Mechatronics I Laboratory Exercise: An Introduction to Lab Windows/CVI

As a controls engineer, you will often find yourself in need of customized data acquisition systems to fit the plant and control scheme that you are attempting to implement. Lab Windows is a software package that is deigned to help the scientist build user friendly instrumentation/control applications in a short amount of time.

In this lab, you will build the beginnings of your own person data acquisition program that you can use for the rest of the Mechatronics series. Your program will display a graphic instrumentation control panel (called the "user interface", UIR) complete with buttons, knobs, and gauges that the user can read and adjust with a keyboard or mouse. It is assumed that you have some familiarity with Windows style applications, but if you have questions, your TA can help. The application code itself will be written in ANSI C, but experience in the C programming language is not necessary. As with all computer programming, make sure you save your file(s) often, and always keep a back up on your own floppy disk. First, make a folder with your name in the C:\CVI\PROGRAMS\MECH_I directory. Chose a name from your group or if you have formed a group name, use your group name. This can be done in a file viewer window, which appears after double clicking the computer icon on the desktop, or by selecting "WINDOWS EXPLORER" from the START button menu. If you have not done this before, ask your lab partners or TA for assistance. This will be the folder you will use to store your data and programs all year. It will not be protected from other students, so make a backup of your folder at the end of every lab session.

For those who have not programmed in C before, a quick word about the architecture of a C application is in order. A C program is usually a collection of functions; each function performs a specific task or determines a value from steps of computations. The primary function in a program is the first to be executed, and other functions are called within it as the program progresses. When a function is called, its name is followed in parentheses by the parameters necessary for the function to perform its task. The general form is : FUNCTION_NAME(PARAMETER_1, PARAMETER_2, . . .). One advantage of Lab View is that it is easy to implement functions into the program since the process is graphical and menu-driven for the user, and the exact syntax for the C program is entered automatically. Equations are very intuitive in C; mathematical operations using variables in a program are not different from standard math notation.

The C variable types we will mainly be using are: integer (int), floating-point number (float), and double-precision floating-point number (double). All variables used in a program must be initialized by listing the variable type before the variable name. Sometimes a function name will be preceded by one of these type descriptors. In this case, it represents the type of variable that the function is returning.

Some parts of a C program are enclosed in {brackets}. These serve to isolate certain parts of the program. They are used to represent the starting and ending points of a function. When these brackets enclose a number of lines of code following an *if* statement, it means that these lines are executed when the *if* statement holds true; otherwise, they are skipped. Other bracketed sections of code follow *for* or *while* loops, which we will see later in upcoming programs. In the case of *for* or *while* loops, the lines enclosed in brackets will be executed repeatedly until the loop is terminated.

The first step in building a CVI application involves a bit of file creation and management.

Start by opening CVI by double clicking on the icon on the desktop or under the START menu. Create a new project file by selecting *FILE/NEW/Project*. When the "untitled" project window appears, save the new project in **YOUR** directory (*FILE/SaveAs*) using the appropriate name (e.g., lab2.prj). Your new project will eventually contain a C program (with the file extension ".c") that will contain your instructions telling the computer what to do. It will also contain a file that controls what your screen shows, such as which knobs, gages, and readouts to display. This file has the file extension ".uir" for "user interface". Finally, your project will contain a header file with the extension ".h" that tells the computer how the other files interact. We will start by creating the necessary C program.

A basic skeleton program called "template.c" already exists in the C:\CVI\PROGRAMS-\LABS\TEMPLATE directory. Open this file with *FILE/Open/Source(*.c)*. Instead of making changes to this basic file, save it with a different name in your directory (*FILE/Save-As*). Remember to use an appropriate name. It is often useful to use the same basic name for all the files in your project (e.g., lab2.prj, lab2.c, lab2.uir, lab2.h). Also create a new user interface file with *FILE/New/UserInterface(*.uir)*. Again use (*FILE/SaveAs*) to save the .uir file in your directory with an appropriate name (such as lab2.uir). These files must be added to your project by clicking on their windows and selecting *FILE/Add File To Project*. Do this in both the .c and .uir windows. These file names should now appear in your CVI project window. There is one more file to add: the header file mentioned earlier. Select *FILE/Open/Include(*.h)* and locate the header (.h) file that belongs to your project (i.e. lab2.h). Add this to your project in a manner similar to the other two files. Your project window should now show thee files (such as lab2.c, lab2.uir, and lab2.h).

If it is not open already, open your .c file by double clicking on it in the project window. Notice that the first lines say to "*#include*" a list of .h files. These are files full of references to other files that your program may want to use. One of the "includes" specifies "*name.h*". Change this to yourprogram.h so that it will include **your header** file (for example, lab2.h). Now look a few lines down and change the line:

```
panel_handle=LoadPanel(0,''name.uir",PANEL);
```

to:

```
panel_handle=LoadPanel(0,''yourname.uir",PANEL);
```

where ***yourname*** is the name of your uir file (for example, lab2.uir).

The first thing that a C program does after including header files is to call the function labeled "main". This main function now calls the C function created by your .uir file whose only job is to display your user interface and wait for your input. When you provide an input (with a mouse or keystroke) the .uir program figures out what you are telling it to do and calls the appropriate function in your C program. For instance, you may want a button on your user interface labeled "QUIT". When you press this button, the uir program will call the "quit" function in your .c file. This quit function closes the uir window and stops the program. A quit function has been included already in the .c template file that you copied. Note that the quit function is itself just more calls to other functions that do the actual work. That is the essence of C programming, building your own program by calling other programs and functions that have already been written. This will become clear as we proceed.

Now let us begin to build your user interface. Open your .uir file and then select *Create/Panel*. A gray square will now pop up, labeled "untitled panel". This is what you

will see when you run your program, and it represents the front control panel of your data acquisition device. Resize the panel to any size you want by dragging the corners. We will put some knobs and gauges on the panel just like if we were building an analog data acquisition device on a workbench. Double click anywhere on the panel to pop up an information window about the panel. Through this window, you can change the attributes of the panel such as the title, color, etc. Play with this as you like, you can always change it later. **One attribute to leave alone is the box labeled "constant name: PANEL"**. PANEL is the formal name that the C program uses to reference your control panel (note the name PANEL in the program lines listed above.)

For this lab, we are going to make a program that will read in a voltage from a voltage source (such as a voltage divider circuit), subtract it from a base value that you choose (set point), then output and display the difference (error), just like you do for a simple proportional controller.

In the .uir window, create a knob that you will use to choose the set point. Do this by selecting *Create/Numeric* and select a knob or dial. A knob will appear on the panel and you can move it where you like. Double click on the knob and give it a short descriptive name (SETPOINT or something similar) in the ***constant name*** box. Be careful, CVI is case sensitive! You can also put a title on the knob, and it does not have to be the same as its constant name.

Create a dial, slider or gauge to display the input voltage that the computer reads, by selecting *Create/Numeric* and choosing a dial or slider. Double click on it and give it an appropriate name in the constant name box, title, etc (such as VIN). Now create a third item, a dial, slider, or gage, which will indicate the difference (error) between the set point and the input voltage. Give it an appropriate name, title, etc.

So far we've made a knob to tell the program what we want the set point to be, and two dial displays so that the program will be able to tell us what it reads and what the difference is. Now create two buttons (*Create/ToggleButton*), one to start the program, and one to quit. Name and title the buttons as before, but do one more thing in addition. In the box labeled "**Callback Function**", put a function name like "**START_FUNC**" and "**QUIT_FUNC**". This tells the uir function to call the "**START_FUNC**" or "**QUIT_FUNC**" functions in your .c file when each button is pressed. Now save the .uir file.

Saving the .uir file automatically modifies and opens the .h file. **Notice the long comments at the first that warns you not to ever modify it!** A nice feature about Lab Windows is that is automatically takes care of your .h file, making it so you do not have to make any modifications. In this .h file, there should be two lines at the bottom similar to:

```
int QUIT_FUNC(int panel, int control, in event,
     void *callbackData, int eventData1, int eventData2);
int START_FUNC(int panel, int control, in event,
     void *callbackData, int eventData1, int eventData2);
```

These two lines reference the quit and start functions to the uir when either button is pressed. Go back to your .c file and notice the form of the quit function already written. It starts with the "*int QUIT. . .*" line similar to the "*int QUIT. . .*" line from the header file, without the semi-colon at the end. Two braces enclose the rest of the function. It ends with return(0) and has an inner set of braces prefaced by: if (*event==EVENT_COMMIT*). Set up your new start function in the same manner by copying all of the quit function and substituting the "*int START_. . .*" into "*int QUIT*". Also remove the line in your new start

function that reads *QuitUserInterface(0)*;. This is left over from the old quit function. Your start function should look as below:

```
int START_FUNC(int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
      {
            if (event==EVENT_COMMIT)
            {
            }
            return(0);
      }     /*End of START_FUNC
```

We will now insert code between the inner set of braces of the START_FUNC to do the actual work of the program. The first thing to have the start function do is to find out what you want the set point to be. Place your cursor on the line between the inner braces and with your mouse select the menu path: *Library/User Interface/Controls/Graphics/Stripcharts/General-/Get Control Value*. A new window will pop up. In the box labeled "*panel handle*", enter the formal name of your panel, "*PANEL*". In the box labeled "*Control ID*", enter the path to the set point control, that is "*PANEL_SETPOINT*". In the box labeled, "*Value*", enter the name of the variable that you want to hold the value from the set point knob, such as "*&setpoint*". This will be the variable that the C program will use to determine the set point. Whenever you use a function call window in CVI Lab Windows, you can check if you need to use a pointer symbol (&) by clicking the right mouse button on the box that is to contain the variable name. If the info window that pops up says "*passed by reference*", the function expects a pointer.

Notice the open box at the bottom of the function window. As you enter names in the panel handle, Control ID, and value boxes, a line of C code is modified in the bottom box. This line of code is in terms that the computer understands, and it will be inserted into your .c file wherever you have left your cursor. To do the insertion, select: *Code/Insert Function Call*. Now go back to your .c file and check to make sure that it put the code in the right place. Now, position the cursor on the next line where the next function call will be placed. What that line of code you just inserted says is: "*Get the value that the user has selected on the set point knob that is located on the panel and assign that value to the variable named setpoint*". Lab Windows makes it easy to use functions because the housekeeping of code syntax is taken care of by the function call windows.

Next, let's have the program find out what voltage is being input to the computer by the outside world. The last function call involved the user interface and the function call code was found in the *User Interface* folder of the *Library*, so it would make sense that to sample an analog voltage from the outside, you would find the correct function call in the *Data Acquisition* folder of the library. The full path is: *Library/DataAcquisition/Analog Input/Single Point/Measure Voltage*. The beginnings of the function call in the box at the bottom of the window should say "*AI_VRead. . . etc . .*". When you have found that window, set the board slider to "**1**", pick an input channel (**0** is fine), set the gain to one, and type in a name with a pointer symbol (I usually pick *&vin*) in the voltage box. Now insert that function call into your .c file. It should appear below your "*GetCtlVal*" function call.

The computer, when running your program, now knows the set point you have chosen and the voltage at the channel you have selected. To find the difference between these two values and assign it to a variable named "*error*", an appropriate next line of code would be:

```
     error=setpoint-vin;
```

Using similar techniques as before, go to the library and insert a function call that displays the value of the *vin* and the *error* (using *SetCtrlVal*) and then another call that outputs the value of the error as an analog voltage (using Generate Voltage). Remember to check if each function expects a pointer (the "&" symbol)!

One thing left to do before your program can compile is to declare the variables at the beginning of the program. Declare them as double floating point integers, or "*double*"s. (If you were using a variable to count, you might declare it as an integer or "int". Different functions expect different types of variables.) Your .c file, with the additional functions calls and variable declarations, should look similar to following:

```
/*   TEMPLATE.C
     Mechatronics Student CVI Lab Windows Template File
     Use this template for CVI Lab Windows applications
     Refer to Mechatronics Lab:  An Introduction of Lab
     Windows/CVI for more info */

#include <ansi_c.h>
#include <formatio.h>
#include <dataacq.h>
#include <userint.h>
#include ''lab2.h"

int panel_handle;
double vin, setpoint, error;

main()
{
     panel_handle = LoadPanel (0, ''lab2.uir", PANEL);
     DisplayPanel (panel_handle);
     RunUserInterface();
}

int QUIT_FUNC(int panel, int control, int event,
     void *callbackData, int eventData1, int eventData2)
{
     if (event==EVENT_COMMIT)
     {
          QuitUserInterface(0);
     }
     return(0);
} /* End of QUIT_FUNC */

int START_FUNC(int panel, int control, int event,
     void *callbackData, int eventData1, int eventData2)
{
     if (event==EVENT_COMMIT)
          {
```

```
            GetCtrlVal(PANEL,PANEL_SETPOINT,&setpoint);
            AI_Vread(1,0,1,&vin);
            error=setpoint-vin;
            SetCtrlVal(PANEL,PANEL_VIN,vin);
            SetCtrlVal(PANEL,PANEL_ERROR,error);
            AO_Vwrite(1,0,error);
            }
      return(0);
} /* End of START_FUNC */
```

You have now written a simple data acquisition and digital controller program. To run it, save your project and then select: *Run/Run Project* from any of the windows. Lab Windows will attempt to compile your project and notify you of any errors. If there are errors, try debugging them, starting with the first error on the list. If you cannot figure out why it won't compile, ask a classmate or your TA for help.

Laboratory Exercise

After your program successfully compiles, your user interface panel should appear on the monitor. Construct a voltage divider circuit that supplies an output voltage between 0 and 5 volts (Refer to "Computer Data Collection Lab" for voltage divider circuit diagram). Connect the output from the voltage divider to your chosen *Vin* channel. Active the "START" button using the mouse. The computer should take one sample from the *Vin* channel, then output and display the difference between the set point and the input voltage. Use a voltmeter to verify that the output voltage is consistent with the error displayed on the gauge.

Now that you are a CVI Lab Windows expert, investigate what else you can do to add more functions, change colors, ranges, or sizes of the display features, and customize your data acquisition program. Browse through the library to see what functions are available.

Post-Lab Report

Explain and include a copy of your C program, header file, and User Interface printout.