Analog and Digital Communications

2nd Year Electronic Laboratory

Imperial College London

Version 2.00

Prepared by:

S. Somuyiwa – D. Kurka – T. Tze-Yang – S. Sreekumar - M. Mohammadi Amiri – J. Puyol-Roig

C. Leung – D. Gunduz

Lab 1: Introduction to LabView

The purpose of this lab is to introduce you to the LabView Communications System Design Suite's environment. In this first lab, you will learn how to implement mathematical functions and create basic communications modules.

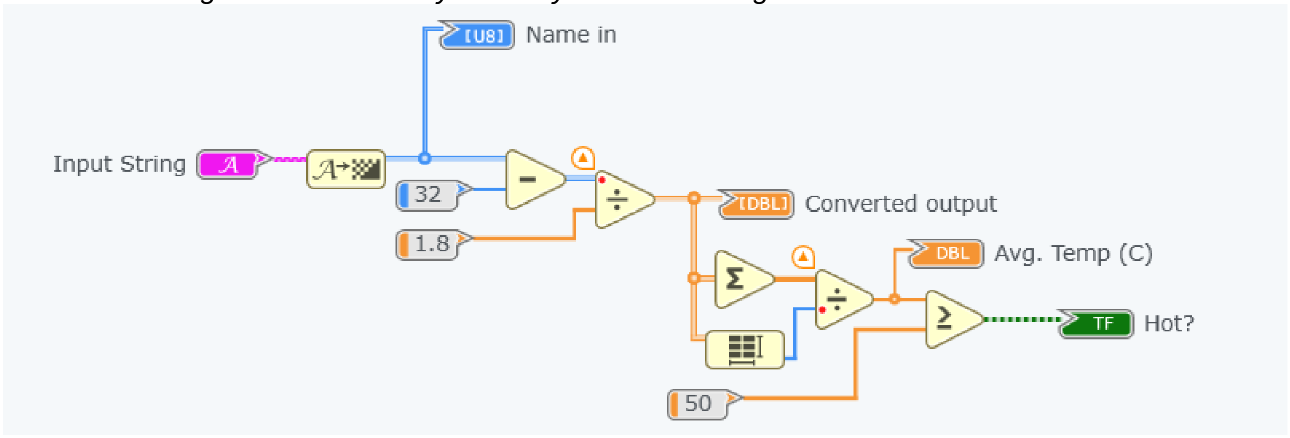
Exercise 1: Data Types in LabView

In this exercise you will examine the data types in LabView, and do some basic conversions between them. The purpose of the code will be to convert each letter of a string into its ASCII binary representation, which will be shown in their equivalent unsigned integer form. Then by treating each integer as a Fahrenheit temperature, you will convert it into Celsius. Finally, an average of the Celsius values will be taken and compared to a threshold value to indicate if the string you inputted is considered "Hot".

Instructions:

1. From the "**File**" menu, create a new "**Blank Project**", and name it **Lab1**.
2. On the "**Files**" pane, add a new VI, and name it **DataTypes.gvi**.
3. Now you will add the module through which you can input strings. This can be done from the "**function palette**" under the "**Diagram tab**". Using the search box, search for the module **String to byte array** and create a control from the input by right clicking on the terminal "string" and selecting "create control". This will allow you to input a string from the Panel tab. You can also learn more about the modules by clicking on the module, and on the right panel click on "**Context Help**".
4. Create an indicator at the output of the **String to byte array** module by right clicking on the "unsigned byte array" terminal and choosing "Create indicator". This will allow you to view the byte array on the Panel tab in the form of unsigned integers representing the ASCII values of each character of the string you input.
5. Assuming that each ASCII value represents a temperature in Fahrenheit, convert the array of integers into Celsius values. Arithmetic operations can be applied to whole arrays by connecting the array and a scalar constant to an operator node. **Hint:** *You can right click on the terminals of a module to create constants, and if it gives you an array constant you can switch to a scalar constant by right clicking on the constant itself.*
6. Create an indicator for the converted array.
7. Calculate the average of the converted values using the modules **Add Array Elements** and **Array Size**. Create an indicator showing the average value.
8. Apply a threshold by using the module **Greater or Equal**, and compare the average value to a constant. Set the constant to 50 and create an indicator called "**Hot?**" at the output of the **Greater or Equal** module.
9. Test the code by going to the Panel tab and placing all the controls and indicators. Input your name into the string control and view the result by pressing the play button.

Hint: *You can change the name of any block by double-clicking on the name.*



Task:

☞ Input your surname on “input string” field and retrieve the integer representation of its first three characters within a single execution of the program. Copy (take a screenshot) the block diagram and the front panel of your code, and include them in your logbook.

Exercise 2: Implementation of the Central Limit Theorem

In this exercise you will visualize the central limit theorem by using loop structures, graphical blocks and arrays. Independent uniformly distributed random numbers will be generated, and the convergence of the sample mean value to a normal distribution will be observed as the number of trials increases. You should create and save a new VI called “**CLT.gvi**” under your project Lab1.

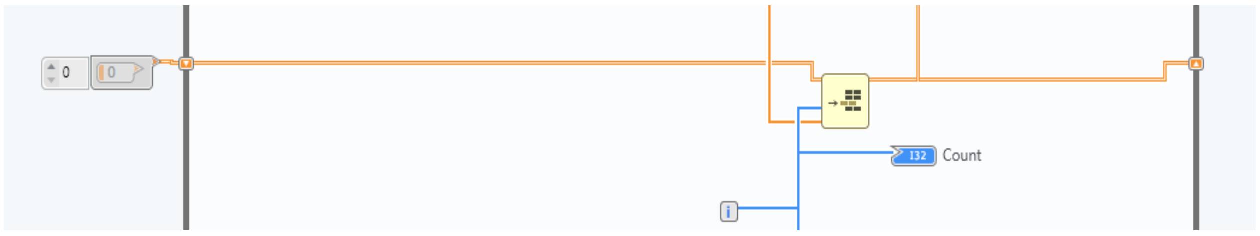
In a nutshell, we will generate a vector of independent and identically distributed random numbers (using a for loop), evaluate their sample mean, and iterate this within a while loop. After each iteration, we will update the histogram of the mean values to observe the convergence of the distribution.

Instructions:

1. In order to observe the random experiment continuously, place a **While Loop** (in Program Flow) to the block diagram. The i symbol on the lower left corner of the loop contains the value of the current iteration of the loop. You will also see a conditional terminal (the red button) created on the lower right corner of the while loop, which allows you to terminate the loop if a certain condition is met. To enter the desired stopping condition, right-click on the input terminal of the conditional terminal, and choose “Create control”. A stop button will also be available on the front panel.
2. Inside the loop, place a timing module called **Wait**, which will control the breaks between each execution of the while loop. To determine the duration of breaks, create a control for this node. Remember to place the corresponding control in the front panel.
3. Place a **For Loop** inside the while loop. As an equivalent of the nested loops in textual programming languages, place a second **For Loop** within the last one which will allow you to create a 2D array.
4. Adjust the number of iterations of both **For Loops** by placing constants connected to the left-hand side loop count terminal (N symbol). This will allow you to define the dimensions of the 2D array you will create.
5. Search for the **Add Array Elements** module**,** and place the function outside the nested **for** loops. This will allow you to sum the elements of the generated array, which is necessary for calculating the sample mean.
6. Inside the **For Loops**, place a **Random Number Generator,** and connect its output to the **Add Array Elements** function. Notice the boxes that are created on the frames of **For Loops**. These boxes are called **Tunnels,** and they specify how the data will be transferred through the frame. By default, these tunnels are indexing tunnels (right click on them and ensure auto-indexing is selected). An auto-indexing output tunnel appends the single value obtained from a single loop iteration to an accumulating array of data.
7. Now, in order to find the mean value, we need to divide the sum of the generated random values to the size of the random array. Find the **Divide** function in the block diagram and connect the output of **Add Array Elements** to the “X” terminal of it. Then right click on the “Y” terminal and create a constant. The constant should be equal to the total number of elements of the array (product of the number of rows and the number of the columns), so that the output of the division operation is the sample mean of your random array.
8. You need to create an array of these sample means in order to create a histogram. Start by placing **Insert into Array function** on the block diagram. This function inserts new elements into a specific position of an input array, and also has the ability to increase the size of the array. Connect the sample mean you calculated to the “New Element/Subarray" terminal.
9. Connect the i symbol of the while loop (loop iteration) to the “Index” terminal of **Insert into**

Array function. This will assign the new mean value generated to its corresponding position within an array storing all results.

10. Now you need a base array to add all these new elements to, and you need it to be available at each iteration. Here a new concept called "**Shift Registers**" will be introduced. Shift register is a type of tunnel that carries information from an iteration to the next one. First, connect the output array of **Insert into Array function** to the right frame of the while loop. Then right click on the tunnel and select "Replace with Shift Register". This will create another tunnel on the left edge of the loop. Connect this new tunnel to the "array" terminal of the **Insert into Array function**. The tunnel on the left edge carries the data from the previous iteration, whereas the one on the right edge sends the data to the next iteration. Finally, right click on the "input" terminal of the left register, and create a **constant array** so it can initialise the program with a 0 array.



11. Your loop is complete, and all you need is to create and observe the histogram. Search for the **Histogram function** and connect the output of the **Insert into Array function** to the "signal" terminal of the **Histogram Function**. Then create a control for "number of bins" terminal that will allow you to control the number of bins in the histogram.
12. Finally, to observe the changes in the histogram, create an indicator to the histogram terminal. You should create the indicator within the while loop. Add the corresponding indicator graph in the front panel
13. Switch to front panel, set "milliseconds to wait" to 10, "number of bins" to 100, and run the code.

Tasks:

☞ Include the block diagram and the front panel in your logbook.

☞ Create a control for stopping the execution of the program after 1000 iterations. Set the value of "intervals" to 10. Include the corresponding histogram in your logbook.

☞ Revise your code such that the final distribution you observe in the histogram is a standard Normal distribution (i.e., it has zero mean and unit variance). Include the revised block diagram in your logbook, and explain each of the changes you have made.

Exercise 3: Waveforms

In this exercise, you will learn how to implement signal generation, filters and waveforms in LabView. Please start with creating a new VI for this experiment.

Instructions:

1. First of all, you need to generate a signal from the **Wave Generator** module, which can be found by searching in the function palette. It generates a sine waveform by default.
2. For the adjustments of the generated signal, you should create the controls for "frequency", "amplitude", "sample rate" and "samples" terminals. Import two more "Sine Waveform" modules to generate three signals, and add these three signals together.
3. To observe the power spectrum of the combined waveform, search for the **FFT Power Spectrum and PSD** module. Insert it to the diagram and change its configuration to "Power Spectral Density".
4. Connect the combined waveform to the "signal" input of the module. To observe the output of the FFT function, a graph from the panel is needed. Create the graph first on the panel,

and then connect the created graph node in the block diagram to the PSD output terminal of the **FFT Power Spectrum and PSD** module.

Hint: *To zoom-in on the graph, click on the graph on the panel page, and on the configuration pane (right) select "Graph Tools" under the Parts section. Now you should see a toolbar underneath your graph, with which you can zoom in/out.*

5. To eliminate the higher frequency components of your signal, a low pass filter should be applied. Use the **Filter** module and select "Lowpass" and "Butterworth" in the panel on the right to configure it as a lowpass Butterworth filter. Connect the combined waveform to the signal input and place controls on the "order" and "cutoff frequency" terminals. Next, add a graph to the output of the filter to observe the filtered signal.
6. Assign the values in the table below to corresponding controls, and run the code.

Low cutoff frequency	10
Sampling Rate	1k
Number of Samples	1k
Order	5
Frequency_1	10
Frequency_2	30
Frequency_3	120
Amplitude	1

Tasks:

☞ Include the block diagram and the front panel in your logbook. The three output graphs, i.e., generated waveform, waveform PSD and filtered waveform should also be included.

☞ Modify the filter parameters such that you have only the second sine wave (one with frequency 30) at the output. Include these parameters in your logbook, and explain the reason why it works.