8 PROGRAMACIÓN

8.1 Visual C++

8.1.1 DirectInput

8.1.1.1 Funciones y propiedades

```
//-----------------------------------------------------------------------------
// Name: InitDirectInput()
// Desc: Initialize the DirectInput variables.
//-----------------------------------------------------------------------------
HRESULT CPrueba1Dlg::InitDirectInput( HWND hDlg )
{
    HRESULT hr;

    // Register with the DirectInput subsystem and get a pointer
    // to a IDirectInput interface we can use.
    // Create a DInput object

          if( FAILED( hr = DirectInput8Create( GetModuleHandle(NULL), DIRECTINPUT_VERSION,
                                IID_IDirectInput8, (VOID**)&_pDI, NULL ) ) )
        return hr;

    // Look for a simple Joystick we can use for this sample program.
    if( FAILED( hr = _pDI->EnumDevices( DI8DEVCLASS_GAMECTRL,
                                EnumJoysticksCallback,
                                NULL, DIEDFL_ATTACHEDONLY ) ) )
        return hr;


    // Make sure we got a Joystick
    if( NULL == _pJoystick )
    {
                    if ((AfxMessageBox(_T("El Joystick no funciona o no está conectado. ¿Desea
cerrar la aplicación y conectar el Joystick?.\n\nIMPORTANTE:\nSi continua, las funciones del
Joystick no funcionarán aunque aparezcan activadas"),
                MB_ICONERROR | MB_OKCANCEL))==TRUE)
                    {
                            EndDialog( 0 );
                    }
                    return S_OK;
    }


    // Set the data format to "simple Joystick" - a predefined data format
    //
    // A data format specifies which controls on a device we are interested in,
```

```
    // and how they should be reported. This tells DInput that we will be
    // passing a DIJOYSTATE2 structure to IDirectInputDevice::GetDeviceState().
    if( FAILED( hr = _pJoystick->SetDataFormat( &c_dfDIJoystick2 ) ) )
        return hr;

    // Set the cooperative level to let DInput know how this device should
    // interact with the system and with other DInput applications.
    if( FAILED( hr = _pJoystick->SetCooperativeLevel( hDlg, DISCL_EXCLUSIVE |
                                            DISCL_FOREGROUND ) ) )
        return hr;

    // Enumerate the Joystick objects. The callback function enabled user
    // interface elements for objects that are found, and sets the min/max
    // values property for discovered axes.
    if( FAILED( hr = _pJoystick->EnumObjects( EnumObjectsCallback,
                                    (VOID*)hDlg, DIDFT_ALL ) ) )
        return hr;


    return S_OK;
}



//-----------------------------------------------------------------------------
// Name: UpdateInputState()
// Desc: Get the input device's state and display it.
//-----------------------------------------------------------------------------
HRESULT CPrueba1Dlg::UpdateInputState( HWND hDlg )
{
    HRESULT     hr;
    TCHAR       strText[512]; // Device state text
    TCHAR                   strTextMotor[512];
            DIJOYSTATE2 js;           // DInput Joystick state
    TCHAR*      str;

    if( NULL == _pJoystick )
        return S_OK;

    // Poll the device to read the current state
    hr = _pJoystick->Poll();
    if( FAILED(hr) )
    {
        // DInput is telling us that the input stream has been
        // interrupted. We aren't tracking any state between polls, so
        // we don't have any special reset that needs to be done. We
        // just re-acquire and try again.
        hr = _pJoystick->Acquire();
        while( hr == DIERR_INPUTLOST )
            hr = _pJoystick->Acquire();

        // hr may be DIERR_OTHERAPPHASPRIO or other errors.  This
        // may occur when the app is minimized or in the process of
        // switching, so just try again later
        return S_OK;
    }
```

```
    // Get the input's device state
    if( FAILED( hr = _pJoystick->GetDeviceState( sizeof(DIJOYSTATE2), &js ) ) )
        return hr; // The device should have been acquired during the Poll()

    // Display Joystick state to dialog
            if(activar==1)
            {

    // Axes Servo
    wsprintf( strText, TEXT("%ld"), js.lY );
    m_TxtAxisX.SetWindowText(strText);

            // Axes Motor
            wsprintf( strTextMotor, TEXT("%ld"), js.lZ );
            m_TxtAxisZ.SetWindowText(strTextMotor);


            PosServo = atoi (strText);
            PosMotor = atoi (strTextMotor);

            m_SliderServo.SetPos(PosServo+((RangeMaxSliderServo)/(2)));
            m_SliderMotor.SetPos(PosMotor+((RangeMaxSliderMotor)/(2)));

            CString GuardaServo;
            int Servo=0;
            CString GuardaMotor;
            int Motor=0;

            m_TxtAxisX.GetWindowText(GuardaServo);
            Servo=atoi(GuardaServo);
            m_TxtAxisZ.GetWindowText(GuardaMotor);
            Motor=atoi(GuardaMotor);


    // Fill up text with which buttons are pressed
    str = strText;
    for( int i = 0; i < 128; i++ )
    {
        if ( js.rgbButtons[i] & 0x80 )
            str += wsprintf( str, TEXT("%02d "), i );
    }
    *str = 0;   // Terminate the string

    m_TxtBoton.SetWindowText(strText );
            }

            UpdateData(FALSE);

    return S_OK;
}
```

```
//-----------------------------------------------------------------------------
// Name: FreeDirectInput()
// Desc: Initialize the DirectInput variables.
//-----------------------------------------------------------------------------
VOID CPrueba1Dlg::FreeDirectInput()
{
    // Unacquire the device one last time just in case
    // the app tried to exit while the device is still acquired.
    if( _pJoystick )
        _pJoystick->Unacquire();

    // Release any DirectInput objects.
    SAFE_RELEASE( _pJoystick );
    SAFE_RELEASE( _pDI );
}


LRESULT CPrueba1Dlg::DefWindowProc(UINT message, WPARAM wParam, LPARAM lParam)
{
          // TODO: Add your specialized code here and/or call the base class

          switch( message )
    {
        case WM_ACTIVATE:
            if( WA_INACTIVE != wParam && _pJoystick )
            {
                // Make sure the device is acquired, if we are gaining focus.
                _pJoystick->Acquire();
            }
            return TRUE;

        case WM_TIMER:
            // Update the input device every timer message
            if( FAILED( UpdateInputState((HWND)this->m_hWnd) ) )
            {
                KillTimer(0);
                AfxMessageBox( _T("Error Reading Input State.The sample will now exit.DirectInput
Sample"),
                        MB_ICONERROR | MB_OK );
                EndDialog(TRUE );
            }
        return TRUE;

        /*//Con esta case no funciona ni el gamepad ni la captura de datos
                    case WM_COMMAND:
            switch( LOWORD(wParam) )
            {
                case IDCANCEL:
                    EndDialog( 0 );
                    return TRUE;
            }*/

        case WM_DESTROY:
            // Cleanup everything
            KillTimer( 0 );
            FreeDirectInput();
```

```
        return TRUE;
    }

        return CDialog::DefWindowProc(message, wParam, lParam);
}
```

8.1.1.2 CallBacks

```
//////////////////////////////////////////////////////////////////////////
// The one and only CPrueba1App object

CPrueba1App theApp;
LPDIRECTINPUT8      _pDI=NULL;
LPDIRECTINPUTDEVICE8 _pJoystick=NULL;

//////////////////////////////////////////////////////////////////////////
// CPrueba1App initialization

BOOL CPrueba1App::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

        CPrueba1Dlg 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;
}
```

```
//-----------------------------------------------------------------------------
// Name: EnumJoysticksCallback()
// Desc: Called once for each enumerated Joystick. If we find one, create a
//       device interface on it so we can play with it.
//-----------------------------------------------------------------------------
BOOL CALLBACK EnumJoysticksCallback( const DIDEVICEINSTANCE* pdidInstance,
                                     VOID* pContext )
{
    HRESULT hr;

    // Obtain an interface to the enumerated Joystick.
    hr = _pDI->CreateDevice( pdidInstance->guidInstance, &_pJoystick, NULL );

    // If it failed, then we can't use this Joystick. (Maybe the user unplugged
    // it while we were in the middle of enumerating it.)
    if( FAILED(hr) )
        return DIENUM_CONTINUE;

    // Stop enumeration. Note: we're just taking the first Joystick we get. You
    // could store all the enumerated Joysticks and let the user pick.
    return DIENUM_STOP;
}

//-----------------------------------------------------------------------------
// Name: EnumObjectsCallback()
// Desc: Callback function for enumerating objects (axes, buttons, POVs) on a
//       Joystick. This function enables user interface elements for objects
//       that are found to exist, and scales axes min/max values.
//-----------------------------------------------------------------------------
BOOL CALLBACK EnumObjectsCallback( const DIDEVICEOBJECTINSTANCE* pdidoi,
                                   VOID* pContext )
{
    HWND hDlg = (HWND)pContext;

            int RangeMaxSliderServo=510;
            int RangeMaxSliderMotor=510;

    static int nSliderCount = 0;  // Number of returned slider controls
    static int nPOVCount = 0;     // Number of returned POV controls

    // For axes that are returned, set the DIPROP_RANGE property for the
    // enumerated axis in order to scale min/max values.
    if( pdidoi->dwType & DIDFT_AXIS )
    {
        DIPROPRANGE diprg;
        diprg.diph.dwSize       = sizeof(DIPROPRANGE);
        diprg.diph.dwHeaderSize = sizeof(DIPROPHEADER);
        diprg.diph.dwHow        = DIPH_BYID;
        diprg.diph.dwObj        = pdidoi->dwType; // Specify the enumerated axis
        diprg.lMin              = -((RangeMaxSliderServo)/(2));
        diprg.lMax              = +((RangeMaxSliderServo)/(2));

        // Set the range for the axis
        if( FAILED( _pJoystick->SetProperty( DIPROP_RANGE, &diprg.diph ) ) )
            return DIENUM_STOP;
```

```
    }


    // Set the UI to reflect what objects the Joystick supports
    if (pdidoi->guidType == GUID_XAxis)
    {
        EnableWindow( GetDlgItem( hDlg, IDC_X_AXIS ), TRUE );
        EnableWindow( GetDlgItem( hDlg, IDC_X_AXIS_TEXT ), TRUE );
    }

    if (pdidoi->guidType == GUID_RzAxis)
    {
        EnableWindow( GetDlgItem( hDlg, IDC_Z_ROT ), TRUE );
        EnableWindow( GetDlgItem( hDlg, IDC_Z_ROT_TEXT ), TRUE );
    }

    return DIENUM_CONTINUE;
}
```

8.1.2 PortCOM

```
PortCOM::PortCOM()
{
        idCom=NULL;
}

PortCOM::~PortCOM()
{

}

int PortCOM::Init(CString port)
{
        idCom=CreateFile(port,GENERIC_READ|GENERIC_WRITE,0,NULL,OPEN_EXISTING,FIL
E_ATTRIBUTE_NORMAL,NULL);
        if(idCom==INVALID_HANDLE_VALUE )
                {
                        idCom=NULL;
                        return(PORT_ERROR_CREATEFILE);
                }
        else
                {

                        GetCommState(idCom, &dcb);
                        dcb.BaudRate = CBR_115200;
                        dcb.ByteSize = 8;
                        dcb.Parity =      NOPARITY;
                        dcb.StopBits = ONESTOPBIT;

                        dcb.fBinary=1;
                        dcb.fOutxCtsFlow=0;
                        dcb.fOutxDsrFlow=0;
                        dcb.fDtrControl=DTR_CONTROL_DISABLE;
                        dcb.fDsrSensitivity=0;
                        dcb.fTXContinueOnXoff=1;
```

```
                        dcb.fOutX=0;
                        dcb.fInX=0;
                        dcb.fErrorChar=0;
                        dcb.fNull=0;
                        dcb.fRtsControl=RTS_CONTROL_DISABLE;
                        dcb.fAbortOnError=1;
                        dcb.XonLim=2048;
                        dcb.XoffLim=512;
                        dcb.XonChar=17;
                        dcb.XoffChar=19;
                        dcb.ErrorChar=0;
                        dcb.EofChar=0;
                        dcb.EvtChar=0;

                        dcb.DCBlength=sizeof(dcb);
                        int ret=SetCommState(idCom,&dcb);
                        if(ret==0)
                                {
                                        int err=GetLastError();
                                        CloseHandle(idCom);
                                        idCom=NULL;
                                        return(PORT_ERROR_SETDCB);
                                }

        timeouts.ReadIntervalTimeout=1/*PORT_COMMS_INTERVAL_TIMEOUT*/;

        timeouts.ReadTotalTimeoutMultiplier=1/*PORT_COMMS_TOTAL_TIMEOUT_MULTIPLIER
*/;

        timeouts.ReadTotalTimeoutConstant=PORT_COMMS_TOTAL_TIMEOUT_CONSTANT;
                        timeouts.WriteTotalTimeoutMultiplier=0;
                        timeouts.WriteTotalTimeoutConstant=0;
                        SetCommTimeouts(idCom,&timeouts);
                        return(PORT_OK);
                }
}

int PortCOM::Close()
{
        if(CloseHandle(idCom)!=0)return(PORT_OK);
        else return(PORT_ERROR_CLOSE_HANDLE);
        idCom=NULL;
}

int PortCOM::Send(unsigned long num,/*unsigned char **/PVOID pmessage)
{
        unsigned long written=0;
        if(WriteFile(idCom,pmessage,num,&written,NULL)==0)
        return(PORT_ERROR_WRITEFILE);
        if(written!=num) return(PORT_ERROR_WRITENUM);
        //flush(idCom);
        return(PORT_OK);
}

int PortCOM::Receive(unsigned long num, /*unsigned char **/PVOID pmessage)
{
```

```
	unsigned long read=0;
	unsigned long EnviadoOk=0;

	EnviadoOk=ReadFile(idCom,pmessage,num,&read,NULL);
	if(EnviadoOk==0) return PORT_ERROR_READFILE;
	//ReadFile -> If the function fails, the return value is zero
	return(EnviadoOk);
}

bool PortCOM::IsPortValid()
{
	if(idCom!=NULL)return(true);
	else return(false);
}

int PortCOM::SetBaud(int baud)
{
	if(baud<1||baud>3)return PORT_ERROR_BAD_PARAMETER;
	if(baud==PORT_COMMS_BAUDS_115200)dcb.BaudRate=CBR_115200;
	if(baud==PORT_COMMS_BAUDS_38400)dcb.BaudRate=CBR_38400;
	if(SetCommState(idCom,&dcb)==0)
		return(PORT_ERROR_SETDCB);
	return(PORT_OK);
}

void PortCOM::SetupCommPort(CString port)
{
  DCB dcb;
  hCom = CreateFile( port, GENERIC_READ | GENERIC_WRITE, 0, 0, OPEN_EXISTING, 0, 0);
  GetCommState(hCom, &dcb);
  dcb.BaudRate = CBR_115200;
  dcb.fParity = FALSE;
  dcb.fOutxCtsFlow = FALSE;
  dcb.fOutxDsrFlow = FALSE;
  dcb.fDtrControl = DTR_CONTROL_DISABLE;
  dcb.fDsrSensitivity = FALSE;
  dcb.fOutX = FALSE;
  dcb.fInX = FALSE;
  dcb.fRtsControl = RTS_CONTROL_DISABLE;
  dcb.fAbortOnError = FALSE;
  dcb.ByteSize = 8;
  dcb.Parity = NOPARITY;
  dcb.StopBits = ONESTOPBIT;
  SetCommState(hCom, &dcb);
}

void PortCOM::SerOut(BYTE sbuf)
{
  DWORD n;
  WriteFile(hCom, &sbuf, 1, &n, NULL);
}

BYTE PortCOM::SerIn(void)
{
  BYTE sbuf;
```

```
  DWORD n;
  ReadFile(hCom, &sbuf, 1, &n, NULL);
  return sbuf;
}
```

8.1.3 Eventos

```
void CPrueba1Dlg::OnButtonInicio()
{
	// TODO: Add your control notification handler code here

		CString SelPort;

		CString strEnviar="";
		char enviar[34]="";
		int i;
		int err1;


		if(m_ComboBoxCOM.GetCurSel()!=CB_ERR)
		{
			m_ComboBoxCOM.GetLBText(m_ComboBoxCOM.GetCurSel(),SelPort);
			//com1.Init("COM3");
			com1.Init(SelPort);
			//com1.IsPortValid();
			if (com1.IsPortValid()==TRUE)
			{
				com1.SetBaud(115200);

				init=1;

				pFileEncoder = "ADC.txt";
				fichEncoder.Open( pFileEncoder, CFile::modeCreate |
CFile::modeWrite );

				m_ButtonInicio.EnableWindow(FALSE);

				activar=1;

				//Borramos el LCD
				for(i=0;i<33;i++)
				{
					enviar[i]=' ';
				}

				enviar[33]=0X0D;
				enviar[34]=NULL;
				err1 = com1.Send(34,enviar);
			}
			else
			AfxMessageBox(_T("El puerto seleccionado no es válido"),
MB_ICONERROR | MB_OK);
		}
		else
```

```
		MessageBox("No se a seleccionado ningún puerto
COM",NULL,MB_ICONWARNING);

}

void CPrueba1Dlg::OnButtonEnviar()
{
	// TODO: Add your control notification handler code here

	CString strEnviar="";
	int longitudStringEnviar;
	char enviar[34]="";
	int i;

	m_Enviar.GetWindowText(strEnviar);
	longitudStringEnviar = strEnviar.GetLength();

	if(longitudStringEnviar>32)
	{
		enviar[0]='E'; enviar[1]='R'; enviar[2]='R'; enviar[3]='O';  enviar[4]='R';
		for(i=5;i<32;i++)
		{
			enviar[i]=' ';
		}

		enviar[33]=0X0D;
		enviar[34]=NULL;

		int err1 = com1.Send(34,enviar);
	}

	if(longitudStringEnviar<33)
	{
		for(i=0;i<longitudStringEnviar;i++)
		{
			enviar[i]=strEnviar[i];
		}

		for(i=longitudStringEnviar;i<33;i++)
		{
			enviar[i]=' ';
		}

		enviar[33]=0X0D;
		enviar[34]=NULL;
		int err1 = com1.Send(34,enviar);
	}

	UpdateData(FALSE);
}

void CPrueba1Dlg::OnSalir()
{
	// TODO: Add your control notification handler code here
```

```
	//Cerramos el puerto COM seleccionado y cerramos el fichero de recogida de datos del
encoder
	com1.Close();
	fichEncoder.Close();

	m_ButtonSalir.EnableWindow(FALSE);
	m_ButtonEnviar.EnableWindow(FALSE);
	m_ButtonEnviarVelocidad.EnableWindow(FALSE);
	m_ButtonRecibirCAD.EnableWindow(FALSE);
}

void CPrueba1Dlg::OnEnviarVelocidad()
{

	//Si no queremos enviar datos al motor con el joystick, lo podemos hacer manualmente
	//rellenando las casillas y pulsando el botón Enviar Velocidad

	// TODO: Add your control notification handler code here
	char enviar[64];
	CString Velocidad_Left;
	CString Velocidad_Right;
	CString Direccion_Left;
	CString Direccion_Right;


	m_Velocidad_Left.GetWindowText(Velocidad_Left);
	m_Direccion_Left.GetWindowText(Direccion_Left);

	m_Velocidad_Right.GetWindowText(Velocidad_Right);
	m_Direccion_Right.GetWindowText(Direccion_Right);

	//Si la longuitud del dato puesto no es 3, no se envia ningún dato al los motores
	if((Velocidad_Left.GetLength()==3)&&(Velocidad_Right.GetLength()==3))
	{
		enviar[0]='9';

		enviar[1]=Direccion_Right[0];

		enviar[2]=Velocidad_Left[0]; enviar[3]=Velocidad_Left[1];
	enviar[4]=Velocidad_Left[2];

		enviar[5]=Direccion_Left[0];

		enviar[6]=Velocidad_Right[0]; enviar[7]=Velocidad_Right[1];
enviar[8]=Velocidad_Right[2];

		enviar[9]=0X0D;		enviar[10]=NULL;

		int err1 = com1.Send(10,enviar);
	}
}

void CPrueba1Dlg::OnRecibirCad()
{
	// TODO: Add your control notification handler code here
```

```
	//Activación y desactivación del CAD, para ello activamos y desactivamos
	//el temporizador de recibir datos del PIC
	switch (bActivarCaptura)
	{
		case FALSE:
		{
			KillTimer(IDT_CAD);
			bActivarCaptura = TRUE;
			m_ButtonRecibirCAD.SetWindowText("Iniciar Captura");

		}
		break;

		case TRUE:
		{
			SetTimer(IDT_CAD,25,NULL);//Recepción de datos del PIC al PC
			bActivarCaptura = FALSE;
			m_ButtonRecibirCAD.SetWindowText("Parar captura");
		}
		break;
	}
}



void CPrueba1Dlg::OnActivarJoystick()
{
	// TODO: Add your control notification handler code here

	//Activación y desactivación del joystick, para ello activamos y desactivamos
	//el temporizador del envio de datos para el motor

	switch (bActivarJoystic)
	{
		case FALSE:
		{
			KillTimer(IDT_MYTIMER);
			bActivarJoystic = TRUE;
			m_Activar_Joystick.SetWindowText("Activar Joystick");
		}
		break;

		case TRUE:
		{
			SetTimer(IDT_MYTIMER,100,NULL);//Timer del envio de datos del PC al
PIC
			bActivarJoystic=FALSE;
			m_Activar_Joystick.SetWindowText("Desactivar Joystick");
		}
		break;
	}

}
```

```
void CPrueba1Dlg::OnBorrarLcd()
{
	// TODO: Add your control notification handler code here

	//Borra el LCD rellenando todos los caracteres con espacios
	char enviar[64];

	for (i=0; i<33;i++)
	{
	enviar[i]=' ';
	}
	enviar[33]=0X0D;
	enviar[34]=NULL;

	int err1 = com1.Send(34,enviar);
}
```

8.1.4 Timer

```
LRESULT CPrueba1Dlg::DefWindowProc(UINT message, WPARAM wParam, LPARAM lParam)
{
	// TODO: Add your specialized code here and/or call the base class

	switch( message )
   {
	case WM_ACTIVATE:
	   if( WA_INACTIVE != wParam && _pJoystick )
	   {
	      // Make sure the device is acquired, if we are gaining focus.
	      _pJoystick->Acquire();
	   }
	   return TRUE;

	case WM_TIMER:
	   // Update the input device every timer message
	   if( FAILED( UpdateInputState((HWND)this->m_hWnd) ) )
	   {
	      KillTimer(0);
	      AfxMessageBox( _T("Error Reading Input State.The sample will now exit.DirectInput
Sample"),
	                  MB_ICONERROR | MB_OK );
	      EndDialog(TRUE );
	   }
	return TRUE;

	/*//Con esta case no funciona ni el gamepad ni la captura de datos
	            case WM_COMMAND:
	   switch( LOWORD(wParam) )
	   {
	      case IDCANCEL:
	         EndDialog( 0 );
	         return TRUE;
	   }*/

	case WM_DESTROY:
```

```
        // Cleanup everything
        KillTimer( 0 );
        FreeDirectInput();
        return TRUE;
    }

        return CDialog::DefWindowProc(message, wParam, lParam);
}


void CPrueba1Dlg::OnTimer(UINT nIDEvent)
{

        // TODO: Add your message handler code here and/or call default

        //Temporizador del envio de datos a los motores
                if (nIDEvent == IDT_MYTIMER)
                {
                        CString Velocidad_Left;
                        CString Velocidad_Right;
                        int iVelocidad_Left;
                        int iVelocidad_Right;

                        m_TxtAxisX.GetWindowText(Velocidad_Left);
                        m_TxtAxisZ.GetWindowText(Velocidad_Right);


                        //Dirección cambia poniendo negativo o positivo
                        iVelocidad_Left = atoi(Velocidad_Left);
                        iVelocidad_Right = atoi(Velocidad_Right);



                        if (iVelocidad_Left < 0)
                        {
                                m_Direccion_Left.SetWindowText(CString("0"));
                        }
                        else
                        m_Direccion_Left.SetWindowText(CString("1"));


                        if (iVelocidad_Right < 0)
                        {
                                m_Direccion_Right.SetWindowText(CString("0"));
                        }
                        else
                        m_Direccion_Right.SetWindowText(CString("1"));


                        Velocidad_Left.Format("%03d",abs(iVelocidad_Left)); // int to CString
                        Velocidad_Right.Format("%03d",abs(iVelocidad_Right));


                        m_Velocidad_Left.SetWindowText(Velocidad_Left);
                        m_Velocidad_Right.SetWindowText(Velocidad_Right);
```

```
                        //OnEnviarVelocidad();

                        char enviar[32];
                        CString Direccion_Left;
                        CString Direccion_Right;

                        m_Direccion_Left.GetWindowText(Direccion_Left);
                        m_Direccion_Right.GetWindowText(Direccion_Right);



                        enviar[0]='9';

                        enviar[1]=Direccion_Right[0];

                        enviar[2]=Velocidad_Left[0]; enviar[3]=Velocidad_Left[1];
        enviar[4]=Velocidad_Left[2];

                        enviar[5]=Direccion_Left[0];

                        enviar[6]=Velocidad_Right[0]; enviar[7]=Velocidad_Right[1];
enviar[8]=Velocidad_Right[2];

                        enviar[9]=0X0D;         enviar[10]=NULL;

                        int err1 = com1.Send(10,enviar);

                }

                //Temporizador de la recepción de datos del CAD de los encoder
                if (nIDEvent == IDT_CAD)
                {

                        char enviar[64];

                        enviar[0]='8';

                        enviar[1]=0X0D;         enviar[2]=NULL;

                        int err1 = com1.Send(2,enviar);

                        int T,fin;

                        kbhit = com1.Receive(32,RecibirChar);

                        if(kbhit==1)
                        {
                                if(RecibirChar[0]=='0')//Recibimos datos del encoder
                                {
                                        //m_Recibir.SetWindowText(RecibirChar);
                                        Encoder = RecibirChar;

                                        T = Encoder.Find('T');
                                        fin = Encoder.Find(0X0D);
```

```
                                        if((fin!=-1)&&(T!=-1))//Si hemos recibido algo distinto de
NULL
                                        {

        m_EncoderRight.SetWindowText(Encoder.Mid((T+1),(fin-(T+1))));
                                                m_EncoderLeft.SetWindowText(Encoder.Mid(1,(T-
1)));


                                                m_EncoderRight.GetWindowText(EncoderRight);
                                                m_EncoderLeft.GetWindowText(EncoderLeft);

                                                GuardarEncoder = EncoderRight + ',' + EncoderLeft
+ '\n';

                                                longitudRecepcion=GuardarEncoder.GetLength();

                                                fichEncoder.Write
(GuardarEncoder,longitudRecepcion);


                                                Encoder.Empty();

                                                for(int i=0; i<33; i++)
                                                {
                                                        RecibirChar[i]=NULL;
                                                }
                                        }
                                }
                        }
                }

        CDialog::OnTimer(nIDEvent);
}
```

8.2 CCS

```
#include <18F4550.h>
#device adc=8
#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay(clock=12000000)
#USE RS232(BAUD=115200, XMIT=PIN_C6, RCV=PIN_C7, stream=PC)

#include <stdlib.h>
#define use_portb_lcd TRUE
#include <lcd.c>

#define MOTOR_RIGHT_0
#define MOTOR_RIGHT_1

#define MOTOR_LEFT_0
#define MOTOR_LEFT_1

long value0=0;
long value1=0;

char string_buffer[32];

// Declaración de Funciones ////////////////////////////////////////////////
void DelBuffer(void);
void Avanzar_Left(void);
void Retroceder_Left(void);
void Avanzar_Right(void);
void Retroceder_Right(void);


char string_buffer_Init[16];
char string_buffer_End[16];

int contador;

char velocidad_Right[4];
char velocidad_Left[4];

//PROGRAMA PRINCIPAL
void main(void) {

  lcd_init();

  setup_adc_ports(AN0_TO_AN1|VSS_VDD);
  setup_adc( ADC_CLOCK_INTERNAL );

  setup_ccp1(CCP_PWM);
  setup_ccp2(CCP_PWM);
  setup_timer_2(T2_DIV_BY_16, 255, 1);

  enable_interrupts(GLOBAL);

  while (TRUE)
  {
```

```
    if(kbhit(PC))
    {
      gets(string_buffer,PC);
      if((string_buffer[0]!='9')&&(string_buffer[0]!='8'))
      {
        for(contador=0;contador<16;contador++)
        {
          string_buffer_Init[contador]=string_buffer[contador];
          string_buffer_End[(contador)]=string_buffer[(contador+16)];
        }
        printf(lcd_putc,"\f%S\n%S",string_buffer_Init,string_buffer_End);
        DelBuffer();
      }
      if(string_buffer[0]=='8')
      {
        set_adc_channel(0);
        value0 = Read_ADC();
        set_adc_channel(1);
        value1= Read_ADC();
        fprintf(PC,"0%4LuT%4Lu\r",value0,value1);
      }

     if(string_buffer[0]=='9')
      {
        if(string_buffer[1]=='0')
        {
          Avanzar_Left();
        }
        if(string_buffer[1]=='1')
        {
          Retroceder_Left();
        }

        if(string_buffer[5]=='0')
        {
          Avanzar_Right();
        }
        if(string_buffer[5]=='1')
        {
          Retroceder_Right();
        }
        velocidad_Left[0]=string_buffer[2];
        velocidad_Left[1]=string_buffer[3];
        velocidad_Left[2]=string_buffer[4];
        velocidad_Left[3]=NULL;

        velocidad_Right[0]=string_buffer[6];
        velocidad_Right[1]=string_buffer[7];
        velocidad_Right[2]=string_buffer[8];
        velocidad_Right[3]=NULL;

        set_pwm2_duty(atoi(velocidad_Left));
        set_pwm1_duty(atoi(velocidad_Right));
      }
    }
}
```

```
}

void DelBuffer(void)
{
  int i;
  for(i=0; i<32; i++)
  {
       string_buffer_Init[i]=NULL;
       string_buffer_End[i]=NULL;
  }

  for(i=0;i<3; i++)
  {
    velocidad_Left[i]=NULL;
    velocidad_Right[i]=NULL;
  }

}

void Avanzar_Left(void)
{
     output_high(MOTOR_LEFT_0);
     output_low(MOTOR_LEFT_1);
}

void Retroceder_Left(void)
{
     output_low(MOTOR_LEFT_0);
     output_high(MOTOR_LEFT_1);
}

void Avanzar_Right(void)
{
     output_high(MOTOR_RIGHT_0);
     output_low(MOTOR_RIGHT_1);
}

void Retroceder_Right(void)
{
     output_low(MOTOR_RIGHT_0);
     output_high(MOTOR_RIGHT_1);
}
```