + orient[2]*c23;

        angulo[4]=atan2(dAux1, dAux2);
        pAplicacion->m_pPuntoTrayectoria->m_sCoordTheta.theta[4]=angulo[4]*180/PI;
```

```
        s5 = sin(angulo[4]);
        c5 = cos(angulo[4]);

        pAplicacion->m_pPuntoTrayectoria->m_sCoordTheta.theta[5]=0;
}
```

B.2.9. CCaptura.

B.2.9.1. Captura.h

```
////////////////////////////////////////////////////////////
//                                                        //
//      Captura.h: Definición de la clase CCaptura.       //
//                                                        //
//      La clase CCaptura se ocupa de obtener los datos de //
//      la trayectoria que va aseguir el robot y leerlos  //
//      cuando lo requiera la clase CMainFrame.           //
//                                                        //
//      Creada por Alejandro Guija Rodríguez / 2006-2007  //
//                                                        //
////////////////////////////////////////////////////////////

#if !defined(AFX_CAPTURA_H__AE3059C2_8441_11D1_B61B_444553540000__INCLUDED_)
#define AFX_CAPTURA_H__AE3059C2_8441_11D1_B61B_444553540000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define BYTE unsigned char

// Estructura con dos campos, uno para los ángulos de los seis
// ejes del robot y otro para abrir y cerrar la garra (en el caso
// de que el extremo no sea la frea neumática).
struct Posicion
{
        double giros[6];
        int garra;
};

class CMainFrame;
class CPuntoTrayectoria;


class CCaptura
{
public:
        void AproximacionPuntoInicial();
        void CalculaPuntoCentral(int NPuntos);
        double aux [3];
        int TotalPuntosAPT;

        int func;
        int iteracion;
```

```
        CWnd Wind2;
        CWnd Wind;
        Posicion Pos;
        void where(void);
        int hand;
        double joint[6];
        double tablaaux[72][6];
        CMainFrame* m_pFrame;
        void Capture(void);
        CCaptura();
        virtual ~CCaptura();
        BOOL AbrirYLeerFicheroAPT(CString& strContFicheroAPT);
        BOOL ProcesarContenidoFicheroAPT(CString& strContFicheroAPT);
        Vector m_vPuntos[3];

        struct coordTheta
        {
                double theta[6];
        } m_sCoordTheta;
};

#endif // !defined(AFX_CAPTURA_H__AE3059C2_8441_11D1_B61B_444553540000__INCLUDED_)
```

B.2.9.2. Captura.cpp

```
////////////////////////////////////////////////////////////
//                                                        //
// Captura.cpp: implementación de la clase CCaptura.      //
//                                                        //
////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Rx90.h"
#include "MainFrm.h"
#include "Captura.h"
#include "PuntoTrayectoria.h"

#include "afxmt.h"

#include <conio.h>
#include <dos.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <iostream>
#include <fstream.h>
#include "afxtempl.h"

#define PI 3.14159

// En "pasos" ponemos el número de posiciones por las que pasará
// antes de colocarse en la posición inicial de la trayectoria
// de corte. Cuanto mayor sea "pasos", más despacio y menos brusco
// será el movimiento del robot hasta dicha posición inicial
```

```
// desde su posición actual.

#define pasos 10

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CCaptura::CCaptura()
{
}

CCaptura::~CCaptura()
{
}

//////////////////////////////////////////////////////////////////////////////
//                                                                          //
// Función utilizada para leer las posiciones de la trayectoria virtual.    //
// Pueden darse dos casos:                                                  //
//               - Se ejecuta la aplicación desde Corte3D.exe.              //
//                       Los datos se leen desde un archivo que se creó en  //
//                       ese programa     para no rehacer los cálculos.     //
//               - Se ejecuta directamente.                                 //
//                       Los datos hay que preguntarselos al operario,      //
//                       leerlos de un archivo .apt y procesarlos. No hay   //
//                       nada hecho previamente.                            //
//                                                                          //
//////////////////////////////////////////////////////////////////////////////
void CCaptura::Capture()
{

        CRx90App* pAplicacion = (CRx90App*)AfxGetApp();

        BOOL fRes=TRUE;

        int nLongitudLinea;

        CFile file;
        try
        {
                file.Open("posiciones.txt", CFile::modeRead);
        }
        catch (CFileException *e)
        {
                CString strError;
                strError.Format("Error al abrir el archivo. Error número %d",e->m_cause);
                e->Delete();
                AfxMessageBox(strError);
```

```
                fRes=FALSE;
        }

        if (fRes)
        {
                int nFileLength = file.GetLength();
                char *lpBuffer = new char[nFileLength +1];
// Leemos todo el contenido del buffer del puerto serie y
// lo almacenamos en nuestro buffer temporal.
// Cuando quitamos el fichero que ha de leer debería dar
// un error y saltar el "catch", pero no lo hace
// NO LO ENTIENDO.
                try
                {
                        file.Read(lpBuffer,nFileLength);
                }
                catch (CFileException *e)
                {
                        CString strError;
                        strError.Format("Error al leer el archivo. Error número %d",e->m_cause);
                        AfxMessageBox(strError);
                        e->Delete();
                        delete[] lpBuffer;
                        fRes=FALSE;
                }

                // Añadimos un carácter NULL para terminar la cadena.
                lpBuffer[nFileLength] = '\0';

                // Cerramos el fichero.
                file.Close();

                CString strContFichero, strBuffer, strBuffer2;

                // Almacenamos el contenido del fichero en una instancia de la clase CString, más
fácil de usar.
                strContFichero.Format(_T("%s"),lpBuffer);

                // Vaciamos y eliminamos el buffer temporal.
                delete[] lpBuffer;

                int nIndex, nNumeroPuntosAPT;

// Contamos el número de puntos que hay en el fichero APT. Para
// ello recorremos el objeto CString y vamos extrayendo las
// líneas donde hay puntos y eliminado aquellas que no los
// contienen. Los puntos extraidos se almacenan en strBuffer.

                for (nNumeroPuntosAPT = 0; ((nIndex = strContFichero.Find(_T('\n'))) != -1);
nNumeroPuntosAPT++)
                {
                        nLongitudLinea = strContFichero.Find(_T('\n'));

// Extraemos las coordenadas del punto, separadas por coma.
// Cada punto en una línea (cogemos tambien '\n').
```

```
                        strBuffer += strContFichero.Left(nLongitudLinea+1);
                        strContFichero.Delete(0,nLongitudLinea+1);
                }
// Le añadimos los puntos que permitirán la maniobra de aproximación.
                TotalPuntosAPT = nNumeroPuntosAPT+pasos;

                pAplicacion->m_pPuntosTrayectoria = new CCaptura[TotalPuntosAPT];
                double dPuntoTrayectoria[6];
                int nIndex2, nIndex3;

                for (nIndex = 0; nIndex < (TotalPuntosAPT-pasos); nIndex++)
                {
                // Calculamos la longitud de la línea para poder trabajar
                // con ella.

                        nLongitudLinea = strBuffer.Find(_T('\n'));

                // Recorremos cada línea y vamos almacenado las coordenadas
                // en un array double[6].

                        for (nIndex2 = 0, nIndex3 = 0; nIndex2 < nLongitudLinea; nIndex2++)
                        {
                                if ((strBuffer[nIndex2] != ' ') && (strBuffer[nIndex2] != '\r'))
                                        strBuffer2 += strBuffer[nIndex2];
                                else
                                {
                                // Estamos en un caracter delimitador (bien el
                                // caracter ' ' o bien '\r'). Por tanto, lo que
                                // tenemos hasta ahora es una coordenada completa
                                // y debemos convertirla en un double y pasarla
                                // al array temporal.
                                        dPuntoTrayectoria[nIndex3] = _tcstod(strBuffer2, '\0');
                                        strBuffer2.Empty();
                                        pAplicacion-
>m_pPuntosTrayectoria[nIndex+pasos].m_sCoordTheta.theta[nIndex3]=dPuntoTrayectoria[nIndex3];
                                        nIndex3++;
                                }
                        }

                // Eliminamos la línea procesada correctamente.
                //strBuffer = strBuffer.Right(strBuffer.GetLength() - nLongitudLinea);
        //DA ERROR TB
                        strBuffer.Delete(0, nLongitudLinea + 1);
                }

                AproximacionPuntoInicial();
        }

// Si no hemos abierto el archivo "posiciones.txt" es porque no
// se ha llamado este programa desde la aplicación "corte3d.exe".
// Habrá que solicitar los datos

        else
        {
                fRes=TRUE;
```

```
// Al no corresponderse esta ejecución de trayectoria de corte
// con una trayectoria del Rx90 real, los tres puntos iniciales
// no importan, así que cogemos 3 puntos con los que sí funciona
// la aplicación.

                // Esquina superior derecha. p(0).

                m_vPuntos[0].x=113;
                m_vPuntos[0].y=713;
                m_vPuntos[0].z=400;

                // Esquina superior izquierda. p(1).

                m_vPuntos[1].x=-125;
                m_vPuntos[1].y=713;
                m_vPuntos[1].z=400;

                // Esquina inferior derecha. p(2).

                m_vPuntos[2].x=113;
                m_vPuntos[2].y=588;
                m_vPuntos[2].z=188;

                pAplicacion->m_MundoReal.CrearMatriz(m_vPuntos);

                // Esquina superior derecha. p(0).

                m_vPuntos[0].x=165.831;
                m_vPuntos[0].y=165.831;
                m_vPuntos[0].z=102.506;

                // Esquina superior izquierda. p(1).

                m_vPuntos[1].x=-165.831;
                m_vPuntos[1].y=165.831;
                m_vPuntos[1].z=102.506;

                // Esquina inferior derecha. p(2).

                m_vPuntos[2].x=-165.831;
                m_vPuntos[2].y=-165.831;
                m_vPuntos[2].z=102.506;

        // Ya tenemos almacenadas en vPuntos[3] los puntos p0,p1 y p2 en el
        // entorno virtual. Creamos la matriz de transformación homogénea.

                pAplicacion->m_MundoVirtual.CrearMatriz(m_vPuntos);

        // 3. Calculamos la matriz de transformación wHwv (transformación de {Wv} a {V},
        // es decir, nos da las coordenadas de cualquier punto respecto al sistema
        // de referencia WORLD del entorno real:
        // wHwv = wHp * (wvHpv)-1tal que          wP = wHwv * wvP

                pAplicacion->m_MundoVirtual.InvertirMatriz();
```

```
                pAplicacion->m_MundoVirtual.Componer(pAplicacion->m_MundoReal);


                CString strContenidoFicheroAPT;

                BOOL fRes = AbrirYLeerFicheroAPT(strContenidoFicheroAPT);

        // Si no han ocurrido errores, procesamos el fichero.
                if(fRes)
                        fRes = ProcesarContenidoFicheroAPT(strContenidoFicheroAPT);

        }

        if (fRes)
                func=1;
}

////////////////////////////////////////////////////////////////////////
//                                                                    //
//      La función "where()" es llamada cada vez que salta un evento  //
//      del temporizador de la clase CMainFrame. Lee las seis siguientes //
//      coordenadas articulares de la trayectoria y las impone a los  //
//      ejes del robot virtual.                                       //
//                                                                    //
////////////////////////////////////////////////////////////////////////

void CCaptura::where()
{
        CRx90App* pAplicacion = (CRx90App*)AfxGetApp();

        int j;
        double th[6];

// Leemos el siguiente punto de la trayectoria: 6 coordenadas en grados.

        for (j=0;j<6;j++)
                th[j]=pAplicacion->m_pPuntosTrayectoria[iteracion].m_sCoordTheta.theta[j]*PI/180;

// Escalamos los valores leídos pasándolos a radianes y dividiéndolos
// por un factor según el eje.

        Pos.giros[0]=th[0]-PI/2;
        Pos.giros[1]=(th[1]+PI/2)/2;
        Pos.giros[2]=(th[2]-PI)/3;
        Pos.giros[3]=th[3]/4;
        Pos.giros[4]=th[4]/5;
        Pos.giros[5]=th[5]/6;

// Aumentamos la variable iteración para en la próxima llamada a esta función
// leamos el punto siguiente.
        iteracion++;

// Imponemos los valores leídos en los ejes del robot,
        for (int i=0;i<6;i++)
                m_pFrame->SetPosition(Pos.giros[i],i);
```

```
}

////////////////////////////////////////////////////////////////////////////
//
// Método que sirve para abrir y leer el contenido del fichero APT.
//
// Parámetros:
//      - Referencia a CString para que guarde el contenido del fichero.
//
// Devuelve TRUE si no han existido errores y FALSE si ha habido.
// De esta forma no se ejecuta V_TRAJSIG si no es necesario.
//
////////////////////////////////////////////////////////////////////////////
BOOL CCaptura::AbrirYLeerFicheroAPT(CString& strContFicheroAPT)
{
        // Leyendo y escribiendo en un archivo de text ASCII:
        // Si usamos Unicode o MBCS hemos de tener cuidado cuando escribamos en archivos de texto
ASCII.
        // La manera mas fácil y segura es usar la clase CStdioFile proporcionada por MFC. Sólo
debemos usar
        // la clase CString y los métodos miembros ReadString y WriteString:
        //              CStdioFile file(..);
        //              CString str = _T("Hola");
        //              file.WriteString(str);
        //
        // Si hemos de usar la clase CFile forzósamente, lo haremos así:
        //              CFile file(..);
        //              CString str = _T("Hola");
        //              file.Write(str,(str.GetLength()+1)*sizeof(TCHAR));
        // Otro comentario importante es q siempre leamos y escribamos usando captura de
        // excepciones try&catch(CFileException *e).

        BOOL fRes = TRUE;

// Mostramos un diálogo modal que nos permite seleccionar el fichero APT a abrir.
        CFileDialog dlgAbrirAPT(TRUE,"dxf",NULL,OFN_FILEMUSTEXIST|OFN_HIDEREADONLY,"Datos APT de
CATIA (*.aptsource)|*.aptsource||");
        CString strNombreFichero;

        if(dlgAbrirAPT.DoModal() == IDOK)
        {
                strNombreFichero = dlgAbrirAPT.GetFileName( );
                CFile fileAPT;

                // Abrimos el fichero seleccionado.
                try
                {
                        fileAPT.Open(strNombreFichero, CFile::modeRead);
                }
                catch (CFileException *e)
                {
                        CString strError;
                        strError.Format("Error al abrir el archivo. Error número %d",e->m_cause);
                        e->Delete();
```

```
                AfxMessageBox(strError);
                return FALSE;
        }

        // Creamos un buffer temporal de lectura
        int nFileLength = fileAPT.GetLength();
        char *lpBuffer = new char[nFileLength +1];
        // Leemos todo el contenido del buffer del puerto serie y lo
        // almacenamos en nuestro buffer temporal
        try
        {
                fileAPT.Read(lpBuffer,nFileLength);
        }
        catch (CFileException *e)
        {
                CString strError;
                strError.Format("Error al leer el archivo. Error número %d",e->m_cause);
                AfxMessageBox(strError);
                e->Delete();
                delete[] lpBuffer;
                return FALSE;
        }

        // Añadimos un carácter NULL para terminar la cadena.
        lpBuffer[nFileLength] = '\0';

        // Cerramos el fichero.
        fileAPT.Close();

        // Almacenamos el contenido del fichero en una instancia de la clase CString, más
fácil de usar.
        strContFicheroAPT.Format(_T("%s"),lpBuffer);

        // Vaciamos y eliminamos el buffer temporal.
        delete[] lpBuffer;
    }

    else
        fRes =FALSE;

    return fRes;
}

//////////////////////////////////////////////////////////////////////////
//                                                                      //
// Método que sirve para procesar el contenido del fichero APT.         //
//                                                                      //
// Parámetros:                                                          //
//      - Referencia a CString para que contiene el contenido del fichero  //
//        APT, que va a ser retocado dentro de la función (podría ser   //
//        LPCTSTR xq no es retocado)                                    //
//                                                                      //
// Devuelve TRUE si todos los puntos son alcanzables,y FALSE si alguno  //
// no lo es. De esta forma no se ejecuta V_TRAJSIG si no es necesario.  //
//                                                                      //
```

```
//////////////////////////////////////////////////////////////////////////
BOOL CCaptura::ProcesarContenidoFicheroAPT(CString& strContFicheroAPT)
{
	int nIndex, nIndex2, nIndex3, nNumeroPuntosAPT, nLongitudLinea;
	CString strBuffer;
	BOOL fRes = TRUE;
	BOOL fRes2 = TRUE;

	// Contamos el número de puntos que hay en el fichero APT. Para
	// ello recorremos el objeto CString y vamos extrayendo las
	// líneas donde hay puntos y eliminado aquellas que no los
	// contienen. Los puntos extraidos se almacenan en strBuffer.

	for (nNumeroPuntosAPT = 0; ((nIndex = strContFicheroAPT.Find(_T("GOTO/"))) != -1);
nNumeroPuntosAPT++)
	{

		// Recorto la cadena hasta la primera coordenada del punto.
		// strContFicheroAPT = strContFicheroAPT.Right(strContFicheroAPT.GetLength() -
nIndex - sizeof(_T("GOTO/")));
		strContFicheroAPT.Delete(0, nIndex + sizeof(_T("GOTO/")));

		// Contamos la longitud de la línea para poder extraerla.
		nLongitudLinea = strContFicheroAPT.Find(_T('\n'));

		// Extraemos las coordenadas del punto, separadas por coma.
		// Cada punto en una línea (cogemos tambien '\n').
		strBuffer += strContFicheroAPT.Left(nLongitudLinea+1);
	}

	// Hacemos que las coordenadas de cada punto estén separadas por
	// un solo espacio en blanco, pues el formato que tenia era:
	// xXXX.XXXX, xXXX.XXXX, xXXX.XXXX ... donde X es un digito
	// y x es el signo - si el número es negativo.

	strBuffer.Remove(' ');
	strBuffer.Replace(',', ' ');

	// Eliminamos el primer punto del fichero APT, que no pertenece
	// a la trayectoria.
	// TENERLO EN CUENTA A LA HORA DE CALCULAR
	// nNumeroPuntosTray!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

	nIndex = strBuffer.Find('\n') + 1;
	strBuffer = strBuffer.Right(strBuffer.GetLength() - nIndex);
	nNumeroPuntosAPT--;

	TotalPuntosAPT=nNumeroPuntosAPT;

	// Ahora sabemos el número de instancias de la clase
	// CPuntoTrayectoria que necesitamos. Declaramos el array
	// donde almacenaremos los puntos definitivamente y los
	// inicializamos.

	CRx90App* pAplicacion = (CRx90App*)AfxGetApp();
```

```
pAplicacion->m_pPuntoTrayectoria = new CPuntoTrayectoria[nNumeroPuntosAPT];

// Para cada punto de la trayectoria...
double dPuntoTrayectoria[6];
CString strBuffer2;

for (nIndex = 0; nIndex < nNumeroPuntosAPT; nIndex++)
{
	// Calculamos la longitud de la línea para poder trabajar
	// con ella.
	nLongitudLinea = strBuffer.Find(_T('\n'));

	// Recorremos cada línea y vamos almacenado las coordenadas
	// en un array double[6].

	for (nIndex2 = 0, nIndex3 = 0; nIndex2 < nLongitudLinea; nIndex2++)
	{
		if ((strBuffer[nIndex2] != ' ') && (strBuffer[nIndex2] != '\r'))
			strBuffer2 += strBuffer[nIndex2];
		else
		{
			// Estamos en un caracter delimitador (bien el
			// caracter ' ' o bien '\r'). Por tanto, lo que
			// tenemos hasta ahora es una coordenada completa
			// y debemos convertirla en un double y pasarla
			// al array temporal.
			dPuntoTrayectoria[nIndex3] = _tcstod(strBuffer2, '\0');
			strBuffer2.Empty();
			nIndex3++;
		}
	}

	// Eliminamos la línea procesada correctamente.
	//strBuffer = strBuffer.Right(strBuffer.GetLength() - nLongitudLinea);
//DA ERROR TB
	strBuffer.Delete(0, nLongitudLinea + 1);

	// COMENTARIO ACLARATORIO EN MODO DE DEPURACIÓN
	TRACE(_T("\n\nzfree[1][%d]\n"),nIndex);

	// Hacemos las operaciones que haya que hacer en cada punto.
	// A) Rellenamos todas las coordenadas XYZvector.
	pAplicacion->m_pPuntoTrayectoria[nIndex].SetXYZvector(dPuntoTrayectoria);

	// B) Pasamos cada punto a su forma matricial.
	//pAplicacion->m_pPuntoTrayectoria[nIndex].SetXYZworld(pAplicacion->m_MundoReal);
	pAplicacion->m_pPuntoTrayectoria[nIndex].XYZvector2Matriz();
}

// Ahora ya tenemos cada punto del fichero APT expresado por
// medio de su matriz de transformación homogénea, todos ellos
// referidos respecto al origen del sistema de referencia
// del diseño del mundo virtual.

// Ahora ya tenemos preparados todos los puntos de la trayectoria
```

```
        // y podemos realizar sobre ellos las operaciones necesarias.
        for (nIndex = 0; nIndex < nNumeroPuntosAPT; nIndex++)
        {
                // COMENTARIO ACLARATORIO EN MODO DE DEPURACIÓN
                TRACE(_T("\n\nzfree[1][%d]\n"),nIndex);

                //1. Componemos cada punto respecto a la transformación wHwv.
                pAplicacion->m_pPuntoTrayectoria[nIndex].Componer(pAplicacion->m_MundoVirtual);


                //2. Resolvemos el modelo cinemático inverso en cada punto (con configuración -1,1,-
1)
                pAplicacion->m_pPuntoTrayectoria[nIndex].ResolverModeloCinematicoInverso(-1,1,-1);

                //3. Comprobamos si el punto es alcanzable. Si no lo es, indicamos el error.

                // SOLO PARA COMPROBAR.SE HACE LUEGO SI LA TRAYECTORIA ES CORRECTA
                pAplicacion->m_pPuntoTrayectoria[nIndex].Matriz2XYZeulerZYZ();

                fRes = pAplicacion->m_pPuntoTrayectoria[nIndex].EsAlcanzable();

                if (fRes == FALSE)
                {
                        CString strError;
                        strError.Format(_T("El punto %d no es alcanzable."), nIndex+1);
                        AfxMessageBox(strError,MB_ICONEXCLAMATION|MB_OK);
                        fRes2 = FALSE;
                }
        }

return fRes2;
}

////////////////////////////////////////////////////////////////////////
//                                                                    //
//      Función que calcula el punto central de la trayectoria de corte    //
//      para posicionar la pieza objetivo en la escena.               //
//      Parámetros:                                                   //
//              - int NPuntos: número de puntos que conforman la      //
//                trayectoria.                                        //
//                                                                    //
////////////////////////////////////////////////////////////////////////

void CCaptura::CalculaPuntoCentral(int NPuntos)
{
        CRx90App* pAplicacion = (CRx90App*)AfxGetApp();
//      double diferencia;
        double MaxDistancia=0;
///     Vector posicion_inicial;
        Vector posicion;
        double th[6];
        int d6=360;
        int a2=450;
        int d4=450;
//      Vector distancia;
```

```
        Vector PuntoCentral;
        Vector maximos;
        Vector minimos;

        for (int i=0;i<NPuntos;i++)
        {
                for (int j=0;j<6;j++)
                        th[j]=pAplicacion->m_pPuntosTrayectoria[i].m_sCoordTheta.theta[j]/180*PI;

// Ya tenemos en th[i] las coordenaddas articulares en RADIANES de el punto [i].
// Ahora aplicamos el modelo directo para obtener sus coordenadas cartesianas
// respecto al sistema World.
                // X=((c1*c23*c4-s1*s4)*s5+c1*s23*c5)*d6+c1*s23*d4+c1*c2*a2;
                posicion.x=((cos(th[0])*cos(th[1]+th[2])*cos(th[3])-
sin(th[0])*sin(th[3]))*sin(th[4])+cos(th[0])*sin(th[1]+th[2])*cos(th[4]))*d6+cos(th[0])*sin(th[1]+th[
2])*d4+cos(th[0])*cos(th[1])*a2;

                // Y=((s1*c23*c4+c1*s4)*s5+s1*s23*c5)*d6+s1*s23*d4+s1*c2*a2;
        posicion.y=((sin(th[0])*cos(th[1]+th[2])*cos(th[3])+cos(th[0])*
sin(th[3]))*sin(th[4])+sin(th[0])*sin(th[1]+th[2])*cos(th[4]))*d6+sin(th[0])*
sin(th[1]+th[2])*d4+sin(th[0])*cos(th[1])*a2;

                // Z=-(s23*c4*s5-c23*c5)*d6+c23*d4-s2*a2+d1;
                posicion.z=-(sin(th[1]+th[2])*cos(th[3])*sin(th[4])-
cos(th[1]+th[2])*cos(th[4]))*d6+cos(th[1]+th[2])*d4-sin(th[1])*a2;

                if (i==0)
                {
                        maximos.x=minimos.x=posicion.x;
                        maximos.y=minimos.y=posicion.y;
                        maximos.z=minimos.z=posicion.z;
                }
                else
                {
// Comprobamos si el punto leído es un máximo o un mínimo de la trayectoria.
                        if (posicion.x>maximos.x) maximos.x=posicion.x;
                        else if (posicion.x<minimos.x) minimos.x=posicion.x;

                        if (posicion.y>maximos.y) maximos.y=posicion.y;
                        else if (posicion.y<minimos.y) minimos.y=posicion.y;

                        if (posicion.z>maximos.z) maximos.z=posicion.z;
                        else if (posicion.z<minimos.z) minimos.z=posicion.z;
                }
        }

        PuntoCentral.x=(maximos.x+minimos.x)/2;
        PuntoCentral.y=(maximos.y+minimos.y)/2;
        PuntoCentral.z=(maximos.z+minimos.z)/2;

//      CString msg;
//      msg.Format("Punto Central= [%f,%f,%f].",PuntoCentral.x,PuntoCentral.y,PuntoCentral.z);
//      AfxMessageBox(msg);

        ofstream out("puntocentral.txt");
```

```
        out << PuntoCentral.x << " " << PuntoCentral.y << " " << PuntoCentral.z << "\n";
        out.close();
}

////////////////////////////////////////////////////////////////////////
//                                                                    //
//      Función que lee la posición actual del robot y la posición inicial //
//      de la trayectoria, e interpone entre ellos tantos puntos como      //
//      indique "pasos". Con ello se consigue un efecto de suavidad en el  //
//      desplazamiento hacia la primera posición de la trayectoria.        //
//                                                                    //
////////////////////////////////////////////////////////////////////////

void CCaptura::AproximacionPuntoInicial()
{
        double PosicionActual[6];
        double PosicionDestino[6];
        double salto[6];

        CRx90App* pAplicacion = (CRx90App*)AfxGetApp();

//      Leemos la posición actual y la primera posición de la trayectoria.
//      La primera se lee en radianes, y la segunda en grados.
        for (int i=0;i<6;i++)
        {
                PosicionActual[i]=m_pFrame->m_pRobot->Posicionact.Artic[i];
                PosicionDestino[i]=pAplicacion->m_pPuntosTrayectoria[pasos].m_sCoordTheta.theta[i];
        }

// Escalamos los valores de la posicion actual para poder trabajar
// con ellos.

        PosicionActual[0]=PosicionActual[0]*180/PI+90;
        PosicionActual[1]=(PosicionActual[1]*180/PI)*2-90;
        PosicionActual[2]=(PosicionActual[2]*180/PI)*3+180;
        PosicionActual[3]=(PosicionActual[3]*180/PI)*4;
        PosicionActual[4]=(PosicionActual[4]*180/PI)*5;
        PosicionActual[5]=(PosicionActual[5]*180/PI)*6;

// Calculamos el salto que debe salvarse en cada articulación.
        for (i=0;i<6;i++)
                salto[i]=PosicionDestino[i]-PosicionActual[i];

        for (i=0;i<pasos;i++)
                for (int j=0;j<6;j++)
                {
                        pAplicacion-
>m_pPuntosTrayectoria[i].m_sCoordTheta.theta[j]=PosicionActual[j]+salto[j]*i/pasos;
                }

// Dividimos la diferencia entre 10 para hacer la aproximacion
// en 10 pasos.

}
```

B.2.10. CPuntoTrayectoria.

B.2.10.1. PuntoTrayectoria.h

```
////////////////////////////////////////////////////////////////////////
//                                                                    //
//      Módulo : PuntoTrayectoria.h                                   //
//                      Definición de la clase CPuntoTrayectoria.     //
//                                                                    //
//      Propósito:      Esta clase contiene las estructuras y operaciones //
//                      necesarias para trabajar con los puntos de las //
//                      trayectorias: permite almacenar los puntos en //
//                      diversos formatos (XYZvector, XYZeulerZYZ, theta[i] //
//                      {i=1..6},matriz de transf. homogénea), realizar //
//                      conversiones entre       ellos y ejecutar las operaciones //
//                      necesarias para cada formato.                 //
//                                                                    //
//      Creado:         Manuel Gómez Langley / 2003-2004              //
//      Modificado: Alejandro Guija Rodríguez / 2006-2007             //
//                                                                    //
////////////////////////////////////////////////////////////////////////

#if !defined(AFX_PUNTOTRAYECTORIA_H__A2D3E6B5_D0C6_47DD_B3F7_C3A4D9C4A6D8__INCLUDED_)
#define AFX_PUNTOTRAYECTORIA_H__A2D3E6B5_D0C6_47DD_B3F7_C3A4D9C4A6D8__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

// Estructura que se utiliza para almacenar las coordenadas cartesianas de un punto.
struct Vector
{
        double x;
        double y;
        double z;
};
//      http://www.vjuegos.org/modules.php?name=Content&pa=showpage&pid=39


//////////////////////////// Clases ///////////////////////////////////////////
/*

//AÑADIR CLASES VECTOR Y MATRIZ, en este mismo fichero valdría.

class CVector
{

////////////////////////////////////////////////////////////////////////////
// Enumeraciones
////////////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////////////
// Variables miembro
```

```
////////////////////////////////////////////////////////////////////////////
protected:
        double X;
        double Y;
        double Z;

////////////////////////////////////////////////////////////////////////////
// Constructor/Destructor
////////////////////////////////////////////////////////////////////////////

public:
        CVector();
        virtual ~CVector();


////////////////////////////////////////////////////////////////////////////
// Métodos miembro
////////////////////////////////////////////////////////////////////////////

        double Modulo();
};

class CMatriz
{

////////////////////////////////////////////////////////////////////////////
// Enumeraciones
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
// Variables miembro
////////////////////////////////////////////////////////////////////////////
protected:
        // Doble array que almacena la localización y orientación del extremo
        // del robot en forma de matriz de transformación homogénea (4x4).
        // Se utiliza para realizar la traslación de cada punto de la
        // trayectoria virtual respecto del punto inicial donde se encuentra la
        // herramienta en la realidad para obtener la trayectoria real.
        double m_dMatriz[4][4];

////////////////////////////////////////////////////////////////////////////
Constructor/Destructor
////////////////////////////////////////////////////////////////////////////

public:
        CMatriz();
        virtual ~CMatriz();

////////////////////////////////////////////////////////////////////////////
// Métodos miembro
////////////////////////////////////////////////////////////////////////////

        void Invertir();
        CMatriz& Premultiplicar(const CMatriz& matriz);
        SetMatriz(CVector* puntos);
```

```
};
*/
class CPuntoTrayectoria
{

////////////////////////////////////////////////////////////////////////////
// Enumeraciones
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
// Variables miembro
////////////////////////////////////////////////////////////////////////////

protected:

        // Estructura que almacena un punto en un formato X,Y,Z,vector, que es
        // el que contienen los ficheros APT.
        // En este formato, los 3 primeros valores son las coordenadas del
        // extremo de la garra en el espacio, mientras que los otros 3 valores
        // nos informan de la orientación de la herramienta a través de un
        // vector unitario.
        struct coordXYZvector
        {
                double X;
                double Y;
                double Z;
                double i;
                double j;
                double k;
        } m_sCoordXYZvector;

        struct coordXYZvectorworld
        {
                double X;
                double Y;
                double Z;
                double i;
                double j;
                double k;
        } m_sCoordXYZvectorworld;

        // Estructura que almacena un punto en un formato X,Y,Z,euler-ZYZ, que
        // es el que utiliza V_TRAJSIG.
        // En este formato, los 3 primeros valores son las coordenadas del
        // extremo de la garra en el espacio, mientras que los otros 3 valores
        // nos informan de la orientación de la garra a través de un sistema de
        // referencia.
        // Los ángulos ZYZ de Euler definen la orientación mediante 3 giros
        // consecutivos respecto a los ejes ZYZ de un sistema de referencia {B}
        // solidario al cuerpo. Se comienza con {B} coincidente con el sistema
        // de referencia {A}. Sucesivamente se realizan 3 giros:
        //      -       Primero se produce una rotación de {B} un ángulo ALFA
        //              alrededor del eje Zb, que en este caso es coincidente con
        //              el eje Z origen (Za).
        //      -       Después se produce una rotación de un ángulo BETA alrededor
```

```
        //              del nuevo eje Yb.
        //      -       Finalmente se produce una rotación de un ángulo GAMMA
        //              alrededor del    nuevo eje Zb.
        struct coordXYZeulerZYZ
        {
                double X;
                double Y;
                double Z;
                double eulerZalfa;
                double eulerYbeta;
                double eulerZgamma;
        } m_sCoordXYZeulerZYZ;

        // Variable estática que se utiliza para almacenar el valor del ángulo
        // alfa de EulerZYZ del punto anterior de la trayectoria.
        static double m_dEulerZalfaPuntoAnterior;

        // Estructura estática que se utiliza para almacenar el valor de los
        // parámetros de Denavit-Hartemberg del robot RX-90.
        struct parametrosDHRobot
        {
                double d1;
                double a2;
                double a3;
                double d3;
                double d4;
                double d6;
        } static m_sParam;


        // Constante estática que almacena el valor de PI.
        //      const double PI = 3.14159265358979310862446 8;
        // Constante estática que almacena el valor de la precision para 0.
        //      const double CERO = 0.00001;

////////////////////////////////////////////////////////////////////////////
// Constructor/Destructor
////////////////////////////////////////////////////////////////////////////

public:
        CPuntoTrayectoria();
        virtual ~CPuntoTrayectoria();

////////////////////////////////////////////////////////////////////////////
// Métodos miembro
////////////////////////////////////////////////////////////////////////////

public:
        void SetXYZworld(const CPuntoTrayectoria& Inicio);
        void XYZeulerZYZ2Matriz();

        void XYZvector2XYZeulerZYZ();                                   // Es la
función zyz() de M.Vargas.
        void XYZvector2Matriz();
        void Matriz2XYZeulerZYZ();
```

```
        void Componer(const CPuntoTrayectoria& Inicio);
// Definimos asi la función xq es mas efectivo pasar objetos por referencia
// (no se llaman constructores ni copy constructors... Definimos el parámetro
// const para que no se modifique dentro de la función. Así tenemos un objeto
// localmente constante dentro de la función.

        void ResolverModeloCinematicoInverso(int nBrazo, int nCodo, int nMuneca);
        BOOL EsAlcanzable();

        void SetXYZvector(double* dCoordenadas);
        void SetXYZeulerZYZ(double* dCoordenadas);
        void GetXYZeulerZYZ(double* dCoordenadas);

        void CrearMatriz(Vector* vPuntos);
        void InvertirMatriz();

        // Doble array que almacena la localización y orientación del extremo
        // del robot en forma de matriz de transformación homogénea (4x4).
        // Se utiliza para realizar la traslación de cada punto de la
        // trayectoria virtual respecto del punto inicial donde se encuentra la
        // herramienta en la realidad para obtener la trayectoria real.
        double m_dMatriz[4][4];

        // Estructura que almacena la localización de un punto en coordenadas
        // articulares del robot.
        // Se utiliza al resolver el modelo cinemático inverso para comprobar si
        // el punto es alcanzable.
        struct coordTheta
        {
                double theta[6];
        } m_sCoordTheta;

        // Como definimos un array de CPuntoTrayectoria con new, no podemos usar
        // el constructor para inicializar, asi q usamos estas funciones para
        // asignar los valores iniciales.
};

#endif // !defined(AFX_PUNTOTRAYECTORIA_H__A2D3E6B5_D0C6_47DD_B3F7_C3A4D9C4A6D8__INCLUDED_)
```

B.2.10.2. PuntoTrayectoria.cpp

```
////////////////////////////////////////////////////////////////////////////
//
//      Módulo : PuntoTrayectoria.cpp
//                      Implementación de la clase CPuntoTrayectoria.
//
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
// Incluimos el fichero de cabecera estándar y el fichero de cabecera
// necesario para realizar operaciones matemáticas.
////////////////////////////////////////////////////////////////////////////

#include "stdafx.h"
```

```
#include <math.h>

//////////////////////////////////////////////////////////////////////////
// Incluimos los ficheros de cabecera del módulo y del programa principal
//////////////////////////////////////////////////////////////////////////

#include "PuntoTrayectoria.h"
#include "Rx90.h"

//////////////////////////////////////////////////////////////
// Habilitamos el 'debug memory manager'                    //
//////////////////////////////////////////////////////////////

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////
// Definición e inicialización de miembros estáticos       //
//////////////////////////////////////////////////////////
double CPuntoTrayectoria::m_dEulerZalfaPuntoAnterior = 0;
struct CPuntoTrayectoria::parametrosDHRobot CPuntoTrayectoria::m_sParam = {450, 0, 0, 450, 85};

//////////////////////////////////////////////////////
// Código                                                  //
//////////////////////////////////////////////////////

CPuntoTrayectoria::CPuntoTrayectoria()//:
{
        m_dMatriz[0][0] = 0;
        m_dMatriz[0][1] = 0;
        m_dMatriz[0][2] = 0;
        m_dMatriz[0][3] = 0;
        m_dMatriz[1][0] = 0;
        m_dMatriz[1][1] = 0;
        m_dMatriz[1][2] = 0;
        m_dMatriz[1][3] = 0;
        m_dMatriz[2][0] = 0;
        m_dMatriz[2][1] = 0;
        m_dMatriz[2][2] = 0;
        m_dMatriz[2][3] = 0;
        m_dMatriz[3][0] = 0;
        m_dMatriz[3][1] = 0;
        m_dMatriz[3][2] = 0;
        m_dMatriz[3][3] = 1;
}

CPuntoTrayectoria::~CPuntoTrayectoria()
{
}

//////////////////////////////////////////////////////////////////////
//                                                                  //
```

```
// Método miembro que sirve para calcular la matriz de transformación       //
// homogénea de un punto a partir de las coordenadas XYZeulerZYZ.           //
//                                                                          //
//////////////////////////////////////////////////////////////////////////////
void CPuntoTrayectoria::XYZeulerZYZ2Matriz()
{
        const double PI = 3.14159265358979310862446 8;

        // Calculamos el valor de las funciones trigonométricas una sola vez
        // para ahorrar tiempo de ejecución.
        double dSenoAlfa = sin(m_sCoordXYZeulerZYZ.eulerZalfa*PI/180);
        double dCosenoAlfa = cos(m_sCoordXYZeulerZYZ.eulerZalfa*PI/180);
        double dSenoBeta = sin(m_sCoordXYZeulerZYZ.eulerYbeta*PI/180);
        double dCosenoBeta = cos(m_sCoordXYZeulerZYZ.eulerYbeta*PI/180);
        double dSenoGamma = sin(m_sCoordXYZeulerZYZ.eulerZgamma*PI/180);
        double dCosenoGamma = cos(m_sCoordXYZeulerZYZ.eulerZgamma*PI/180);

        // Calculamos el valor de los elementos de la matriz de rotación.
        m_dMatriz[0][0] = dCosenoAlfa*dCosenoBeta*dCosenoGamma - dSenoAlfa*dSenoGamma;
        m_dMatriz[0][1] = -(dCosenoAlfa*dCosenoBeta*dSenoGamma) - dSenoAlfa*dCosenoGamma;
        m_dMatriz[0][2] = dCosenoAlfa*dSenoBeta;

        m_dMatriz[1][0] = dSenoAlfa*dCosenoBeta*dCosenoGamma + dCosenoAlfa*dSenoGamma;
        m_dMatriz[1][1] = -(dSenoAlfa*dCosenoBeta*dSenoGamma) + dCosenoAlfa*dCosenoGamma;
        m_dMatriz[1][2] = dSenoAlfa*dSenoBeta;

        m_dMatriz[2][0] = -(dSenoBeta*dCosenoGamma);
        m_dMatriz[2][1] = dSenoBeta*dSenoGamma;
        m_dMatriz[2][2] = dCosenoBeta;

        // Fijamos el valor de los elementos del vector de traslación.
        m_dMatriz[0][3] = m_sCoordXYZeulerZYZ.X;
        m_dMatriz[1][3] = m_sCoordXYZeulerZYZ.Y;
        m_dMatriz[2][3] = m_sCoordXYZeulerZYZ.Z;

//      CString strMsg;
//      strMsg.Format("Matriz: \n%f %f %f %f\n%f %f %f %f\n%f %f %f %f\n%f %f %f
%f\n",m_dMatriz[0][0],m_dMatriz[0][1],m_dMatriz[0][2],m_dMatriz[0][3],m_dMatriz[1][0],m_dMatriz[1][1]
,m_dMatriz[1][2],m_dMatriz[1][3],m_dMatriz[2][0],m_dMatriz[2][1],m_dMatriz[2][2],m_dMatriz[2][3],m_dM
atriz[3][0],m_dMatriz[3][1],m_dMatriz[3][2],m_dMatriz[3][3]);
//      AfxMessageBox(strMsg);

        TRACE(_T("Matriz de transformación homogénea:\n%f %f %f %f\n%f %f %f %f\n%f %f %f %f\n%f %f
%f %f\n"),
                                m_dMatriz[0][0],m_dMatriz[0][1],m_dMatriz[0][2],m_dMatriz[0][3],
                                m_dMatriz[1][0],m_dMatriz[1][1],m_dMatriz[1][2],m_dMatriz[1][3],
                                m_dMatriz[2][0],m_dMatriz[2][1],m_dMatriz[2][2],m_dMatriz[2][3],
                                m_dMatriz[3][0],m_dMatriz[3][1],m_dMatriz[3][2],m_dMatriz[3][3]);
}

//////////////////////////////////////////////////////////////////////////////
//                                                                          //
// Método miembro que sirve para calcular las coordenadas XYZeulerZYZ de    //
// un punto a partir de las coordenadas XYZvector.                          //
// Empleamos el método seguido en la función zyz() de M.Vargas.             //
```

```
//                                                                          //
//////////////////////////////////////////////////////////////////////////////
void CPuntoTrayectoria::XYZvector2XYZeulerZYZ()
{
        double dAux, dSenoAux, dCosenoAux, dAux2, dDifAux, dDifAux2;
        const double PI = 3.14159265358979310862468;

        // Primero almacenamos el vector de translación.
        m_sCoordXYZeulerZYZ.X = m_sCoordXYZvector.X;
        m_sCoordXYZeulerZYZ.Y = m_sCoordXYZvector.Y;
        m_sCoordXYZeulerZYZ.Z = m_sCoordXYZvector.Z;

        // Ahora calculamos el valor de los ángulos que nos dan la orientación de la herramienta.
        dAux = atan2(m_sCoordXYZvector.j,m_sCoordXYZvector.i);
        dSenoAux = sin(dAux);
        dCosenoAux = cos(dAux);

        if (fabs(dSenoAux) > 0.5)
                dAux2 = m_sCoordXYZvector.j / dSenoAux;
        else
                dAux2 = m_sCoordXYZvector.i / dCosenoAux;

        m_sCoordXYZeulerZYZ.eulerZalfa = dAux * 180/PI;
        m_sCoordXYZeulerZYZ.eulerYbeta = (atan2(dAux2,m_sCoordXYZvector.k)) * 180/PI;
        m_sCoordXYZeulerZYZ.eulerZgamma = 0;

        dAux = m_sCoordXYZeulerZYZ.eulerZalfa;
        dAux2 = m_sCoordXYZeulerZYZ.eulerZalfa + 180;

        if (dAux2 > 180)
                dAux2 -= 360;

        dDifAux = dAux - m_dEulerZalfaPuntoAnterior;
        dDifAux2 = dAux2 - m_dEulerZalfaPuntoAnterior;

        if (dDifAux < -180)
                dDifAux += 360;
        if (dDifAux > 180)
                dDifAux -= 360;

        if (dDifAux2 < -180)
                dDifAux2 += 360;
        if (dDifAux2 > 180)
                dDifAux2 -= 360;

        if (fabs(dDifAux2) < fabs(dDifAux))
        {
                m_sCoordXYZeulerZYZ.eulerZalfa = dAux2;
                m_sCoordXYZeulerZYZ.eulerYbeta = -m_sCoordXYZeulerZYZ.eulerYbeta;
        }

        // Almacenamos el valor del ángulo alfa para ser utilizado por el punto siguiente.
        m_dEulerZalfaPuntoAnterior = m_sCoordXYZeulerZYZ.eulerZalfa;

        TRACE(_T("Coordenadas XYZeulerZYZ: %f %f %f %f %f %f\n"),
```

```
                        m_sCoordXYZeulerZYZ.X, m_sCoordXYZeulerZYZ.Y, m_sCoordXYZeulerZYZ.Z,
                        m_sCoordXYZeulerZYZ.eulerZalfa, m_sCoordXYZeulerZYZ.eulerYbeta,
m_sCoordXYZeulerZYZ.eulerZgamma);
}



////////////////////////////////////////////////////////////////////////
//                                                                    //
// Método miembro que sirve para calcular la matriz de transformación //
// homogénea de un punto a partir de las coordenadas XYZvector.       //
//                                                                    //
////////////////////////////////////////////////////////////////////////
void CPuntoTrayectoria::XYZvector2Matriz()
{
        XYZvector2XYZeulerZYZ();
        XYZeulerZYZ2Matriz();
}

////////////////////////////////////////////////////////////////////////
//                                                                    //
// Método miembro que sirve para calcular las coordenadas XYZeulerZYZ //
// de un punto a partir de su matriz de transformación homogénea.     //
//                                                                    //
// Utilizamos el método seguido en HEMERO (distinto al del libro de   //
// Robótica).                                                         //
//      function euler = tr2eul(m)                                    //
//              euler = zeros(1,3);                                   //
//              if abs(m(2,3)) > eps & abs(m(1,3)) > eps,             //
//                      euler(1) = atan2(m(2,3), m(1,3));             //
//                      sp = sin(euler(1));                           //
//                      cp = cos(euler(1));                           //
//                      euler(2)=atan2(cp*m(1,3) + sp*m(2,3), m(3,3) ;//
//                      euler(3)=atan2(-sp*m(1,1)+cp*m(2,1),-sp*m(1,2)+//
//                                              + cp*m(2,2));         //
//              else,                                                 //
//                      euler(1) = 0;                                 //
//                      euler(2) = atan2(m(1,3), m(3,3));             //
//                      euler(3) = atan2(m(2,1), m(2,2));             //
//      end                                                           //
//                                                                    //
////////////////////////////////////////////////////////////////////////
void CPuntoTrayectoria::Matriz2XYZeulerZYZ()
{
        // Definimos las constante que almacenan los valores de la precision para 0 y de pi.
        const double CERO = 0.0001;
        const double PI = 3.14159265358979310862446 8;

        // Primero almacenamos el valor del vector de translación.
        m_sCoordXYZeulerZYZ.X = m_dMatriz[0][3];
        m_sCoordXYZeulerZYZ.Y = m_dMatriz[1][3];
        m_sCoordXYZeulerZYZ.Z = m_dMatriz[2][3];

        // Ahora calculamos los ángulos ZYZ de euler.
        if ((fabs(m_dMatriz[1][2]) > CERO) && (fabs(m_dMatriz[0][2]) > CERO))
```

```
        {
                m_sCoordXYZeulerZYZ.eulerZalfa = atan2(m_dMatriz[1][2], m_dMatriz[0][2]);
                double dSenoAlfa = sin(m_sCoordXYZeulerZYZ.eulerZalfa);
                double dCosenoAlfa = cos(m_sCoordXYZeulerZYZ.eulerZalfa);

                m_sCoordXYZeulerZYZ.eulerYbeta = atan2(dCosenoAlfa*m_dMatriz[0][2] +
dSenoAlfa*m_dMatriz[1][2], m_dMatriz[2][2]);
                m_sCoordXYZeulerZYZ.eulerZgamma = atan2(-dSenoAlfa*m_dMatriz[0][0] +
dCosenoAlfa*m_dMatriz[1][0], -dSenoAlfa*m_dMatriz[0][1] + dCosenoAlfa*m_dMatriz[1][1]);
        }
        else
        {
                m_sCoordXYZeulerZYZ.eulerZalfa = 0;
                m_sCoordXYZeulerZYZ.eulerYbeta = atan2(m_dMatriz[0][2], m_dMatriz[2][2]);
                m_sCoordXYZeulerZYZ.eulerZgamma = atan2(m_dMatriz[1][0], m_dMatriz[1][1]);
        }

        // Pasamos los ángulos a grados.
        m_sCoordXYZeulerZYZ.eulerZalfa *= (180/PI);
        m_sCoordXYZeulerZYZ.eulerYbeta *= (180/PI);
        m_sCoordXYZeulerZYZ.eulerZgamma *= (180/PI);

        TRACE(_T("Coordenadas XYZeulerZYZ: %f %f %f %f %f %f\n"),
                        m_sCoordXYZeulerZYZ.X, m_sCoordXYZeulerZYZ.Y,
                        m_sCoordXYZeulerZYZ.Z,m_sCoordXYZeulerZYZ.eulerZalfa,
                        m_sCoordXYZeulerZYZ.eulerYbeta, m_sCoordXYZeulerZYZ.eulerZgamma);
}

////////////////////////////////////////////////////////////////////////////
//                                                                        //
// Método miembro que sirve para componer un punto respecto a otro.       //
// Para ello hace una multiplicación de las matrices de transformación    //
// homogénea de los 2 puntos.                                             //
// Debemos premultiplicar la matriz de transformación homogénea de esta   //
// instancia de la clase CPuntoTrayectoria por la matriz de la instancia  //
// pasada como parámetro. Lo hacemos elemento a elemento.                 //
//                                                                        //
// Parámetros:                                                            //
//      - Referencia constante a CPuntoTrayectoria que contiene el punto  //
//        respecto al cual queremos componer el nuestro.                  //
//                                                                        //
////////////////////////////////////////////////////////////////////////////
void CPuntoTrayectoria::Componer(const CPuntoTrayectoria& Inicio)
{
        double dMatrizAux[4][4];
        int nIndex, nIndex2;

        for (nIndex = 0; nIndex < 4; nIndex++)
                for (nIndex2 = 0; nIndex2 < 4; nIndex2++)
                        dMatrizAux[nIndex][nIndex2] =
Inicio.m_dMatriz[nIndex][0]*m_dMatriz[0][nIndex2]
                                                                                +
Inicio.m_dMatriz[nIndex][1]*m_dMatriz[1][nIndex2]
                                                                                +
Inicio.m_dMatriz[nIndex][2]*m_dMatriz[2][nIndex2]
```

```
                                                                                +
Inicio.m_dMatriz[nIndex][3]*m_dMatriz[3][nIndex2];

        for (nIndex = 0; nIndex < 4; nIndex++)
                for (nIndex2 = 0; nIndex2 < 4; nIndex2++)
                        m_dMatriz[nIndex][nIndex2]=dMatrizAux[nIndex][nIndex2];

//      CString strMsg;
//      strMsg.Format("Matriz: \n%f %f %f %f\n%f %f %f %f\n%f %f %f %f\n%f %f %f
%f\n",m_dMatriz[0][0],m_dMatriz[0][1],m_dMatriz[0][2],m_dMatriz[0][3],m_dMatriz[1][0],m_dMatriz[1][1]
,m_dMatriz[1][2],m_dMatriz[1][3],m_dMatriz[2][0],m_dMatriz[2][1],m_dMatriz[2][2],m_dMatriz[2][3],m_dM
atriz[3][0],m_dMatriz[3][1],m_dMatriz[3][2],m_dMatriz[3][3]);
//      AfxMessageBox(strMsg);


        TRACE(_T("Matriz de transformación homogénea compuesta:\n[%f %f %f %f;\n%f %f %f %f;\n%f %f
%f %f;\n%f %f %f %f];\n"),
                                m_dMatriz[0][0],m_dMatriz[0][1],m_dMatriz[0][2],m_dMatriz[0][3],
                                m_dMatriz[1][0],m_dMatriz[1][1],m_dMatriz[1][2],m_dMatriz[1][3],
                                m_dMatriz[2][0],m_dMatriz[2][1],m_dMatriz[2][2],m_dMatriz[2][3],
                                m_dMatriz[3][0],m_dMatriz[3][1],m_dMatriz[3][2],m_dMatriz[3][3]);
}


////////////////////////////////////////////////////////////////////////////
//                                                                        //
// Método que sirve para realizar la resolución del problema cinemático   //
// inverso para el robot Stäubli RX90 en el punto almacenado.             //
// Para ello utiliza los valores almacenados en la matriz de transformación //
// homogénea y devuelve el resultado en coordenadas articulares.          //
//                                                                        //
// Parámetros:                                                            //
//      - nBrazo: variable que especifica brazo izquierdo o derecho.      //
//  - nCodo: variable que especifica codo arriba o abajo.                 //
//  - nMuneca: variable que especifica giro de muñeca en sentido horario  //
//      o antihorario.                                                    //
//                                                                        //
// Función creada por Alejandro Guija.                                    //
////////////////////////////////////////////////////////////////////////////
void CPuntoTrayectoria::ResolverModeloCinematicoInverso(int nBrazo, int nCodo, int nMuneca)
{
        double dX = m_dMatriz[0][3];
        double dY = m_dMatriz[1][3];
        double dZ = m_dMatriz[2][3];

        double dSeno[6] = {0,0,0,0,0,0};
        double dCoseno[6] = {0,0,0,0,0,0};
        double theta23 = 0;
        double dSeno23 = 0;
        double dCoseno23= 0;

        double dK, dAux1, dAux2, dAux3, dAux4, dAux5;

        const double PI = 3.14159265358979310862 4468;
```

```
        //      El parámetro d6 lo hemos metido con un valor aproximado
        //  Según el proyecto de Manolo es de unos 275 mm;
        m_sParam.d6=360;
        m_sParam.d1=0;
        m_sParam.a2=450;
        m_sParam.d4=450;
        m_sParam.a3=0;
        m_sParam.d3=0;

        // Deducción del tamaño de la herramienta
        // Si quito esto no funciona el método simplificado.
        // Desactivado aki xq ya lo hacemos en main.
        dX = dX - m_sParam.d6*m_dMatriz[0][2];
        dY = dY - m_sParam.d6*m_dMatriz[1][2];
        dZ = dZ - m_sParam.d6*m_dMatriz[2][2];

//      dX = dX - m_sParam.d6*m_sCoordXYZvectorworld.i;
//      dY = dY - m_sParam.d6*m_sCoordXYZvectorworld.j;
//      dZ = dZ - m_sParam.d6*m_sCoordXYZvectorworld.k;

        dK=(dX*dX + dY*dY + dZ*dZ - m_sParam.a2*m_sParam.a2 - m_sParam.a3*m_sParam.a3 -
m_sParam.d3*m_sParam.d3 - m_sParam.d4*m_sParam.d4)/(2*m_sParam.a2);
//      cout<<"\nLocalizacion sin herramienta:"<<dK<<"\n";

        // Cálculo de ángulos de las articulaciones

        // Calculamos theta1
        //m_sCoordTheta.theta[0] = atan2(-nBrazo*dY,-nBrazo*dX);
        m_sCoordTheta.theta[0] = atan2(dY,dX);
        dSeno[0] = sin(m_sCoordTheta.theta[0]);
        dCoseno[0] = cos(m_sCoordTheta.theta[0]);

/*      // Calculamos theta2 según Manolo Gomez
        dAux1 = sqrt(dX*dX + dY*dY + dZ*dZ);
        dAux2 = sqrt(dX*dX + dY*dY);
        dAux3 = (dX*dX + dY*dY + dZ*dZ + m_sParam.a2*m_sParam.a2 -
m_sParam.d4*m_sParam.d4)/(2*m_sParam.a2*dAux1);
        dAux4 = -dZ*dAux3/dAux1 + nBrazo*nCodo*(-nBrazo*dAux2/dAux1)*sqrt(1 - dAux3*dAux3);
        dAux5 = (-nBrazo*dAux2*dAux3/dAux1) - nBrazo*nCodo*(-dZ/dAux1)*sqrt(1 - dAux3*dAux3);

        m_sCoordTheta.theta[1] = atan2(dAux4, dAux5);
        dSeno[1] = sin(m_sCoordTheta.theta[1]);
        dCoseno[1] = cos(m_sCoordTheta.theta[1]);
*/
        // Calculo de theta2 según Ale
        dAux1=sqrt(dX*dX+dY*dY);
        dAux2=dX*dX+dY*dY+dZ*dZ;
        dAux3=atan2(dZ,dAux1);
        dAux4=acos((m_sParam.d4*m_sParam.d4-dAux2-m_sParam.a2*m_sParam.a2)/(-
2*sqrt(dAux2)*m_sParam.a2));
        m_sCoordTheta.theta[1]=-(dAux3+dAux4);
        dSeno[1]=sin(m_sCoordTheta.theta[1]);
        dCoseno[1]=cos(m_sCoordTheta.theta[1]);

/*      // Calculamos theta3 según Manolo
```

```
        dAux1 = dK/m_sParam.d4;
        dAux2 = nBrazo*nCodo*sqrt(1 - dAux1*dAux1);

        m_sCoordTheta.theta[2] = atan2(dAux1, dAux2);
                // Realmente es -atan2(h2,h1), pero no funciona.
        m_sCoordTheta.theta[2] += 2*PI;
        // Paso el ángulo al intervalo correcto sumándole 2*PI.
        dSeno[2] = sin(m_sCoordTheta.theta[2]);
        dCoseno[2] = cos(m_sCoordTheta.theta[2]);
*/
        //      Lo calculamos a mi manera (Ale)

        m_sCoordTheta.theta[2]=PI/2+2*dAux4;
        dSeno[2] = sin(m_sCoordTheta.theta[2]);
        dCoseno[2] = cos(m_sCoordTheta.theta[2]);

        // Calculamos el seno y el coseno de (theta2+theta3)
        dSeno23 = sin(m_sCoordTheta.theta[1] + m_sCoordTheta.theta[2]);
        dCoseno23 = cos(m_sCoordTheta.theta[1] + m_sCoordTheta.theta[2]);

        // Calculamos theta4

// La matriz 4T0 se multiplicaría por el vector de orientacion
// de la fresa m_dMatriz[i][2] para obtener los nuevos vectores de
// orientacion en el sistema de referencia de la articulacion 4 y 5.
// se multulicaría R*P, siendo wRm=-(mRw)t
// Luego:  (el signo negativo no sé por qué no lo pone)
//                       |c1c23 s1c23 -s23| |m_dMatriz[0][2]|
//                       | -s1    c1          0 |*|m_dMatriz[1][2]|
//                       |c1s23 s1s23  c23| |m_dMatriz[2][2]|

        dAux1 = nMuneca*(dCoseno[0]*m_dMatriz[1][2] - dSeno[0]*m_dMatriz[0][2]);
        dAux2 = nMuneca*(dCoseno[0]*dCoseno23*m_dMatriz[0][2] + dSeno[0]*dCoseno23*m_dMatriz[1][2] -
dSeno23*m_dMatriz[2][2]);

        m_sCoordTheta.theta[3]=atan2(dAux1, dAux2);
        dSeno[3] = sin(m_sCoordTheta.theta[3]);
        dCoseno[3] = cos(m_sCoordTheta.theta[3]);

        // Calculamos theta5
        dAux1 = m_dMatriz[0][2]*(dCoseno[0]*dCoseno23*dCoseno[3] - dSeno[0]*dSeno[3]) +
m_dMatriz[1][2]*(dSeno[0]*dCoseno23*dCoseno[3] + dCoseno[0]*dSeno[3]) -
m_dMatriz[2][2]*dCoseno[3]*dSeno23;
        dAux2 = m_dMatriz[0][2]*dCoseno[0]*dSeno23 + m_dMatriz[1][2]*dSeno[0]*dSeno23 +
m_dMatriz[2][2]*dCoseno23;

        m_sCoordTheta.theta[4]=atan2(dAux1, dAux2);
        dSeno[4] = sin(m_sCoordTheta.theta[4]);
        dCoseno[4] = cos(m_sCoordTheta.theta[4]);
/*
        dAux1=sqrt(dAux1*dAux1+dAux2*dAux2);
        dAux3=nMuneca*(dCoseno[0]*dSeno23*m_dMatriz[0][2]+dSeno[0]*dSeno23*m_dMatriz[1][2]+dCoseno23
*m_dMatriz[2][2]);
        m_sCoordTheta.theta[4]=atan2(dAux1,dAux2);
        dSeno[4] = sin(m_sCoordTheta.theta[4]);
```

```
        dCoseno[4] = cos(m_sCoordTheta.theta[4]);
*/
        // Calculamos theta6
        dAux1 = -dCoseno[0]*dCoseno23*dSeno[3] - dSeno[0]*dCoseno[3];
        dAux2 = -dSeno[0]*dCoseno23*dSeno[3] + dCoseno[0]*dCoseno[3];
        dAux3 = dSeno[3]*dSeno23;
        dAux4 = m_dMatriz[0][0]*dAux1 + m_dMatriz[1][0]*dAux2 + m_dMatriz[2][0]*dAux3;
        dAux5 = m_dMatriz[0][1]*dAux1 + m_dMatriz[1][1]*dAux2 + m_dMatriz[2][1]*dAux3;

        m_sCoordTheta.theta[5]=atan2(dAux4, dAux5);
        dSeno[5] = sin(m_sCoordTheta.theta[5]);
        dCoseno[5] = cos(m_sCoordTheta.theta[5]);

        // Pasamos los ángulos a grados, dentro del rango [0,360).
        int nIndex;
        for (nIndex = 0; nIndex < 5; nIndex++)
        {
                m_sCoordTheta.theta[nIndex] *= (180/PI);

                while (m_sCoordTheta.theta[nIndex] >= 360)
                        m_sCoordTheta.theta[nIndex] -= 360;
        }

//      CString strMsg;
//      strMsg.Format("Coordenadas articulares:
\ntheta[0]=%f\ntheta[1]=%f\ntheta[2]=%f\ntheta[3]=%f\ntheta[4]=%f\ntheta[5]=%f",m_sCoordTheta.theta[0
],m_sCoordTheta.theta[1],m_sCoordTheta.theta[2],m_sCoordTheta.theta[3],m_sCoordTheta.theta[4],m_sCoor
dTheta.theta[5]);
//      AfxMessageBox(strMsg);

        TRACE(_T("Coordenadas articulares: %f %f %f %f %f %f\n"), m_sCoordTheta.theta[0],
m_sCoordTheta.theta[1],
                                m_sCoordTheta.theta[2], m_sCoordTheta.theta[3],
m_sCoordTheta.theta[4], m_sCoordTheta.theta[5]);
}

////////////////////////////////////////////////////////////////////
//                                                                //
// Método que sirve para comprobar que el punto es alcanzable.    //
// Para ello comprueba que el valor de las coordenadas            //
// articulares está dentro del rango alcanzable de cada           //
// articulación.                                                  //
//      theta1 = theta[0] = (-160,160)                            //
//      theta2 = theta[1] = (-200,35)                             //
//              El rango teórico es (-137.5,137.5), pero el origen //
//              es -90 y no 0. En la práctica, he llegado a       //
//              (-207.75,41.96) hasta la base y a (-220.15,43.05) //
//              hasta el suelo.                                   //
//      theta3 = theta[2] = (-52.5,232.5)                         //
//              El rango teórico es (-142.5,142.5), pero el origen //
//              es 90 y no 0                                      //
//      theta4 = theta[3] = (-270,270)                            //
//      theta1 = theta[4] = (-105,120)                            //
//      theta1 = theta[5] = (-270,270)                            //
//                                                                //
```

```
// Devuelve TRUE si el punto es alcanzable y FALSE si no lo es              //
// (si el valor de alguna coordenada articular está fuera de                //
// rango).                                                                  //
//                                                                          //
//////////////////////////////////////////////////////////////////////////
BOOL CPuntoTrayectoria::EsAlcanzable()
{
        BOOL fRes = TRUE;
        // Comprobamos que el valor de todas las variables articulares esté
        // dentro del rango alcanzable:
        if ((m_sCoordTheta.theta[0] < -160) || (m_sCoordTheta.theta[0] > 160))
        {
                fRes = FALSE;
                TRACE(_T("ERROR: La coordenada articular theta1 = %f no está dentro del rango [-
160,160]\n"), m_sCoordTheta.theta[0]);
        }
        else if ((m_sCoordTheta.theta[1] < -200) || (m_sCoordTheta.theta[1] > 35))

        {
                fRes = FALSE;
                TRACE(_T("ERROR: La coordenada articular theta2 = %f no está dentro del rango [-
200,35]\n"), m_sCoordTheta.theta[1]);
        }
        else if ((m_sCoordTheta.theta[2] < -52.5) || (m_sCoordTheta.theta[2] > 232.5))
        {
                fRes = FALSE;
                TRACE(_T("ERROR: La coordenada articular theta3 = %f no está dentro del rango [-
52.5,232.5]\n"), m_sCoordTheta.theta[2]);
        }
        else if ((m_sCoordTheta.theta[3] < -270) || (m_sCoordTheta.theta[3] > 270))
        {
                fRes = FALSE;
                TRACE(_T("ERROR: La coordenada articular theta4 = %f no está dentro del rango [-
270,270]\n"), m_sCoordTheta.theta[3]);
        }
        else if ((m_sCoordTheta.theta[4] < -105) || (m_sCoordTheta.theta[4] > 120))
        {
                fRes = FALSE;
                TRACE(_T("ERROR: La coordenada articular theta5 = %f no está dentro del rango [-
105,120]\n"), m_sCoordTheta.theta[4]);
        }
        else if ((m_sCoordTheta.theta[5] < -270) || (m_sCoordTheta.theta[5] > 270))
        {
                fRes = FALSE;
                TRACE(_T("ERROR: La coordenada articular theta6 = %f no está dentro del rango [-
270,270]\n"), m_sCoordTheta.theta[5]);
        }

        return fRes;
}

//////////////////////////////////////////////////////////////////////////
//                                                                      //
// Método que sirve para fijar las coordenadas XYZeulerZYZ del punto.   //
//                                                                      //
```

```
// Parámetros:                                                              //
//       - Array double[6] donde están almacenadas las coordenadas a        //
//         guardar.                                                         //
//                                                                          //
//       QUIZAS MEJOR const double* &datos                                  //
//                                                                          //
//////////////////////////////////////////////////////////////////////////////
void CPuntoTrayectoria::SetXYZvector(double* dCoordenadas)
{
        m_sCoordXYZvector.X = dCoordenadas[0];
        m_sCoordXYZvector.Y = dCoordenadas[1];
        m_sCoordXYZvector.Z = dCoordenadas[2];
        m_sCoordXYZvector.i = dCoordenadas[3];
        m_sCoordXYZvector.j = dCoordenadas[4];
        m_sCoordXYZvector.k = dCoordenadas[5];

        TRACE(_T("Coordenadas XYZvector: %f %f %f %f %f %f\n"),
                        m_sCoordXYZvector.X, m_sCoordXYZvector.Y, m_sCoordXYZvector.Z,
                        m_sCoordXYZvector.i, m_sCoordXYZvector.j, m_sCoordXYZvector.k);
}

//////////////////////////////////////////////////////////////////////////////
//                                                                          //
// Método que sirve para fijar las coordenadas XYZeulerZYZ del punto.       //
//                                                                          //
// Parámetros:                                                              //
//       - Array double[6] donde están almacenadas las coordenadas a        //
//         guardar.                                                         //
//                                                                          //
//////////////////////////////////////////////////////////////////////////////
void CPuntoTrayectoria::SetXYZeulerZYZ(double* dCoordenadas)
{
        m_sCoordXYZeulerZYZ.X = dCoordenadas[0];
        m_sCoordXYZeulerZYZ.Y = dCoordenadas[1];
        m_sCoordXYZeulerZYZ.Z = dCoordenadas[2];
        m_sCoordXYZeulerZYZ.eulerZalfa = dCoordenadas[3];
        m_sCoordXYZeulerZYZ.eulerYbeta = dCoordenadas[4];
        m_sCoordXYZeulerZYZ.eulerZgamma = dCoordenadas[5];
}

//////////////////////////////////////////////////////////////////////////////
//                                                                          //
// Método que sirve para obtener las coordenadas XYZeulerZYZ del punto.     //
//                                                                          //
// Parámetros:                                                              //
//       - Array double[6] donde se van a almacenar las coordenadas         //
//         requeridas.                                                      //
//                                                                          //
//       QUIZAS MEJOR double* &datos                                        //
//                                                                          //
//////////////////////////////////////////////////////////////////////////////
void CPuntoTrayectoria::GetXYZeulerZYZ(double* dCoordenadas)
{
        dCoordenadas[0] = m_sCoordXYZeulerZYZ.X;
        dCoordenadas[1] = m_sCoordXYZeulerZYZ.Y;
```

```
        dCoordenadas[2] = m_sCoordXYZeulerZYZ.Z;
        dCoordenadas[3] = m_sCoordXYZeulerZYZ.eulerZalfa;
        dCoordenadas[4] = m_sCoordXYZeulerZYZ.eulerYbeta;
        dCoordenadas[5] = m_sCoordXYZeulerZYZ.eulerZgamma;
}

////////////////////////////////////////////////////////////////////////
//                                                                    //
// Método que sirve para crear la matriz de transformación homogénea a //
// partir de 3 puntos en coordenadas cartesianas (p0,p1,p2).          //
// p0 = Esquina superior iqda.                                        //
// p1 = Esquina superior derecha.                                     //
// p2 = Esquina inferior derecha.                                     //
//                                                                    //
// Parámetros:                                                        //
//       - Array double[3][3] donde están almacenados las coordenadas //
//         cartesianas     de los 3 puntos.                           //
//                                                                    //
//       QUIZAS MEJOR double* &datos                                  //
//                                                                    //
////////////////////////////////////////////////////////////////////////
void CPuntoTrayectoria::CrearMatriz(Vector* vPuntos)
{
//      Vector Xp, Yp, Zp;
        double dModulo;
        Vector vAux, vU, vV;

//      Definir clases CMatriz y CVector.

        // La matriz de transformación homogénea será de la forma:
        // [ u   v   w : p0 ]
        // [.............]
        // [ 0   0   0 :  1 ]

        // Rellenamos la 1ª columna con el vector unitario diferencia entre p0 y p1(u).
        vAux.x = vPuntos[2].x - vPuntos[0].x;
        vAux.y = vPuntos[2].y - vPuntos[0].y;
        vAux.z = vPuntos[2].z - vPuntos[0].z;

        dModulo = sqrt(vAux.x*vAux.x + vAux.y*vAux.y + vAux.z*vAux.z);

        m_dMatriz[0][0] = vU.x = vAux.x/dModulo;
        m_dMatriz[1][0] = vU.y = vAux.y/dModulo;
        m_dMatriz[2][0] = vU.z = vAux.z/dModulo;

        // Rellenamos la 2ª columna con el vector unitario diferencia entre p0 y p2 (v).
        vAux.x = vPuntos[1].x - vPuntos[0].x;
        vAux.y = vPuntos[1].y - vPuntos[0].y;
        vAux.z = vPuntos[1].z - vPuntos[0].z;

        dModulo = sqrt(vAux.x*vAux.x + vAux.y*vAux.y + vAux.z*vAux.z);

        m_dMatriz[0][1] = vV.x = vAux.x/dModulo;
        m_dMatriz[1][1] = vV.y = vAux.y/dModulo;
        m_dMatriz[2][1] = vV.z = vAux.z/dModulo;
```

```
        // Rellenamos la 3ª columna con el vector unitario que completa la base ortonormal (w).
        vAux.x = vU.y*vV.z - vU.z*vV.y;
        vAux.y = vU.z*vV.x - vU.x*vV.z;
        vAux.z = vU.x*vV.y - vU.y*vV.x;

        dModulo = sqrt(vAux.x*vAux.x + vAux.y*vAux.y + vAux.z*vAux.z);

        m_dMatriz[0][2] = vAux.x/dModulo;
        m_dMatriz[1][2] = vAux.y/dModulo;
        m_dMatriz[2][2] = vAux.z/dModulo;

        // Rellenamos la 4ª columna con el punto p0.
        m_dMatriz[0][3] = vPuntos[0].x;
        m_dMatriz[1][3] = vPuntos[0].y;
        m_dMatriz[2][3] = vPuntos[0].z;
/*

        Xp.x=vPuntos[0].x-vPuntos[1].x;
        Xp.y=vPuntos[0].y-vPuntos[1].y;
        Xp.z=vPuntos[0].z-vPuntos[1].z;

        dModulo=sqrt((Xp.x*Xp.x)+(Xp.y*Xp.y)+(Xp.z*Xp.z));
        Xp.x=Xp.x/dModulo;
        Xp.y=Xp.y/dModulo;
        Xp.z=Xp.z/dModulo;

        Yp.x=vPuntos[1].x-vPuntos[2].x;
        Yp.y=vPuntos[1].y-vPuntos[2].y;
        Yp.z=vPuntos[1].z-vPuntos[2].z;

        dModulo=sqrt((Yp.x*Yp.x)+(Yp.y*Yp.y)+(Yp.z*Yp.z));
        Yp.x=Yp.x/dModulo;
        Yp.y=Yp.y/dModulo;
        Yp.z=Yp.z/dModulo;

// La componente Z la obtenemos mediante elproducto vectorial de Xp y Yp

        Zp.x=Xp.y*Yp.z-Xp.z*Yp.y;
        Zp.y=Xp.z*Yp.x-Xp.x*Yp.z;
        Zp.z=Xp.x*Yp.y-Xp.y*Yp.x;

// La matriz queda de la siguiente forma:

//              [   R     : P(x,y,z]                        [ Xp.x Yp.x Zp.x ]
//              [.............]                 con R=   [ Xp.y Yp.y Zp.y ]
//              [0  0  0:   1       ]                      [ Xp.z Yp.z Zp.z ]

        m_dMatriz[0][0]=Xp.x;
        m_dMatriz[1][0]=Xp.y;
        m_dMatriz[2][0]=Xp.z;
        m_dMatriz[0][1]=Yp.x;
        m_dMatriz[1][1]=Yp.y;
        m_dMatriz[2][1]=Yp.z;
        m_dMatriz[0][2]=Zp.x;
```

```
        m_dMatriz[1][2]=Zp.y;
        m_dMatriz[2][2]=Zp.z;

        m_dMatriz[0][3]=vPuntos[1].x;
        m_dMatriz[1][3]=vPuntos[1].y;
        m_dMatriz[2][3]=vPuntos[1].z;
*/
//      CString strMsg;
//      strMsg.Format("Matriz: \n%f %f %f %f\n%f %f %f %f\n%f %f %f %f\n%f %f %f
%f\n",m_dMatriz[0][0],m_dMatriz[0][1],m_dMatriz[0][2],m_dMatriz[0][3],m_dMatriz[1][0],m_dMatriz[1][1]
,m_dMatriz[1][2],m_dMatriz[1][3],m_dMatriz[2][0],m_dMatriz[2][1],m_dMatriz[2][2],m_dMatriz[2][3],m_dM
atriz[3][0],m_dMatriz[3][1],m_dMatriz[3][2],m_dMatriz[3][3]);
//      AfxMessageBox(strMsg);

        TRACE(_T("Matriz de transformación homogénea:\n%f %f %f %f\n%f %f %f %f\n%f %f %f %f\n%f %f
%f %f\n"),
                        m_dMatriz[0][0],m_dMatriz[0][1],m_dMatriz[0][2],m_dMatriz[0][3],
                        m_dMatriz[1][0],m_dMatriz[1][1],m_dMatriz[1][2],m_dMatriz[1][3],
                        m_dMatriz[2][0],m_dMatriz[2][1],m_dMatriz[2][2],m_dMatriz[2][3],
                        m_dMatriz[3][0],m_dMatriz[3][1],m_dMatriz[3][2],m_dMatriz[3][3]);
}


////////////////////////////////////////////////////////////////////////
//                                                                    //
// Método que sirve para invertir la matriz de transformación homogénea //
// (4x4) por el método de Gauss-Jordan.                               //
//                                                                    //
// Realizada por Óscar Collazo, adaptada por Manuel Vargas. Luego por //
// Manuel Gómez.                                                      //
// Escribe la matriz inversa en m_dMatriz.                            //
//                                                                    //
////////////////////////////////////////////////////////////////////////
void CPuntoTrayectoria::InvertirMatriz()
{
        double dMatrizInv[4][4];
        double dMatrizAux[4][4];
        int nIndex, nIndex2;
        int i, j, f;
        int nEncontrePivote;
        double dPivote1, dPivote2, dAux;

        // Inicialmente, la matriz inversa será la identidad:
        for (nIndex = 0; nIndex < 4; nIndex++)
        {
                for (nIndex2 = 0; nIndex2 < 4; nIndex2++)
                {
                        if (nIndex == nIndex2)
                                dMatrizInv[nIndex][nIndex2] = 1;
                        else
                                dMatrizInv[nIndex][nIndex2] = 0;
                }
        }

        // La matriz dMatrizAux es inicialmente igual a la original.
```

```
        for (nIndex = 0; nIndex < 4; nIndex++)
                for (nIndex2 = 0; nIndex2 < 4; nIndex2++)
                        dMatrizAux[nIndex][nIndex2] = m_dMatriz[nIndex][nIndex2];

        // Diagonalización inferior.
        for (f=0; f<4; f++)
        {
      // Tomamos el elemento de la diagonal correspondiente a la fila en la que estamos:
                dPivote1 = dMatrizAux[f][f];

      // Si el pivote es prácticamente cero, debemos intercambiar la fila actual por una
      // de las inferiores que no tenga ese problema.

/*        if (_Despreciable(dPivote1))
        // Un nro. real lo consideraremos despreciablemente pequeño si se cumple esa
        // condición. Realmente, ésta es una cota para los float pero para los double
        // podría ser bastante menor(1e-307 )
        // Una división en la que aparecezca un nro. de este orden en el denominador
        // es arriesgada (aunque todo depende del numerador, por supuesto).
        #define _Despreciable(x) (fabs(x) < 1e-37)
*/
                if (fabs(dPivote1) < 1e-50 )
                {
                        nEncontrePivote = 0;
                for (i=f+1; i<4; i++)
                        {
                        dPivote1 = dMatrizAux[i][f];

                                if(! (fabs(dPivote1) < 1e-50))
                                {
                                        // Hemos encontrado un elemento no nulo, intercambiamos
filas:
                                        nEncontrePivote = 1;
                                        for (j=0; j<4; j++)
                                        {
                                                // Intercambio los elementos de ambas filas:
                                                dAux = dMatrizInv[i][j];
                                                dMatrizInv[i][j] = dMatrizInv[f][j];
                                                dMatrizInv[f][j] = dAux;
                                                if (j >= f)
                                                {
                                                dAux = dMatrizAux[i][j];
                                                        dMatrizAux[i][j] = dMatrizAux[f][j];
                                                        dMatrizAux[f][j] = dAux;
                                                }
                                        }

                                // Salimos del bucle, para conservar en dPivote1 el valor.
                   break;
                }
         }
        // Si todos los elementos inferiores de esa columna son cero => la matriz no es invertible:
        if (!nEncontrePivote)
           return;//return(1);
      }
```

```
                // En primer lugar dividimos la fila actual por su pivote:
                for (i=0; i<4; i++)
                {
                        dMatrizInv[f][i] /= dPivote1;
                        if (i >= f)  // En el caso de la matriz que se está llevando a la
identidad, todos los elementos previos al de la diagonal ya deben de ser cero.
                        dMatrizAux[f][i] /= dPivote1;
                }

                // Ahora se diagonaliza la parte inferior a la diagonal principal
                for (i=f+1; i<4; i++)
                {
                        dPivote2 = dMatrizAux[i][f];
                        for (j=0; j<4; j++)
                        {
                                dMatrizInv[i][j] -= dMatrizInv[f][j] * dPivote2;
                                if (j >= f)
                                        dMatrizAux[i][j] -= dMatrizAux[f][j] * dPivote2;
                        }
                }
        }

        // Ahora comienza la diagonalización superior:
   for (f=3; f>=0; f--)
        {
                for (i=f-1; i>=0; i--)
                {
                dPivote2 = dMatrizAux[i][f];
                        for (j=3; j>=0; j--)
                        {
                                dMatrizInv[i][j] -= dMatrizInv[f][j] * dPivote2;
                                if (j <= f)
                                        dMatrizAux[i][j] -= dMatrizAux[f][j] * dPivote2;
                        }
                }
        }

   // Si hemos llegado hasta aquí, no ha habido error. Pasamos los valores de
   // dMatrizInv a la matriz mienbro.
        for (nIndex = 0; nIndex < 4; nIndex++)
                for (nIndex2 = 0; nIndex2 < 4; nIndex2++)
                        m_dMatriz[nIndex][nIndex2] = dMatrizInv[nIndex][nIndex2];


        TRACE(_T("Matriz de transformación homogénea invertida:\n%f %f %f %f\n%f %f %f %f\n%f %f %f
%f\n%f %f %f %f\n"),
                        m_dMatriz[0][0],m_dMatriz[0][1],m_dMatriz[0][2],m_dMatriz[0][3],
                        m_dMatriz[1][0],m_dMatriz[1][1],m_dMatriz[1][2],m_dMatriz[1][3],
                        m_dMatriz[2][0],m_dMatriz[2][1],m_dMatriz[2][2],m_dMatriz[2][3],
                        m_dMatriz[3][0],m_dMatriz[3][1],m_dMatriz[3][2],m_dMatriz[3][3]);
}

//////////////////////////////////////////////////////////////////////
//                                                                  //
```

```
//      Método que compone los puntos en formato XYZvector respecto        //
//      del sistema de referencia world.                                    //
//      Función creada por Alejandro Guija y usada para depurar             //
//      resultados.                                                         //
//                                                                          //
//////////////////////////////////////////////////////////////////////////////

void CPuntoTrayectoria::SetXYZworld(const CPuntoTrayectoria& Inicio)
{
        Vector vAux;
        Vector vAux2;
        double aux;
//      En Inicio.m_dMatriz tenemos la matriz de transformacion wTp
//      que nos permite pasar un punto en el sistema de referencia
//      de la pieza al sistema de referencia real del robot.

//  Pasamos el punto p de la trayectoria respecto al sistema world.

        vAux.x=Inicio.m_dMatriz[0][0]*m_sCoordXYZvector.X+
                                Inicio.m_dMatriz[0][1]*m_sCoordXYZvector.Y+
                                Inicio.m_dMatriz[0][2]*m_sCoordXYZvector.Z+
                                Inicio.m_dMatriz[0][3];

        vAux.y=Inicio.m_dMatriz[1][0]*m_sCoordXYZvector.X+
                                Inicio.m_dMatriz[1][1]*m_sCoordXYZvector.Y+
                                Inicio.m_dMatriz[1][2]*m_sCoordXYZvector.Z+
                                Inicio.m_dMatriz[1][3];

        vAux.z=Inicio.m_dMatriz[2][0]*m_sCoordXYZvector.X+
                                Inicio.m_dMatriz[2][1]*m_sCoordXYZvector.Y+
                                Inicio.m_dMatriz[2][2]*m_sCoordXYZvector.Z+
                                Inicio.m_dMatriz[2][3];

        m_sCoordXYZvectorworld.X=vAux.x;
        m_sCoordXYZvectorworld.Y=vAux.y;
        m_sCoordXYZvectorworld.Z=vAux.z;

//  Calculamos el punto cuya diferencia con el anterior forma el
//  vector (i,j,k) en el sistema de referencia del diseño;

        vAux.x=m_sCoordXYZvector.X+m_sCoordXYZvector.i;
        vAux.y=m_sCoordXYZvector.Y+m_sCoordXYZvector.j;
        vAux.z=m_sCoordXYZvector.Z+m_sCoordXYZvector.k;

//  Calculamos las coordenadas de este punto en el sistema world.

        vAux2.x=Inicio.m_dMatriz[0][0]*vAux.x+
                                Inicio.m_dMatriz[0][1]*vAux.y+
                                Inicio.m_dMatriz[0][2]*vAux.z+
                                Inicio.m_dMatriz[0][3];

        vAux2.y=Inicio.m_dMatriz[1][0]*vAux.x+
                                Inicio.m_dMatriz[1][1]*vAux.y+
                                Inicio.m_dMatriz[1][2]*vAux.z+
                                Inicio.m_dMatriz[1][3];
```

```
        vAux2.z=Inicio.m_dMatriz[2][0]*vAux.x+
                                Inicio.m_dMatriz[2][1]*vAux.y+
                                Inicio.m_dMatriz[2][2]*vAux.z+
                                Inicio.m_dMatriz[2][3];

        m_sCoordXYZvectorworld.i=vAux2.x-m_sCoordXYZvectorworld.X;
        m_sCoordXYZvectorworld.j=vAux2.y-m_sCoordXYZvectorworld.Y;
        m_sCoordXYZvectorworld.k=vAux2.z-m_sCoordXYZvectorworld.Z;

// El vector debe ser unitario, lo comprobamos:

        aux=sqrt(m_sCoordXYZvectorworld.i*m_sCoordXYZvectorworld.i+
                        m_sCoordXYZvectorworld.j*m_sCoordXYZvectorworld.j+
                        m_sCoordXYZvectorworld.k*m_sCoordXYZvectorworld.k);

//      CString msg;
//      msg.Format("Coordenadas cartesianas: \nX=%f\nY=%f\nZ=%f\n\nCoordenadas de orientación:
\ni=%f\nj=%f\nk=%f",
//
        m_sCoordXYZvectorworld.X,m_sCoordXYZvectorworld.Y,m_sCoordXYZvectorworld.Z,
//
        m_sCoordXYZvectorworld.i,m_sCoordXYZvectorworld.j,m_sCoordXYZvectorworld.k);
//      AfxMessageBox(msg);
}
```

B.2.11. Otros archivos de cabecera.

B.2.11.1. Estructura.h

```
////////////////////////////////////////////////////////////
//                                                        //
//      Estructura.h: archivo de cabecera que define las  //
//      estructuras utilizadas para manejar el robot.     //
//                                                        //
//      Creado por F.Javier Rueda 2001                    //
////////////////////////////////////////////////////////////

//Estructura de posicion del robot

#ifndef _ESTRUCTURA_
#define _ESTRUCTURA_

struct Rx90Pos
{
        double Artic[6];
        double Cart[6];
};

#endif
```

B.2.11.2. Resource.h

```
////////////////////////////////////////////////////////////
//                                                        //
```

```
//  resource.h: Fichero de cabecera de identificadores.    //
//                                                         //
//      Creado por Microsoft Developer Studio.             //
//                                                         //
/////////////////////////////////////////////////////////////

//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Rx90.rc
//
#define IDD_ABOUTBOX                    100
#define IDR_MAINFRAME                   128
#define IDR_RX90TYPE                    129
#define IDI_ICON1                       130
#define IDD_DIALOGCAMARA                131
#define IDD_DIALOGARTIC                 132
#define IDD_DIALOGANGULO                135
#define IDD_DIALOGPOS                   136
#define IDC_SLIDER1                     1000
#define IDC_SLIDER2                     1001
#define IDC_STATIC_YAW                  1002
#define IDC_SLIDER3                     1002
#define IDC_STATIC_PITCH                1003
#define IDC_SLIDER4                     1003
#define IDC_SLIDER5                     1004
#define IDC_SLIDER6                     1005
#define IDC_ANGULO1                     1017
#define IDC_BARRA1                      1018
#define IDC_BARRA2                      1019
#define IDC_ANGULO2                     1020
#define IDC_BARRA3                      1021
#define IDC_ANGULO3                     1022
#define IDC_BARRA4                      1023
#define IDC_ANGULO4                     1024
#define IDC_BARRA5                      1025
#define IDC_ANGULO5                     1026
#define IDC_BARRA6                      1027
#define IDC_ANGULO6                     1028
#define IDC_POSx                        1029
#define IDC_POSy                        1030
#define IDC_POSz                        1031
#define IDC_YAW                         1032
#define IDC_SPIN1                       1032
#define IDC_PITCH                       1033
#define IDC_COORDX                      1033
#define IDC_ROLL                        1034
#define IDC_BARRA7                      1038
#define IDC_XPOS                        1039
#define IDC_BARRA8                      1040
#define IDC_YPOS                        1041
#define IDC_BARRA9                      1042
#define IDC_ZPOS                        1043
#define IDC_THETA1                      1044
#define IDC_THETA2                      1045
#define IDC_THETA3                      1046
```

```
#define IDC_THETA4                      1047
#define IDC_THETA5                      1048
#define IDC_THETA6                      1049
#define ID_CAPTURA_GOON                 32771
#define ID_STOP                         32772
#define ID_VER_PUNTODEVISTA             32773
#define ID_MOVIMIENTO_START             32774
#define ID_MOVIMIENTO_STOP              32775
#define ID_MOVIMIENTO_RESET             32776
#define ID_MOVIMIENTO_MANUAL_ARTICULAR  32778
#define ID_MOVIMIENTO_MANUAL_CARTESIANO 32779
#define ID_MOVIMIENTO_MANUAL_CARTESIANA 32782

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS                     1
#define _APS_NEXT_RESOURCE_VALUE        137
#define _APS_NEXT_COMMAND_VALUE         32783
#define _APS_NEXT_CONTROL_VALUE         1050
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```