ECE 202 – LAB 1
INTRODUCTION TO LABVIEW

BEFORE YOU BEGIN

EXPECTED KNOWLEDGE

You should be familiar with the basics of programming, as introduced by courses such as CS 161.

PREREQUISITE LABS

None.

EQUIPMENT

Access to a PC with LabVIEW installed.

MATERIALS

None.

INTRODUCTION

LabVIEW is an automation software environment designed to be flexible enough to accomplish virtually any automation goal while still remaining easy to use. It is often found in robotics, data and instrumentational analysis, data acquisition, etc. The language “G”, developed by National Instruments, is based upon the C language and the library provided with the language is immense. There are add-on packages to accomplish a variety of goals such as internet development, instrument communication hardware, mathematical manipulation, and database management.

LabVIEW has been used to design programs for automation of industrial plants, in high tech facilities such as Intel and Tektronics, and in the space industry at the Jet Propulsion Labs in Pasadena. As you explore the LabVIEW environment, try to think of applications that might prove useful. Furthermore, you may encounter situations while doing the labs where you think that a program written in LabVIEW may be useful (think of repetitious tasks). In this case you are encouraged to write a program which might prove useful to yourself and others. Keep in mind that drivers are available to communicate with all of the lab equipment, so complete automation of any task is possible.

OBJECTIVES

LabVIEW is not the only automation software available. There are many competitors in this field. However, since automation software has become so widely used in industry, a basic understanding of its capabilities has become desirable. With that in mind this lab will
demonstrate a simple program design. The following lab will expand that design to include instrument communication. The objective is not to teach you how to program in LabVIEW, which would require a complete course of its own. Instead, we will focus on what automation software is and how it can help in a work environment.

**PRELAB**

*Answer Question 1.*

**STARTING LABVIEW**

1. Open the Labview program by selecting Start->programs->National Instruments LabVIEW->LabVIEW.

   The screen that appears offers several choices. Notice the Search Examples selection in particular. One of the nice things about LabVIEW is that just about anything you can think of has been done before. Even more examples are available at the National Instruments website at [www.natinst.com](http://www.natinst.com).

**BUILDING A SIMPLE APPLICATION**

Programs developed in LabVIEW are called Virtual Instruments or VIs. The concept behind virtual instrumentation is that the programmer should be able to easily develop an application that has the appearance and functionality of a standard instrument. In other words, a VI might have the same appearance as a power supply front panel, and may actually control a power supply located in some other building (or even some other city!). While the program is running, when the user changes the input on the computer front panel, a command is sent out on the bus line connecting to the instrument, and the instrument reads that command and responds accordingly. A scheme like this would be commonly used in harsh environments such as a "tool" room in a semiconductor facility (where exposure to deadly gases is possible). In the same environment you may also have a VI that reads values from gas monitors to detect a leak. In this case the instruments are sending out values that your program is reading. In the case where a leak is detected, your program would sound an alarm and possibly begin a venting or shutdown procedure. You can imagine how this scenario applies to other industries or applications.

In the next two labs we will start with a simple VI and eventually build a simple application that communicates with the DMM. Remember to save your final VI of this lab as it will be used in the next lab.

2. Select New VI to start a new session.

   The next screen to appear is the Front Panel. On this screen you can design the user interface for your program. Usually, an attempt is made to make the user interface appear similar to the equipment you are trying to automate.
3. Select the Windows menu item and click on Show Diagram (or simply click on the panel in
the background) to access the programming window where the functionality of the
application is coded.

4. Select Windows->Tile Left and Right.

This allows you to see what is happening on both screens at the same time.

5. Select the Front Panel and right-click anywhere on the background. By doing this you can
access the controls palette.

6. Place the mouse over the first menu item.

Notice that the name for that menu item (Numeric in this case) appears at the top of the menu.

7. Move the mouse over the Numeric Menu icon.

This exposes the selections for the Numeric Menu.

8. Select Digital Control by dragging and dropping it on the Front Panel where you want to
locate it.

You will notice that when you place a control on the front panel, a symbol is also placed on the
Diagram Panel.

**POINTER TOOL**

The pointer tool can be altered for different functionality by using the tab key.

9. Select the Front Panel and press the tab key several times.

The sequence observed for the pointer tool is as follows: the arrow (selector), the box with a line
through it (the lettering tool), the paintbrush (colors may be changed by right clicking on an
object and selecting the color of choice), and the hand tool (used to manipulate a control’s
values).

On the Diagram Panel the selections are the same except that a wiring tool is added which allows
the terminals to be wired together.

Note: You must select the appropriate panel before changing the pointer tool option.

10. Make a copy of your Front Panel object by selecting the arrow, moving the arrow over the
object, pressing <control> and left-clicking and holding on the object at the same time, and
dragging the new object to the new location. Alternatively, you can highlight the control
with the arrow tool and use the standard windows copy and paste procedure.

11. Right-click on the new object and select change to indicator.
Note: in LabVIEW a control can only be wired to an indicator. A control wired to a control, or indicator wired to an indicator results in an error.

12. Using the wiring tool in the Diagram Panel, click once on the terminal with the solid border (solid borders indicate controls and empty borders indicate indicators).

13. Move the mouse in any direction.

You will see a dashed line appear. This is the wire you will be attaching to your indicator. The wire will make a 90 degree turn to follow the wiring tool. Left-clicking at any point will “pin” the wire at that location. This process will continue until you have left-clicked on the indicator. A dashed line after you have clicked on the destination indicates an error. Notice that the indicator, the control, and the wire connecting them are all orange. In LabVIEW every data type has a separate color. For example, later you will be working with boolean expressions that use the color green. In large applications this allows the programmer to more easily see the relevant portions of the code. It also allows the programmer to readily determine what data type is requested by an indicator or generated by a control. The line in between is always the same color as the control to which it is connected. In a large program this allows the programmer to quickly trace the line.

Note: you can remove a broken line by selecting the arrow tool and single, double or triple left-clicking on the wire. Alternatively, you can remove all broken lines by pressing Ctrl-B.

**RUNNING THE SIMPLE PROGRAM**

14. Select the Front Panel.

15. Tab to the hand tool.

16. Move the tool into the display of the control and double click.

17. Enter a new number.

18. Press the arrow button on the toolbar in the top left hand corner of the Front Panel.

This button runs the program. You will notice that your indicator now reads the same as your control. This is the simplest of programs where the information is passed between two objects.

19. Print the front panel by selecting File->Print Window. Select the diagram window and print it out the same way.

**Answer Question 2.**

**A MORE COMPLEX PROGRAM**

This program will demonstrate how to design a simulated multithreaded program. Multithreading enables two or more things to happen simultaneously as compared to sequential operations in which only one application can run at a time. Our program will count loop
iterations in one thread and in the other thread we will be controlling the level of a liquid tank indicator.


21. On the front panel select the control palette (right-click on the background) and select Boolean->Rectangular Stop Button.

22. Place this object on the front panel (left-click at the desired location).

23. Right-click on the button and select Mechanical Action->Switch When Released.

24. Select Control Palette->Numeric->Digital Control and place it on the front panel.

Renaming a control (or indicator) can be done by simply typing in the name immediately after placing the control on the Front Panel (notice that the name is highlighted), or by selecting the lettering tool and placing it on the name of the control and left clicking once (to place the cursor), twice (to highlight a word), or three times (to select multiple words), and then entering the new name.

In the following steps substitute your last name for “Smith”.

25. Rename the digital control to “Smith: Number of Loops to Complete”.

26. Select Control Palette->Numeric->Digital Indicator and place it on the front panel.

27. Rename the indicator to “Smith: Current Iterations”.

28. Select Control Palette->Numeric->Digital Indicator and place it on the front panel.

29. Rename the indicator to “Smith: Current Loop Value”.

30. Controls and indicators may be moved by changing the cursor to the arrow, then dragging and dropping at the appropriate location. Alternatively, simply left-clicking to select the object and then moving the object with the keyboard arrows accomplishes the same task.

31. Place the objects in the desired locations. The Front Panel should now look similar to Figure 1.
32. Select the diagram window.

33. Select the functions menu (right-click on the background).

This menu provides all of the programming functions that one would normally encounter in C, and some additional functions for communication, mathematics, etc.

34. Select Structures->While Loop.

When you left-click to place it on the diagram window, left-click where you want the upper left corner to be and hold the mouse button down until you have reached the desired location for the lower right hand corner. This creates a while loop.

35. Select the stop button terminal and move it inside of the loop, as shown in Figure 2.
The small square with the circular arrow inside is the “continue while” feature. We want this loop to continue until we have pushed the STOP button.

36. Select Functions Palette->Boolean->Not and place it between the STOP button terminal and the “continue while” feature.

37. Wire these connections together as shown in Figure 3.

![Figure 3](image1.png)

38. In order to gain some more room to work with, move all of the buttons inside of the while loop and place the arrow cursor at the upper left hand corner. Two angle brackets will appear. By left clicking and holding, you can now drag the while loop out to a desired size as shown in Figure 4.

![Figure 4](image2.png)

39. Select Functions Palette->Structures->For Loop and place it inside of the while loop in the same manner that the while loop was made.
40. Wire the Number of Loops to Complete to the “N” located in the upper left hand corner. Leave the control outside of the For Loop.

41. Move the Current Iterations terminal inside of the For Loop and wire the “i” to it as shown:

42. Right-click on the left edge of the For Loop and select Add Shift Register.

Note: The Shift Register is used to transfer values between different iterations of a loop. In this case we are going to generate a value, add one to it, and pass it on to the next iteration of the loop.

43. Select Functions Palette->Numeric->Increment and place the object in between the two shift register arrows.

44. Wire the left (down) arrow to the input of the Increment function, and the right (up) arrow to the output of the Increment function (in LabVIEW the input is usually on the left and the output is usually on the right of any object).
Note: a broken line will appear when only one side is connected. The line will become solid when the other side as been connected.

45. Right click on the left hand shift register arrow and select Create Constant (this initializes the shift register). Then move the Current Loop Value terminal inside of the For Loop and connect it as shown in Figure 7.

![Figure 7.](image)

46. Finally select Functions Palette->Time and Dialog->Wait Until Next ms Multiple and place the object inside of the For Loop.

47. Right-click on the left side of the object and select Create Constant.

48. Using the hand tool enter a number of 100.

Note: right-clicking on the right hand side of the object and selecting Create Constant creates the constant, but doesn’t wire it to the input.

This part of the code has the effect of slowing the program down so that the loop is running 10 times per second instead of at full processor speed (we don’t want to use more resources than we need). The final code segment is shown in Figure 8.
Figure 8.

49. Select the Front Panel and using the hand tool input a value (ie 10) into the Number of Loops to Complete control.

50. Run the program by pressing the toolbar arrow in the upper left hand corner of the screen.

51. When you are done, stop the program with the stop button you have added on the front panel.

We have now completed a simple single threaded program.

MULTITHREADING

To add the next thread we need to use a concept called False Data Dependence. General multithreading could be accomplished by merely adding a new While Loop structure. However, we run into a problem when we terminate the program. At what point is the program actually done? If we have two loops or more, and only one stop button, we have a potential ambiguity as to the control of the loops. To solve this issue we are going to create a sequence loop. This loop requires that all of the other loops that have been wired to it must terminate before the sequence loop can run. The only thing the sequence loop is going to do is reset the stop button for the next execution of the program.

Note: stop buttons have boolean values of 0 or 1 (false and true accordingly). If the button on the Front Panel appears to be out then the value is 0 (false). If the button appears to be in then the value is 1(true).

52. In the diagram panel select Functions Palette->Structures->Sequence Loop and place this loop next to the original For Loop as shown in Figure 9.
Figure 9.

53. Using the wiring tool, connect the wire in between the Not function and the “continue while” function to the edge of the sequence loop.

Data is only passed out of a block when the execution of that block has terminated. Additionally, a structure may only execute when all of the input data is ready. By wiring the diagram in this manner, we can assure that the sequence will only run once the loop has terminated. This relationship will not change if additional loops are added in the same manner as we will see.

Figure 10.

54. Place a second while loop next to the original while loop.

55. Inside of the while loop place a Wait Until Next ms Multiple with the constant value of 100 (same as before). The result is shown in Figure 11.
56. Right-click on the STOP button and select Create->Local Variable.

57. Move this new object to the second while loop.

58. Repeat this procedure and create a second local variable. Place this local variable in the sequence structure. The result is shown in Figure 12.

59. In the second while loop select Function Palette->Boolean->Not and place it between the STOP local variable and the “continue function” as before.

60. Right-click on the STOP local variable and select Change to Read Local. Wire this function in as before. The results are shown in Figure 13.
61. Right-click on the STOP local variable inside of the sequence loop and select create constant. Tab to the hand tool and left click on the “F”. The results are shown in Figure 14.

62. Move to the front panel and select the Controls Palette->Numeric->Tank and place it on the front panel.

63. Select Control Palette->Numeric->Dial and place it as well on the front panel. The results are shown in Figure 15.
Figure 15.

The limits on the indicators may be changed by single, double or triple clicking on the number with the lettering tool. Additionally, many different attributes may be changed by right clicking on the indicator. You should experiment with these options.

The size of the indicator can be changed by moving the arrow tool to the corner of the indicator. When the small triple angle bracket appears, left click and drag the bracket to enlarge the object.

64. In the diagram panel move both indicators inside of the second while loop and wire them together.

65. Connect the wire in between the Not symbol and the “continue function” to the edge of the sequence loop as shown in Figure 16.

Figure 16.

The program may now be run from the Front Panel by entering the number of iterations that you would like for the “for loop” and pressing the start button (arrow) in the upper left corner of the screen. While the program is running you may move the dial by placing the hand over the dial,
left clicking (hold the button down) and rotating the dial to the desired position. To end the program, simply press the STOP button.

The stop sign next to the toolbar start arrow in the upper left hand corner halts the program immediately. However, if we want our program to accomplish certain tasks during shutdown, we must implement our own shutdown procedure. Our stop button finishes counting before ending the program. The stop sign stops the program at its current location and will not complete the shutdown procedure.

This is a simple program that allows us to have two threads running at once in our program. The application as stands would not require us to have multiple threads. However, as we will see in the next lab, when we include “pop up” screens, we would stop execution in our main program until the “pop up” screen has been closed. For most purposes in data acquisition this would not be desirable.

**Answer Questions 3 – 7.**

**DEBUGGING**

One of the benefits of having a visual language environment like LabVIEW is that the progress of information through the code can easily be seen. LabVIEW has several debugging methods. The only one we will mention at this time is the data highlighting feature.

66. Press the light bulb on the taskbar (in the Diagram Window).

67. Start the program.

Notice in the diagram window that there are small dots progressing through the code. Using this feature you can see if and how a section of code is running.

Note save your file at this time to be used in the next lab.

**RANDOM NOISE PROGRAM**

LabVIEW is commonly used to control other programs as well as test equipment. In this program we will generate white noise, view and listen to the results through the MATLAB interface. If you look at the Diagram Panel you will see a large block containing the MATLAB code. This is but one type of interface supported by LabVIEW. Additionally LabVIEW supports ActiveX, DDE, HiQ, SQL, Data Sockets, and much more. As such, control of such programs as Excel, Word, and Access is easily accomplished.

68. Open the program titled “Filter.vi”.

On the Front Panel you will see four buttons (one for each filter type: lowpass, highpass, bandwidth and no filter). Next to each button is a slider bar. The limits on the slider bar indicate the percent of sample transmitted.

69. Run the program.
70. Select the appropriate filter type.

71. Click on the generate Generate Graph button.

At this point you will see the a MATLAB interface opens and the data is displayed on the graphs. The top graph shows the unfiltered sample, the middle graph shows the filtered sample and the bottom graph shows the unfiltered minus the filtered sample. Additionally, you will hear the sample through the speakers. Allow the sound to terminate before continuing.

72. Close the MATLAB graph window.

Note: LabVIEW will not update the window. It must be closed and a new window must be generated to display new information.

73. Select a new filter type and repeat the above instructions.

74. Push the End Program button when you are finished.

**EXITING LABVIEW**

You can exit LabVIEW the same as any Windows based program. Simply select File->Exit, CTRL-Q, or press the small “X” in the upper right hand corner. When you exit LabVIEW you will be prompted as to whether or not you wish to save any unsaved modified programs. If you make any modifications to a program, an asterisk will appear immediately following the name of the program.

Note: If you have extra time, take a look at the program called robot in the LabVIEW-Examples->Picture directory. This is a good example of what LabVIEW is capable of doing. Remember to access the diagram panel. A white square is a called a subVI (sub Virtual Instrument), and may be accessed by double clicking on it. In this way you can see many layers of code.