Lesson 6: DirectX Software Development Kit
Focus on Event Driven programming and Joystick Input
By Mark Aull, 2009

Part 1: Getting access to the DirectX libraries

Important Note: this was written to allow use of DirectX in the OCC labs. You can install it on your own computer instead of using this section.

Open “My Computer” and go to “Tools>Map Network Drive”.

Enter this path:\ucrobotics1.rhod.uc.edu\Microsoft DirectX SDK (June 2008) into the “Folder” box, then click “different user name”, and enter the usual.
After clicking “OK” and “Finish”, you should see a window like the following:

**Connect As...**

By default, you will connect to the network folder as RH628A\Administrator. To connect as another user, enter their user name and password below.

- **User name:** robotfan
- **Password:** ********

**OK**  **Cancel**

After clicking “OK” and “Finish”, you should see a window like the following:
Part 2: Event driven programming concepts using the DirectX Joystick program

Go to “Z:\Samples\C++\DirectInput\”, copy the folder “Joystick” somewhere else and open “joystick_2005.sln” from that somewhere else.

In Visual Studio, add “Z:\Include” to the “include” path and “Z:\Lib\x86” to the library path as demonstrated in lesson1. Then ensure that you can build the project, and run it.

Inside “joystick.cpp”, there is a lot of code. The following functions are overhead that you almost never need to look at:
For starting the joystick:
BOOL CALLBACK EnumObjectsCallback( const DIDEVICEOBJECTINSTANCE* pdidoi, VOID* pContext );
BOOL CALLBACK EnumJoysticksCallback( const DIDEVICEINSTANCE* pdidInstance, VOID* pContext );
HRESULT InitDirectInput( HWND hDlg );
HRESULT SetupForIsXInputDevice();
bool IsXInputDevice( const GUID* pGuidProductFromDirectInput );

and for closing it:
VOID FreeDirectInput();
void CleanupForIsXInputDevice();
We will start with the only exception to that “almost”: in the function

```
HRESULT InitDirectInput( HWND hDlg )
```

There is a segment of code that kills the program when there’s no joystick. This is inconvenient, so we will fix it.

```c
// Make sure we got a joystick
if( NULL == g_pJoystick )
{
    MessageBox( NULL, TEXT( "Joystick not found. The sample will now exit." ),
               TEXT( "DirectInput Sample" ),
               MB_ICONERROR | MB_OK );
    EndDialog( hDlg, 0 );
    return S_OK;
}
```

The easiest way to do this is to comment out the lines that start with “EndDialog” and “return”, put “else {“ after the }, and put another } just before the “return” at the end of “InitDirectInput”. Now the program should run normally when a joystick is present, and just sort of sit there when one isn’t.

Now on to the important functions:

```
int APIENTRY WinMain( HINSTANCE hInst, HINSTANCE, LPSTR, int )
```

is the function that is run first, and is a good place to put initialization code.

```
INT_PTR CALLBACK MainDlgProc( HWND hDlg, UINT msg, WPARAM wParam, LPARAM lParam );
```

is a giant switch statement which is run constantly to capture and handle “messages” which are generated by things like buttons being clicked. We therefore modify this function whenever we want to add a button or other control to the code.

```
HRESULT UpdateInputState( HWND hDlg );
```

is the function that actually polls the joystick and displays its output. You may have noticed that it is run by MainDlgProc when it gets the message “WM_TIMER”. You may also have noticed that when the message “WM_INITDIALOG” is received (meaning the dialog box is initialized), the code

```
SetTimer( hDlg, 0, 1000 / 30, NULL );
```

is run. This starts a timer that sends a “WM_TIMER” message every 30\textsuperscript{th} of a second, which then runs the “UpdateInputState” function. We will add code here that we want to automatically run frequently.

Next we will add a text box to the demo, and set up that box to display “left” when the joystick is pushed to the left, and “right” when pushed to the right, and “center” for a deadband in the middle.

Since this operation must be performed whenever the joystick moves, the most logical place to put it is in UpdateInputState. About the first third of UpdateInputState obtains
the joystick state, and the other two-thirds display that information on the dialog box. Scroll to the lines between these two sections, and look at the lines

    StringCchPrintf( strText, 512, TEXT("%ld"), js.lX);
    SetWindowText( GetDlgItem( hDlg, IDC_X_AXIS ), strText);

The command “StringCchPrintf” does exactly the same thing as the command “sprintf” that we have already learned; it is just an example of Microsoft trying to make its code as incompatible with standard C++ as possible. We will now demonstrate replacing their weird commands with standard ones. First, go to the top of the UpdateInputState function, and find the declaration of strText:

    TCHAR strText[512] = {0};

Below this, add the line

    char strTexta[512];

Then, go back to the lines we were looking at before. Modify them to look like the following:

    sprintf( strTexta, "%ld", js.lX);
    SetWindowTextA( GetDlgItem( hDlg, IDC_X_AXIS ), strTexta);

When you rebuild your code, it should behave exactly as before. Note that instead of changing SetWindowText to SetWindowTextA, you can use the instructions in lesson4 for disabling unicode, however you would then have to make these modifications for the entire file.

Now that you know what that first line is doing (writing js.lX into a string, strTexta), the name of the second function is pretty intuitive: it writes the string that was just created somewhere. The GetDlgItem command gets a “Handle” to the object referred to as “IDC_X_AXIS”. This is where the string is written.

To accomplish our goal of writing “left”, “right”, and “center”, we will use this same functionality: start with an “if” statement on js.lX, and in each branch, add a SetWindowTextA command. Next you will probably ask “but where are we writing this text?”. We have to create an appropriate space.

Open the resource editor, either by clicking on its tab, next to the solution explorer, or selecting it from the “view” menu. I believe some versions of Visual Studio hide this option inside of a sub-menu, so you may have to search around for it.

In the resource editor, open “joystick.rc” and “dialog”, and double-click “IDD_JOYSY_IMM”.


You should now see a window that looks like the program when it is running. Use the small blue square under the window to make it taller. Then select “static text” in the toolbox (open the toolbox from the view menu if it didn't open by itself like it should have). Draw a small rectangle at the bottom of the window, in the new space you created by resizing it. Right-click on it, and select properties, and at the bottom of the properties box, change its ID to IDC_TURN. We now have a place to display text, so the text below will perform the desired operation:

```c
if (js.lX>15)
    SetWindowTextA( GetDlgItem( hDlg, IDC_TURN ), "right" );
else if (js.lX<-15)
    SetWindowTextA( GetDlgItem( hDlg, IDC_TURN ), "left" );
else
    SetWindowTextA( GetDlgItem( hDlg, IDC_TURN ), "center" );
```
Run the program and move the joystick to ensure that it operated as desired. If you are going through this lesson without a joystick, you can check this later.

Next we will add a button to the dialog box and demonstrate how to use it. Go back to the dialog editor, select the button tool from the toolbox, and draw another small rectangle.

![joystick_2005 - Microsoft Visual Studio](image)

I have already said the the MainDlgProc function handles things like buttons. Scroll up to this function, and find the “WM_COMMAND” case. You will notice another switch statement inside it, which determines what send the command. The only thing in it now is the handler for the “Exit” button, which has the ID IDCANCEL (check its properties). Add a case above or below it for IDC_BUTTON1 (which is by default, the ID of the button you made).

To test the button, we will use a simple message box. The code is below:
case IDC_BUTTON1:
    MessageBox(hDlg,"world","hello",MB_OK);
    return true;

Now if you run the code, it should have the new button, and when you click it, it should produce a message box that says “hello world” with an OK button below it.

Finally, go back to the dialog editor and add a slider bar to the dialog box. Note that its ID is IDC_SLIDER2 rather than IDC_SLIDER1 as you might expect. This is because Microsoft used IDC_SLIDER0 and IDC_SLIDER1 as displays for joystick sliders. Also add another static text box, and change its ID to IDC_SLIDEVAL.

I chose to handle the slider inside of the UpdateInputState function. I added the following lines just after the declarations at the top of the function so they would still run without a joystick present.

    int slider=SendMessage(GetDlgItem(hDlg, IDC_SLIDER2 ),TBM_GETPOS,0,0);
    sprintf(strTexta,"%03d",slider);
    SetWindowTextA( GetDlgItem( hDlg, IDC_SLIDEVAL ), strTexta );

These lines get the position of the slider as an integer (from 0 left to 100 right), write that integer to a string, and write that string to the window.

Assignment:
You guessed it: write a program to drive the robot on joystick control.
You can declare the Galil variable (the HANDLEDMC) as a global (this file already has a space labeled for them; g_pDI and g_pJoystick are in it).
Add the commands to start the Galil to WinMain.
Use a slider to control the maximum speed of the Galil, and add the commands to the Galil to UpdateInputState. Use a maximum speed of 20000: the value of a slider varies between 0 and 100, and the value of the joystick axes vary between -1000 and 1000, so a formula involving an axis value can be multiplied by the slider value and divided by 5 to achieve this maximum speed.
Commands stopping the Galil can be added to either the end of WinMain (before the return command) or in case WM_DESTROY: in MainDlgProc.