



Islamic University of Gaza
Faculty of Engineering
Electrical Engineering Department



Linear Control Laboratory Manual

This manual was developed by:

Dr. Basil Hamed – Supervisor
Eng. Nael Sofyan Dokhan – Author
Eng. Haya Ashraf Swedan – Author

June 2019

Experiments 1

Introduction to MATLAB

Objectives:

- To become familiar with the MATLAB software by introducing you the basic features, commands, and functions.

Introduction:

- The MATLAB Environment:

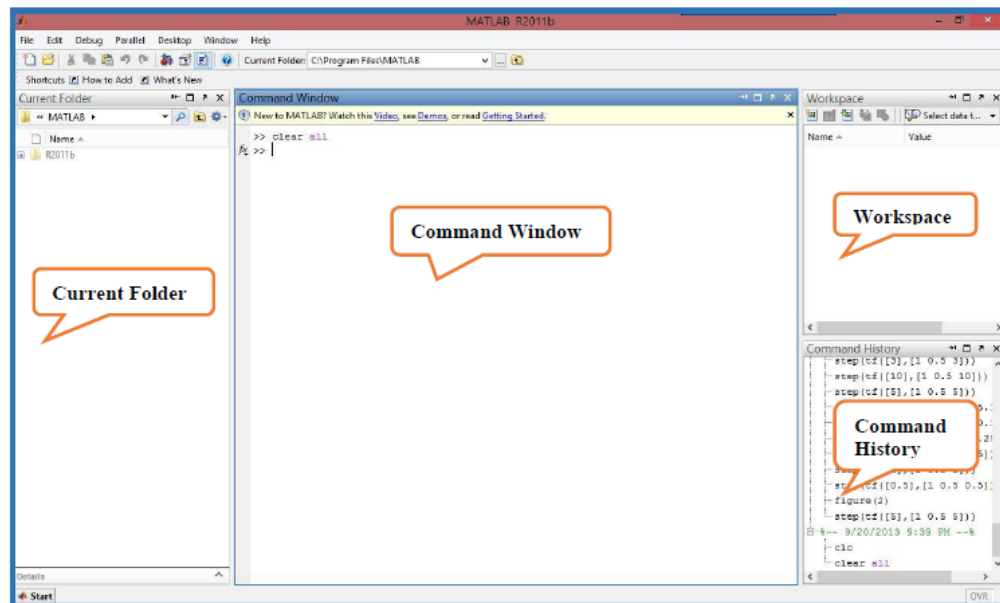


Figure 1 Matlab Environment

✍ Note: The most important function for learning MATLAB on your own:

»help sin

»doc sin

○ Basic Operations:

- Computations:

>> 2*3

ans = 6

>> 2+3

ans = 5

>> 2-3

ans = -1

>> 2/3

ans = 0.6667

- Suppress Display of Results:

>> 2*3;

% A semi-colon (;) after an expression suppresses the output.

- Commands:

CLC	Delete commands from the command window
Clear	Delete variables from the workspace
Who	Tells you what variables are currently defined
Whos	Tells you more about the variables which currently defined .

○ *Matrices:*

- All variables represent matrices.
- A scalar is a 1x1 matrix and a vector is a matrix with only one row or column.

Generating Vectors :

- 1- By directly inputting the elements,

Row Vector

```
>> a=[1, 2, 3]
```

a =

1 2 3

Column vector

```
>> a=[1; 2; 3]
```

a = 1

2

3

- 2- By using the colon (:) operator.

```
>> a=[1:3]
```

a =

1 2 3

```
>> c=[2: 2: 10]
```

c =

2 4 6 8 10

Generating Matrices :

```
X=[1 2 3; 4 5 6]
```

X =

1 2 3

4 5 6

1- For finding individual elements of a matrix.

» X(3,2)

ans =

8

2- For Accessing Sub matrices:

» X(1:3,3)

ans =

3

6

9

3- For Deleting Rows and Columns:

» X(:,2) = [] %To delete the second column of X

» X(1,:) = [] %To delete the first row of X

Matrix operation:

+	Addition
-	Subtraction
*	Multiplication
^	Power
'	Transpose (real) or conjugate transpose (complex)
,	Transpose (real or complex)
\	Left division
/	Right division

If A is an invertible square matrix and b is a compatible column or row vector, then :

☐ $x = A \backslash b$ is the solution of $A * x = b$

☐ $x = b/A$ is the solution of $x * A = b$

If you have a matrix $B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, What is the differences between $B*B$ and $B.*B$??

Matrix Building Functions:

EYE	Identity matrix
ZEROS	Matrix of zeros
ONES	Matrix of ones
TRIU	Upper triangular part of a matrix
TRIL	Lower triangular part of a matrix
RAND	Matrix with random elements

» zeros(2,3)

ans =

```
0 0 0
0.1465    0.16354
```

```
0 0 0
0.5642    0.14654
```

0.65847

» A=rand(3)

A=

0.4684 0.1465 0.16354

0.9985 0.5642 0.14654

0.1254 0.1454 0.65847

» triu(A)

ans=

0.4684

0

0 0

○ **Complex number:**

- The symbol "i" identifies the imaginary part and has to be typed immediately after the numerical value of the imaginary part: (you can use j instead of i.)
- It is also important to point out that termination with the character i only works with simple numbers, not expressions.
- If you want to multiply a complex expression by i, you have to use the multiplication operation symbol (*).(Like this: (1 - 2i)*i)

Some complex building function

COMPLEX(X,Y)	Return a complex number x+yi
REAL(X)	Real part of a complex number
IMAG(X)	Imaginary part of a complex number
ABS(X)	Magnitude of the complex number
ANGLE(X)	Angle of a complex number x
CONJ(X)	Complex conjugate of the complex number x

- **Control Flow Statements:**

The relational operators in MATLAB are:

<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal
==	Equal
~=	Not equal
&	And
	OR
~	Not

1- Variable Controlled Loops (for):

```
for variable = first: inc: last
    statements
end
```

2- Relational Controlled Loops (while):

```
while relation
    Statements
End
```

3- Branching (if):

```
if relation
    true alternative
else
    false alternative
end
```

4- Switch

switch variable

case value 1

statement group 1

case value 2

statement group 2

...

otherwise

last statement group

end

✍ Note:

- A 'while' loop is useful when, in contrast to a 'for' loop, the calculations are to be repeated an undetermined number of times.
- The 'break' command causes the enclosed loop – either a 'for' or 'while' loop to terminate.
- The 'return' command causes the currently executing function M-file to terminate.

○ Plotting:

1- Basic Two-Dimensional Plot:

```
t=0:0.01:2;
```

```
Temp=exp(-t);
```

```
plot(t,Temp)
```

```
xlabel('Time')
```

```
ylabel('Temp')
```

```
title('Transient Temperatures')
```

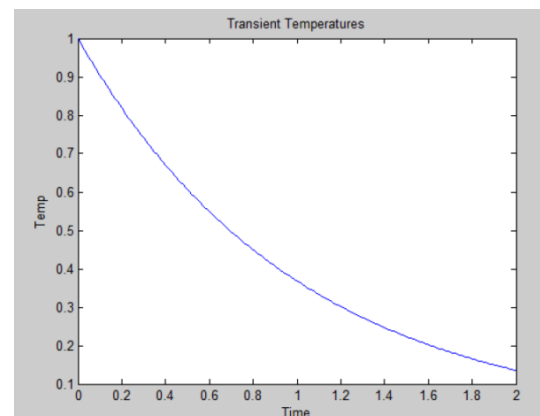


Figure 2 Two-Dimensional Plot

2- Multiple Plots on a Single Graph:

```

x=0:pi/16:2*pi;
y1=sin(x);
y2=cos(x);
plot(x,y1,'* -',x,y2,'r s -')
xlabel('x')
ylabel('sin(x), cos(x)')
title('Trig Functions')
legend('sin','cos')

```

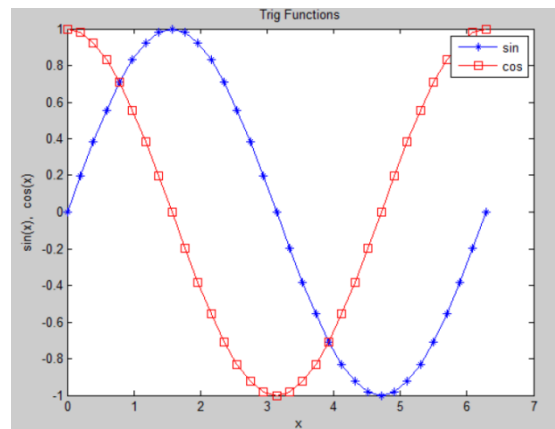


Figure 3 Multiple Plots on a Single Graph

3- Create graphics arrays:

```

x=0:pi/16:2*pi;
y1=sin(x);
y2=cos(x);
subplot(2,1,1)
plot(x,y1,'* -')
xlabel('x')
ylabel('sin(x)')
subplot(2,1,2)
plot(x,y2,'r s -')
xlabel('x')
ylabel('cos(x)')

```

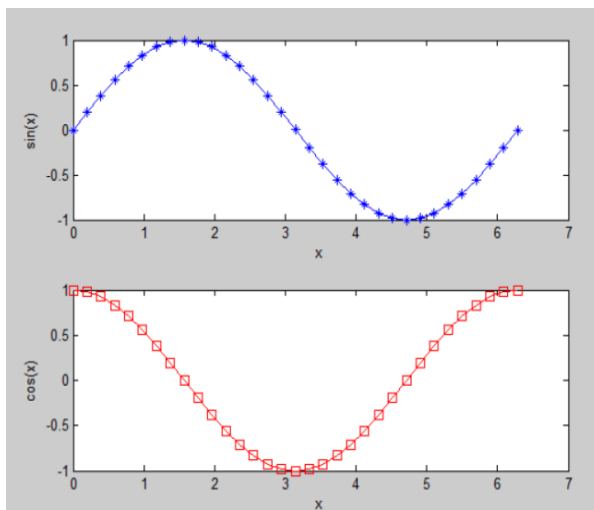


Figure 4 Graphics Arrays

- ❑ You can change colors, symbols, markers and line types as demonstrated below:

SYMBOL	COLOR	SYMBOL	MARKER	SYMBOL	LINE STYLE
B	Blue	.	Point	-	Solid line
G	Green	O	Circle	:	Dotted line
R	Red	X	X-mark	-.	Dash-dot line
C	Cyan	+	Plus	--	Dashed line
M	Magenta	*	Star		
Y	Yellow	S	Square		
K	Black	D	Diamond		
W	White	v	Triangle down		
		^	Triangle up		
		<	Triangle left		
		>	Triangle right		
		P	Pentagram		
		H	hexagram		

○ MATLAB Functions:

1- Scalar Functions:

FUNCTIONS	DESCRIPTION
ABS	Abs (x) is the absolute value of the elements of x.
SIGN	Sign (x) returns 1 if the element x is greater than zero, 0 if it equals zero and -1 if it is less than zero.
ROUND	Round (x) rounds the elements of x to the nearest integers.
SIN	Sin (x) is the sine of the elements of x, where x in radian.
ASIN	asin Inverse sine.
EXP	exp(x) is the exponential of the elements of x.
LOG	log(x) is the natural logarithm of the elements of x.
LOG10	log10(x) is the base 10 logarithm of the elements of x.

- ✓ Try to see what are doing these functions:

(floor, ceil, rem, sqrt)

- ✓ Try to find out its function:

hold, grid, axis, ezplot, gtext, ginput

2- Vector Functions:

FUNCTIONS	DESCRIPTION
MAX	For vectors, max(x) is the largest element in x. for matrices, max(x) is a row vector containing the maximum element from each column.
MIN	For vectors, min(x) is the smallest element in x. for matrices, min(x) is a row vector containing the minimum element from each column.
SORT	Sort in increasing or decreasing order.
SUM	Find the sum of elements.
PROD	prod (x) is the product of the elements of x.
MEAN	For vectors, mean(x) is the mean value of the elements in x. for matrices, mean(x) is a row vector containing the mean value of each column.
MEDIAN	for vectors, median(a) is the median value of the elements in a, for matrices, median(a) is a row vector containing the median value of each column
ANY	True if any element of a vector is a nonzero number or is logical 1 (true).
ALL	True if all elements of a vector are nonzero.
FIND	Find the index of nonzero elements.
Linspace	linspace (x1, x2, n) generates n points between x1 and x2.

Certain MATLAB functions operate essentially on scalars, but operate element wise when applied to a matrix. Some of these functions are:

FUNCTIONS	DESCRIPTION
INV	Inv (X) is the inverse of the square matrix X.
SIZE	Size of array.
DET	Determinant.
POLY	Convert roots to polynomial.
RANK	Matrix rank.
EIG	E = eig(X) is a vector containing the eigenvalues of a square matrix X.
POLYVAL	Y = polyval(P,X) returns the value of a polynomial P evaluated at X.
POLYVALM	Evaluate polynomial with matrix argument.

○ M-Files:

- Such files are called M-files because they must have the file type '.m'.
- There are two types of M-files:
 - 1- Script files.
 - 2- Function files.
- The name of an M-file must begin with an alphabetic character.
- Script files are simply pieces of code that are organized inside a file (i.e. they could be run line-by-line inside the MATLAB workspace) so they can add to workspace variables.
- Function files cannot be run separately but they have to be called inside other scripts/functions and their variables are local.
- You must save the m-file as the name of the function to be able to fetch it at write it in command window.

○ Linear algebra in MATLAB:

1- Solving Equation: In matrix notation, this problem can be stated as follows:

We may need to find X_1 , X_2 , and X_3 so that :

$$\begin{aligned} X_1 + 3X_2 + X_3 &= 1 \\ -X_1 - 3X_2 + 10X_3 &= 0 \\ 3X_1 + 2X_2 + X_3 &= -1 \end{aligned}$$

The problem can be rewritten in matrix-vector notation. We introduce a matrix A and a vector b by :

$$A = \begin{bmatrix} 1 & 3 & 1 \\ -1 & -3 & 10 \\ 3 & 2 & 1 \end{bmatrix} ; \quad B = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$$

Now we want to find the solution vector $X = [X_1 \ X_2 \ X_3]$ so that $AX = b$

```
>> a=[1 3 1;-1 -3 10;3 2 1];
>> b=[1;0;-1];
>> c=inv(a);
>> v=b*c
>> v=c*b

v =

    -0.7273
     0.5455
     0.0909
```

2- Laplace Transformation:

- MATLAB has a function called (Laplace) which transfer a function from time domain to S-Domain.
- Before you use this function you must declare the variable by syms function.
- To get the Laplace inverse, only use (ilaplace) after declare the variable.

```
>> syms t
>> ilaplace(1/(t-1))
ans = exp(x)

>> syms x
>> laplace(exp(x))
ans = 1/(s - 1)
>> laplace(exp(2*x))
ans = 1/(s - 2)
```

○ Simulink:

- Simulink is an environment for simulation and model-based design for dynamic and embedded systems.
- Simulink is integrated with MATLAB, providing immediate access to an extensive range of tools that let you develop algorithms, and provides a graphical user interface for constructing block diagram models using “drag-and-drop” operations.
- You start Simulink by open MATLAB and select the Simulink icon in the Toolbar as shown in the figure or type “Simulink” in the Command window

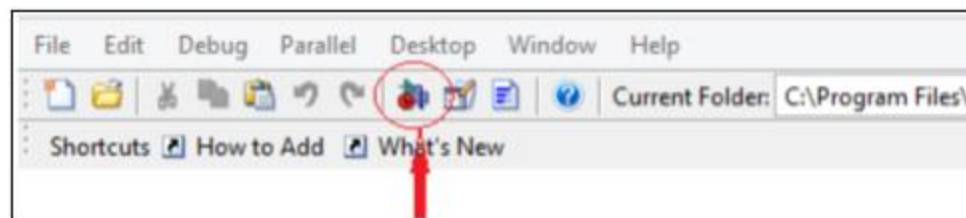


Figure 5 Simulink

- Simulink Library:

The Simulink Library Browser is the library where you find all the blocks you may use in Simulink. Simulink software includes an extensive library of functions commonly used in modeling a system. These include:

- 1- Continuous and discrete dynamics blocks, such as Integration, Transfer functions, Transport Delay, etc.
- 2- Math blocks, such as Sum, Product, Add, etc
- 3- Sources, such as Ramp, Random Generator, Step, etc

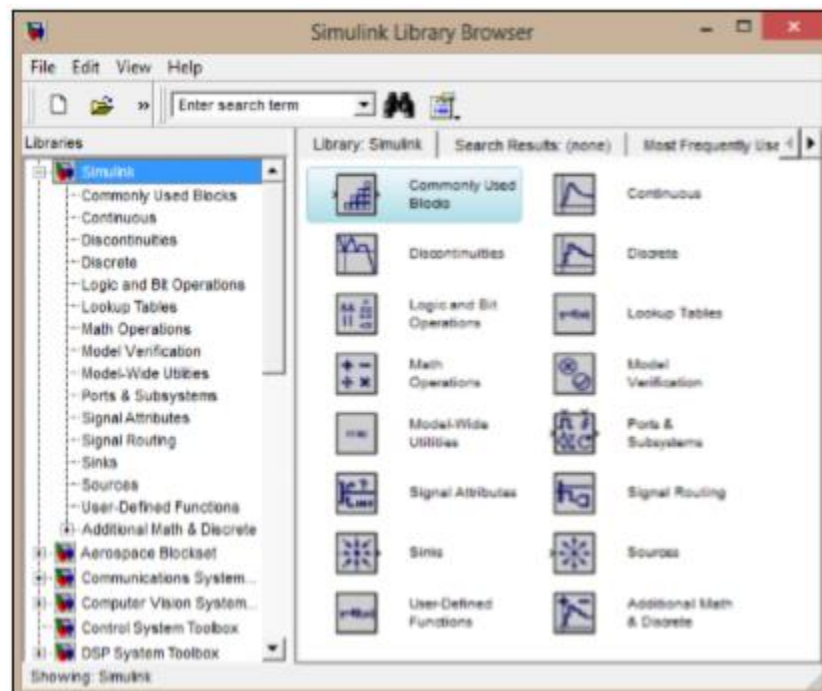


Figure 6 Simulink Library

Experiments 2

Introduction to LabVIEW

Objectives:

- To study introduction to labview program.
- To apply some basic examples on labview.

Introduction:

LabVIEW programs are called virtual instruments, or VIs, because their appearance and operation imitate physical instruments, such as oscilloscopes and millimeters. LabVIEW contains a comprehensive set of tools for acquiring, analyzing, displaying, and storing data, as well as tools to help you troubleshoot code you write.

LabVIEW is a graphical programming language that uses icons instead of lines of text to create applications. In contrast to text-based programming languages, where instructions determine program execution, LabVIEW uses dataflow programming, where the flow of data determines execution.

In LabVIEW, you build a user interface, or front panel, with controls and indicators. Controls are knobs, push buttons, dials, and other input mechanisms. Indicators are graphs, LEDs, and other output displays. After you build the front panel, you add code using VIs and structures to control the front panel objects. The block diagram contains this code.

○ Opening a Blank VI:

If no template is available for the VI you want to build, you can start with a blank VI and add Express VIs to accomplish a specific task.

Complete the following steps to open a blank VI.

1. Launch LabVIEW program.
2. Select Blank VI to create new VI program from the list display on started window.
3. A blank front panel window and block diagram window appear.
4. If the Functions palette is not visible, right-click any blank space on the block diagram to display a temporary version of the Functions palette. Click the thumbtack, shown below, in the upper left corner of the Functions palette to pin the palette so it is no longer temporary.



Figure 1: Thumbtack

A VI contains three main parts:

- **Front panel:** How the user interacts with the VI.
- **Block diagram:** The code that controls the program.
- **Icon and connector pane:** Means of connecting a VI to other VIs.

In LabVIEW, you build a user interface by using a set of tools and objects. The user interface is known as the front panel. You then add code using graphical representations of functions to control the front panel objects. The block diagram contains this code. In some ways, the block diagram resembles a flowchart. Users interact with the Front Panel when the program is running. Users can control the program, change inputs, and see data updated in real time. Controls are used for inputs such as, adjusting a slide control to set an alarm value, turning a switch on or off, or to stop a program. Indicators are used as outputs. Thermometers, lights, and other indicators display output values from the program. These may include data, program states, and other information.

Every front panel control or indicator has a corresponding terminal on the block diagram. When a VI is run, values from controls flow through the block diagram, where they are used in the functions on the diagram, and the results are passed into other functions or indicators through wires.

1- Front Panel:

The front panel is the user interface of a VI. You build the front panel by using controls and indicators, which are the interactive input and output terminals of the VI, respectively. Controls and indicators are located on the Controls palette.

Controls are knobs, push buttons, dials, and other input mechanisms. Controls simulate instrument input mechanisms and supply data to the block diagram of the VI.

Indicators are graphs, LEDs, and other displays. Indicators simulate instrument output mechanisms and display data the block diagram acquires or generates.

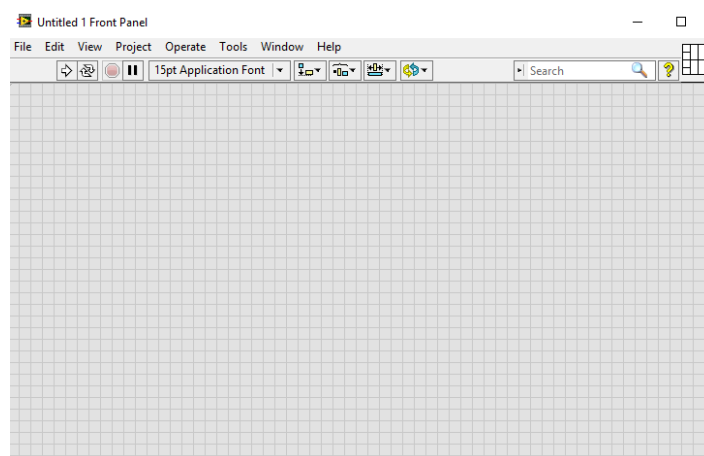


Figure 2: Front panel

2- Block Diagram:

The block diagram contains the graphical source code, also known as G code or block diagram code, for how the VI runs.

The block diagram code uses graphical representations of functions to control the front panel objects. Front panel objects appear as icon terminals on the block diagram. Wires connect control and indicator terminals to Express VIs, VIs, and functions. Data flows through the wires in the following ways: from controls to VIs and functions, from VIs and functions to indicators, and from VIs and functions to other VIs and functions. The movement of data through the nodes on the block diagram determines the execution order of the VIs and functions. This movement of data is known as dataflow programming.

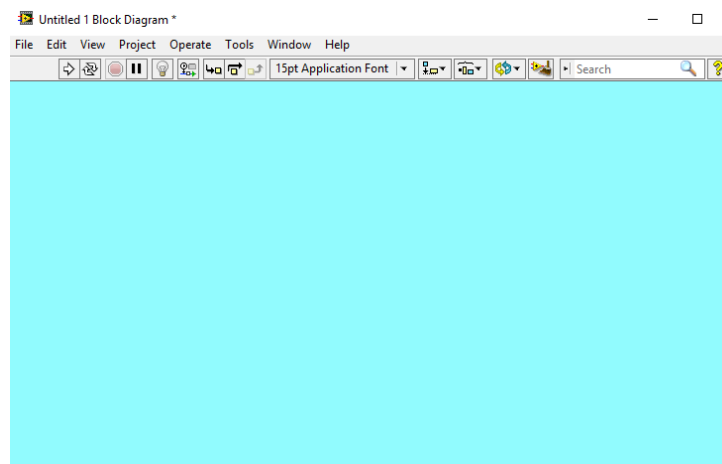


Figure 3: Block diagram

3- Icon and connector pane:

- A connector pane defines VI inputs (controls) and outputs (indicators).
- One can change the number of inputs and outputs by using different connector pane patterns.
- A VI icon is shown at the top right corner of the front panel.

Table 1: Shortcuts

Shortcut	Function
<Ctrl-R>	Runs a VI.
<Ctrl-Z>	Undoes the last action.
<Ctrl-E>	Switches between the block diagram and the front panel window.
<Ctrl-S>	Saves a VI.

○ Graphical Environment:

The LabVIEW palettes give you the options you need to create and edit the front panel and block diagram.

1- Tools Palette:

The Tools palette is available on the front panel and the block diagram.

A tool is a special operating mode of the mouse cursor, when you select a tool, the cursor icon changes to the tool icon, we use the tools to operate and modify front panel and block diagram objects.

Select view → Tools Palette to display the Tools palette. You can place the Tools palette anywhere on the screen. If automatic tool selection is enabled and you move the cursor over objects on the front panel or block diagram, LabVIEW automatically selects the corresponding tool from the Tools palette.



Automatic Selection Tool



Operating Tool



Scrolling Tool



Positioning/Resizing Tool



Breakpoint Tool



Labeling Tool



Probe Tool



Wiring Tool



Color Copy Tool



Shortcut Menu Tool



Coloring Tool

2- Control Palette:

The Controls palette is available only on the front panel, it contains the controls and indicators you use to create the front panel.

Select view → Controls Palette or right-click the front panel workspace to display the Controls palette, you can place it at anywhere on the screen.

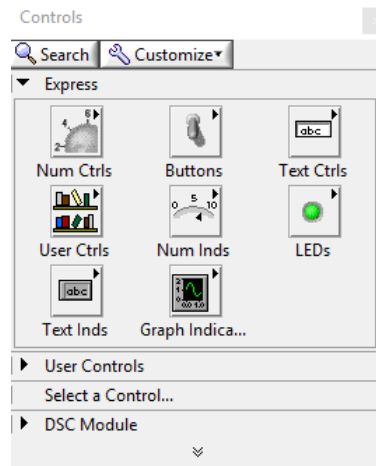


Figure 4: Control Palette

3- Function Palette:

The Functions palette is available only on the block diagram, it contains the VIs and functions you use to build the block diagram.

Select view → Functions Palette or right-click the block diagram workspace to display the Functions palette. You can place it at anywhere on the screen.

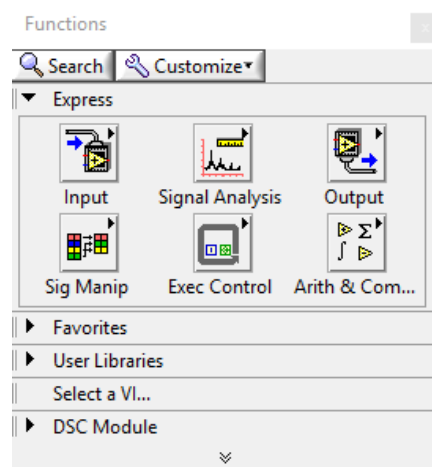


Figure5: Function Platte

Lab work:

Example1: Make a LabVIEW program that ask the user to enter(x,y) two numbers and the program find :

- ❖ $x+y$
- ❖ $x-y$
- ❖ $x*y$
- ❖ x/y if y equal zero (answer equal infinity) a led should light to warn the user.

Solution:

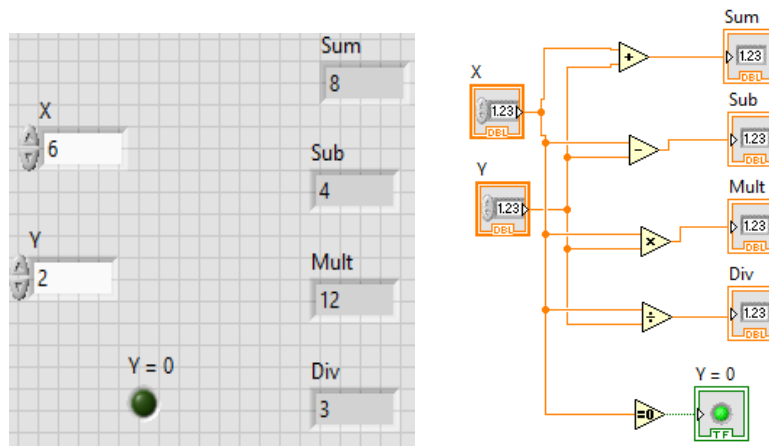


Figure 6: Front panel and Block diagram

Example2: Make a LabVIEW program to execute the following formula:

$$y = 5(x_1 * x_2)^2 + 4x_3 + \sin x_2$$

Solution:

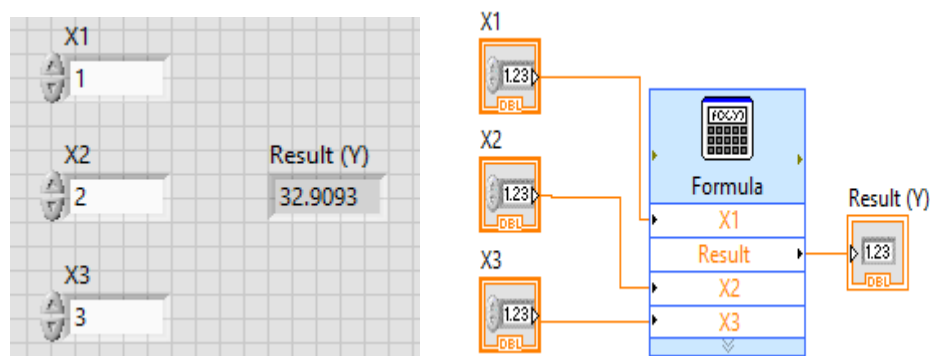


Figure 7: Front panel and Block diagram

 Note:

- ✓ Formula tool can be found at arithmetic and comparison and is used to enter a formula in an easy way for the user.
- ✓ If the formula is correct then this led is green.

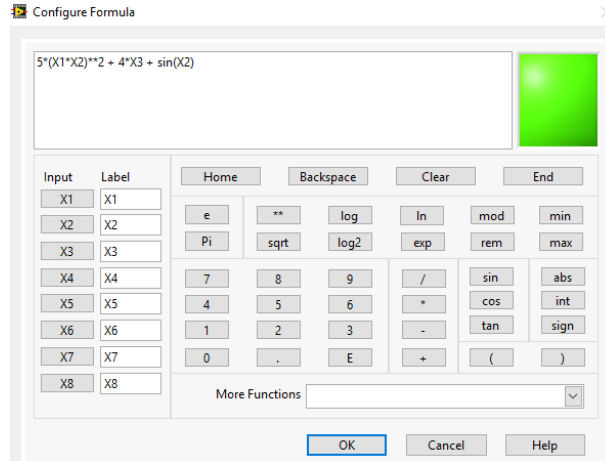


Figure 8: for writing Formula

Example3: Create a LabVIEW program that allows the user to enter the student marks in five subjects and then calculate the average. The program contain six leds as following.

Led 1	Average > 100 (Error)
Led 2	$90 \leq \text{Average} \leq 100$
Led 3	$80 \leq \text{Average} < 90$
Led 4	$70 \leq \text{Average} < 80$
Led 5	$60 \leq \text{Average} < 70$
Led 6	Average < 60 (Fail)

Solution:

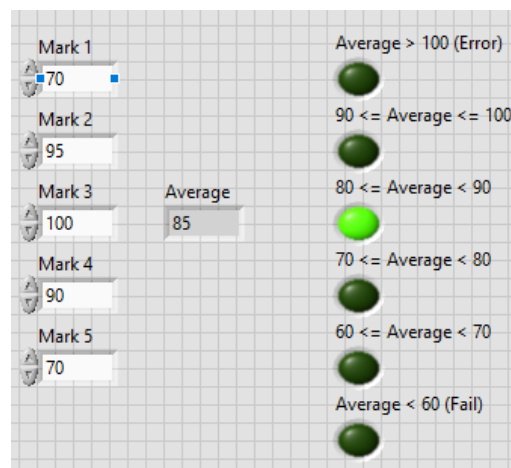


Figure 9: Front Panel

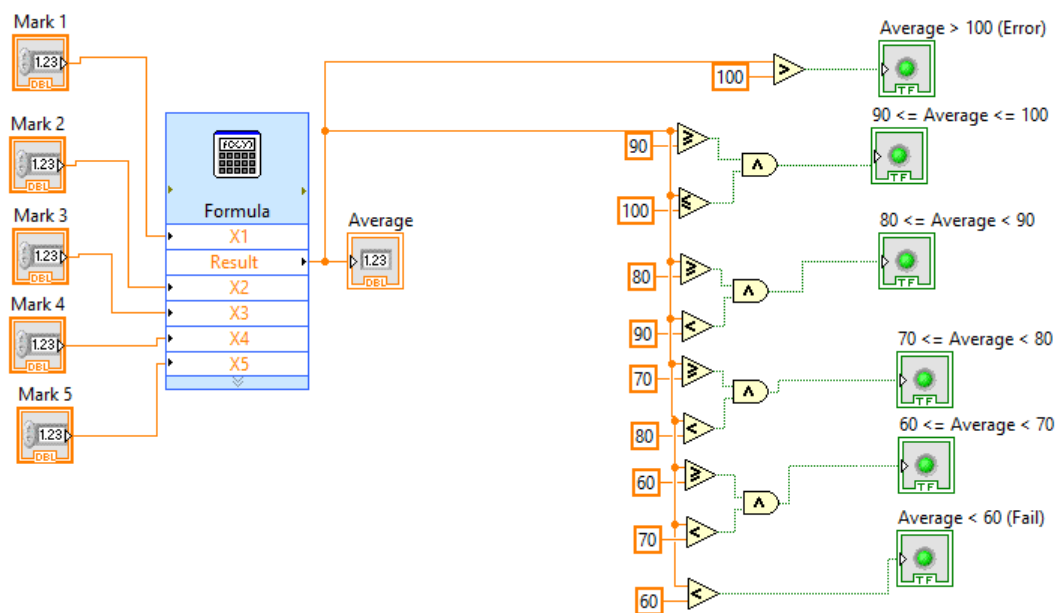


Figure 10: Block diagram

Example4: Make a LabVIEW program that plots a sine signal or a square according to a toggle switch control by the user. If the switch is on then the plotted signal is sine if the switch is off then the plotted signal is square.

Solution:

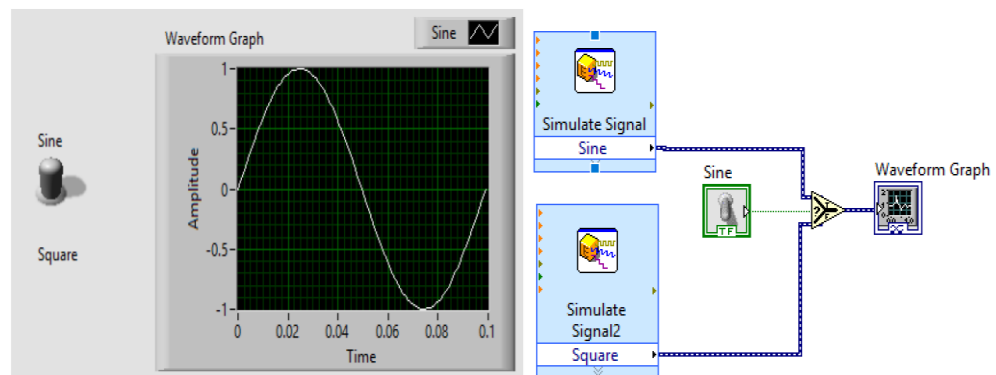


Figure 11: Front panel and Block diagram

Note:

- ✓ Simulated signal can be found on the function palette then input.
- ✓ The select operator can be found in arithmetic and comparison then comparison, it work as a multiplexer with a selector to choose the output.
- ✓ You can change Amplitude, Frequency and phase for any type of signal.

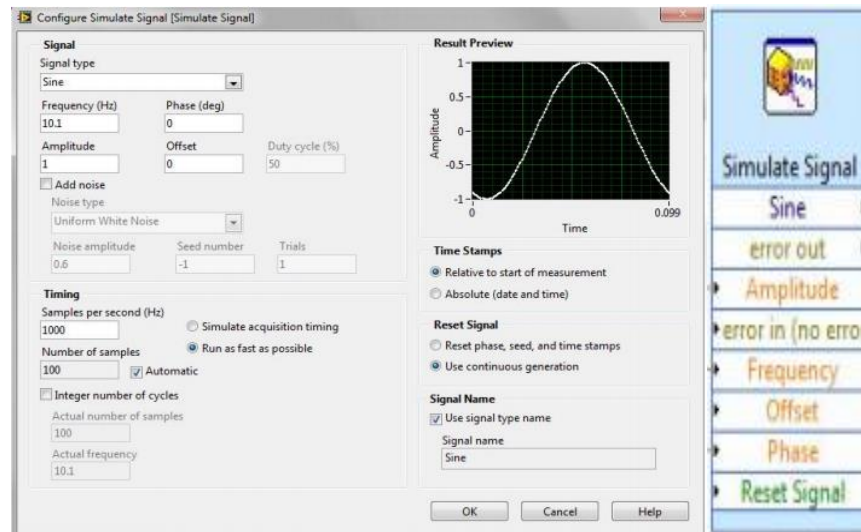


Figure 12: Changing parameter for signals

Example5: Modified previous example by adding noise to each signal and control the amplitude, frequency and amplitude of noise, the following must also be implemented:

- 1- When amplitude noise greater than amplitude signal the led will light on.
- 2- Display amplitude signal and amplitude noise by using gauge.

Solution:

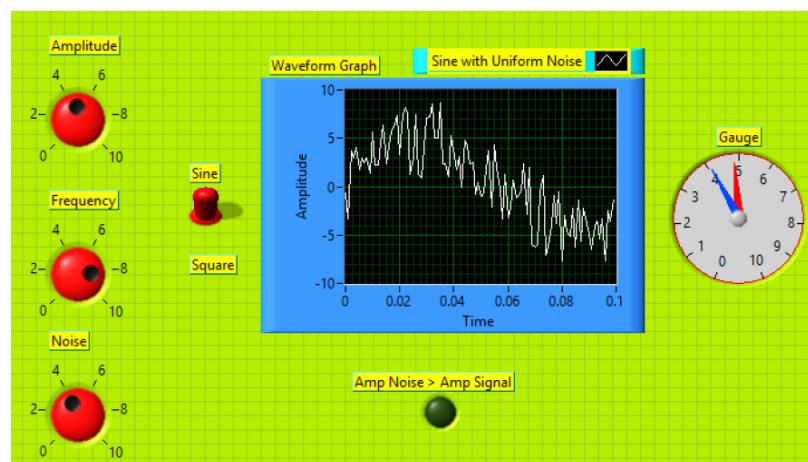


Figure 13: Front panel

Note:

- ✓ For controlling the parameter manually for any signal, you can use Knob. (Knob as input).
- ✓ Adding gauge by two needle to display amplitude signal and amplitude noise

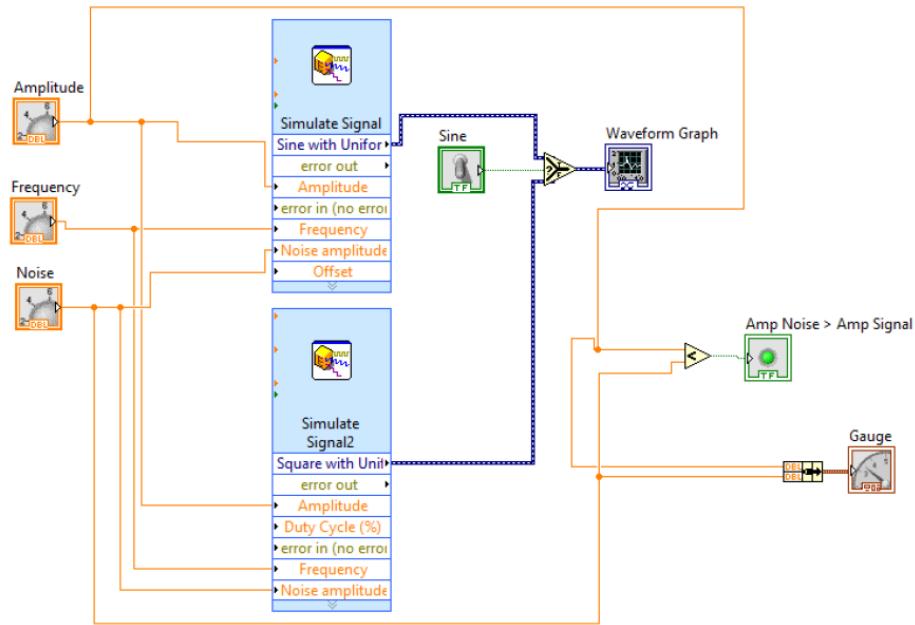


Figure 14: Block Diagram

Experiments 3

Execution Structures in LabVIEW

Objectives:

- To study how can we use While Loops, For Loops and Case Structures in LabVIEW and apply many examples for them.

Introduction:

In this lab we will explain the basic concepts of Loops and Structures in LabVIEW, Loops are used to run a specific part of a code iteratively. If you are a user of any programming language you must be familiar with loops. Some of the most commonly used loops in every programming language are **for loop**, **while loop** and **do....while loops**.

Loops are mostly used in the programs where we need the result of a function at various data points. We can update the value of variable, store values in arrays (we will discuss them in upcoming tutorials) and also plot graphs (we will discuss plotting in upcoming tutorials) and many more things iteratively. In programming language, a loop is defined as sequence of commands which are required to be repeated until a certain condition provided by the programmer is satisfied. After the condition is satisfied the control flow leaves the loop and stop executing the program over and over again.

Typically, for loop is used when the number of iterations are known and while loop allows the code to be executed repeatedly depending upon a Boolean condition. If the condition is true the code inside the loop is executed and if it is false the loop is skipped. While working with loops, one must remember that the condition given to the loop must converge to satisfaction, i.e. the if the condition is never going to be satisfied ever, your loop will run infinitely, and ultimately your program will never stop, which is not a good programming skill.

The topics are as follows:

- While Loop
- For Loop
- Case Structure

The different Loops and Structures available are located in the “Structures” sub palette in the Functions palette on the Block Diagram.

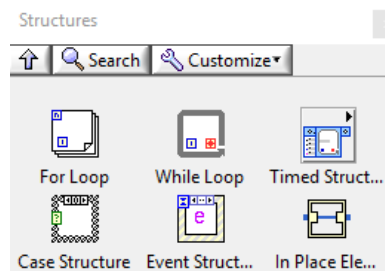


Figure 1: Structures loop

Firstly: While loop:

A While loop repeats the sub diagram inside it until the conditional terminal, an input terminal, receives a particular Boolean value. The Boolean value depends on the continuation behavior of the While Loop.

The While Loop is located on the Structures palette. Select the While Loop from the palette then use the cursor to drag a selection rectangle around the section of the program you want to repeat. When you release the mouse button, a While Loop boundary encloses the section you selected. Below we see an empty While loop:



Figure 2: While loop Structure

The While Loop executes the code it contains until the conditional terminal, which is an input terminal, receives a specific Boolean value. 

The iteration terminal is an output terminal that contains the number of completed iterations. 
The iteration count for the While Loop always starts at zero.

 Note:

The While Loop always executes at least once.

Example (1):

Create a LabVIEW program with three leds with the following specification:

- The first led lights red when the iteration is less than 15.
- The second led lights yellow when the iteration is more than 15 and less than 20.
- The third led lights green when the iteration is more than 20.
- The system should reset (iteration=0) when the iteration reaches 40.

Note: Control the loop delay so the led can go faster or slower

Solution:

1. First create the front panel design which consist of three leds as shown in figure below.
2. The main idea depends on the iteration value so we need to use the comparison function in order to meet the system specification, for example the first led should light when the iteration is less than or equals to 15 so we use the function as for the second led it should light when two conditions are true so we use comparison function in addition to an AND gate and final the same for the third led.
3. To reset the system at 40 we use the equal function as shown in figure below.
4. When a loop finishes executing an iteration, it immediately begins executing the next iteration, unless it reaches a stop condition. Most often, you need to control the iteration frequency or timing. For example, if you are acquiring data, and you want to acquire the data once every 10 seconds, you need a way to time the loop iterations so they occur once every 10 seconds. Even if you do not need the execution to occur at a certain frequency, you need to provide the processor with time to complete other tasks, such as responding to the user interface (Time Delay).

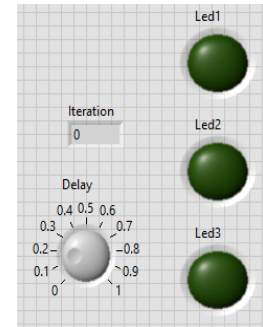


Figure 3 Front Panel

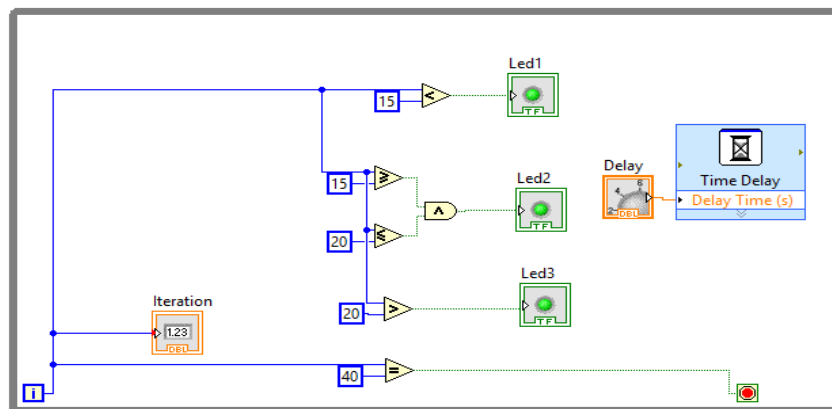


Figure 4 Block diagram

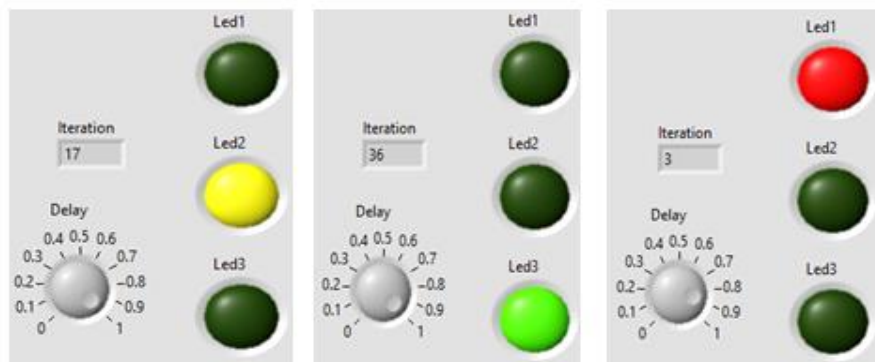
❖ After run program:

Figure 5 Front panel

Secondly: For loop

The for is used to perform a task according to a number defined by the user. The For Loop differs from the While Loop in that the For Loop executes a set number of times. A While Loop stops executing only if the value at the conditional terminal exists. The for loop has two terminals as in figure:

- 1-  The count terminal is an input terminal whose value indicates how many times to repeat the subdiagram
- 2-  The iteration terminal is an output terminal that contains the number of completed iterations.

The for loop iteration always starts from zero.



Figure 6 For Loop Structure

Example (2):

Use the for loop structure to make a flashing ring of seven leds with the following specification. The first led light then the second led light and the first led remains light then the third led light and the first and second led remains light and so on until the seventh led light, then all the leds are tuned off and repeated the previous repetition.

Solution:

1. Design the front panel with seven leds and a Knob which is used to control the speed as shown in figure.



Figure 7 Front panel

2. Since we have seven led flasher we need the loop to run eight time because in the final loop all leds off so we put the counter terminal to 8 as shown in figure below, put delay equal 0.5sec.

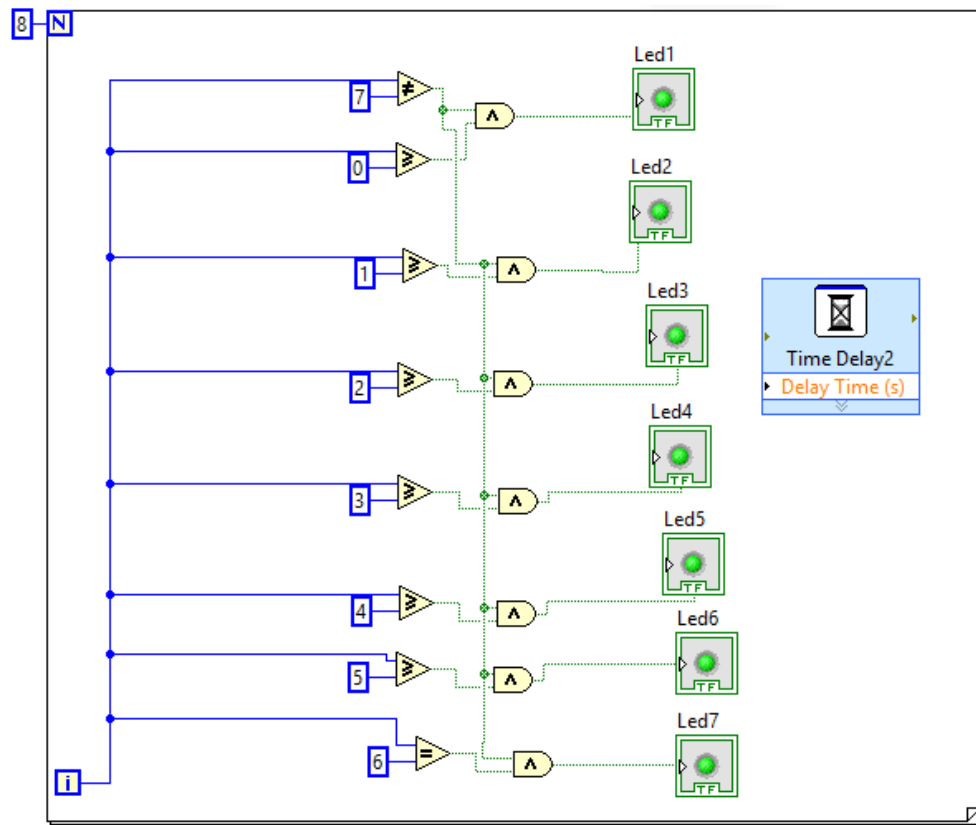


Figure 8 Block diagram

❖ **After run program:**

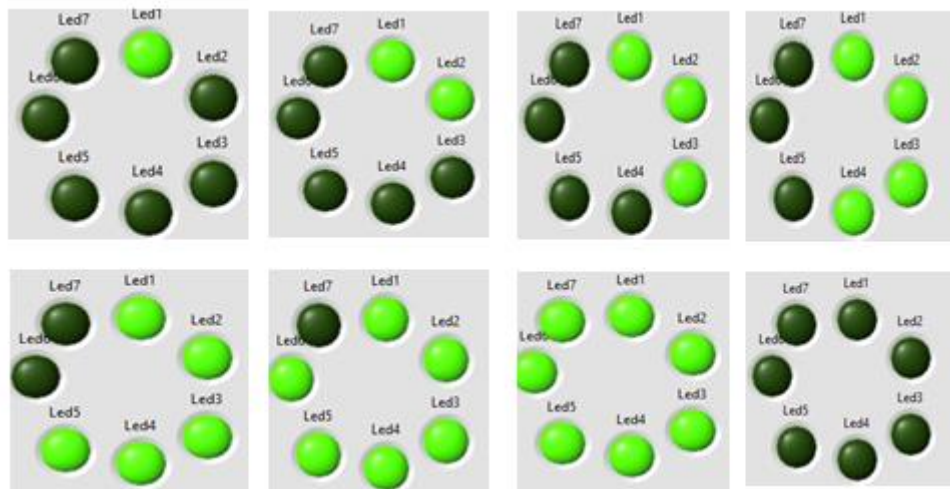


Figure 9 Front panel

Report exercise (2):

Use the for loop structure to make a flasher of one led with following specification:

- ❖ The user should control the flashing speed.
- ❖ Hint (the main idea is to divide the iteration by 2 and see if the result (القسمة باقي) equal zero or not, you can find the Modulo operator on mathematics).

Report exercise (3):

Use the loop structures to make a flashing wave of five leds with following specifications:-

- ❖ The user should control the flashing speed.
- ❖ The system contains a button to change the flashing direction (bio directional flasher).

Thirdly: Case Structure

A Case structure has two or more sub diagrams, or cases. Only one sub diagram is visible at a time, and the structure executes only one case at a time. An input value determines which sub diagram executes. The Case structure is similar to switch statements or if...then...else statements in text-based programming languages see figure.

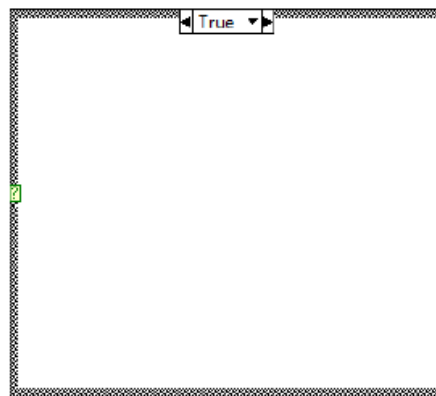
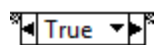


Figure 10 Case Structure



The case selector label at the top of the Case structure contains the name of the selector value that corresponds to the case in the center and decrement and increment arrows on each side. Click the decrement and increment arrows to scroll through the available cases. You also can click the down arrow next to the case name and select a case from the pull down menu.



Wire an input value, or selector, to the selector terminal to determine which case executes. You must wire an integer, Boolean value, string, or enumerated type value to the selector terminal. You can position the selector terminal anywhere on the left border of the Case structure. If the data type of the selector terminal is Boolean, the structure has a True case and a false case. If the selector terminal is an integer, string, or enumerated type value, the structure can have any number of cases. Right-click the Case structure border to add, duplicate, remove, or rearrange cases, and to select a default case.

Example3:

Design a LabVIEW program to calculate the area of a circle, and circumference of a circle using case structure.

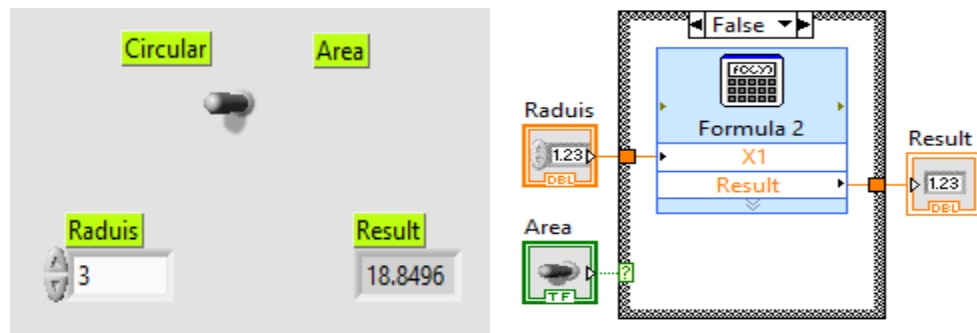
Solution:

Figure 11 Front panel and Block diagram

Example4:

Built a simple function generator that generates sin, square, and triangular waves. The user should control the amplitude, frequency, and phase (delay) using case structure.

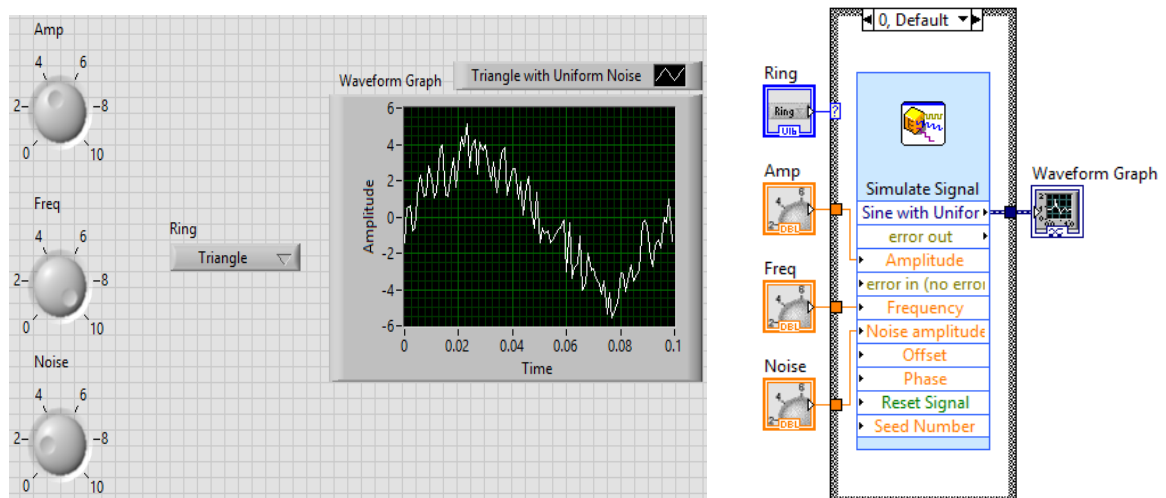
Solution:

Figure 12 Front panel and Block diagram

Experiment 4

System Modeling By MATLAB & LabVIEW

Objectives:

- Find a mathematical model, called a state-space representation, for a linear, time invariant system using MATLAB and LabVIEW.
- Convert a transfer function to state space.
- Convert a state-space representation to a transfer function.
- Reduce a block diagram of multiple subsystems to a single block representing the transfer function.

Introduction:

The first step in the control design process is to develop appropriate mathematical models of the system to be controlled. These models may be derived either from physical laws or experimental data. In this section, we introduce the state-space and transfer function representations of dynamic systems. We then review some basic approaches to modeling mechanical and electrical systems and show how to generate these models in MATLAB for further analysis.

Modeling is the translator that represents a physical device by its equivalent equations. In modeling we need some standard physical laws that relate our variables.

○ Types of Model Representation

- Transfer Function Models.
- State Space Models.

Firstly // Transfer Function Models

Transfer function models describe the relationship between the inputs and outputs of a system using a ratio of polynomials. The model order is equal to the order of the denominator polynomial. The roots of the denominator polynomial are referred to as the model poles. The roots of the numerator polynomial are referred to as the model zeros. The parameters of a transfer function model are its poles and zeros.

Now form the ratio of the output transform(s), divided by the input transform R(s) called transfer function G(s).

$$G(s) = \frac{C(s)}{R(s)}, \text{ For Example } G(s) = \frac{s+2}{s^2+s+10}$$

Secondly // State Space Models.

For continuous linear time-invariant (LTI) systems, the standard state-space representation is given below:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$$

$\mathbf{x}$ = state vector

$\dot{\mathbf{x}}$ = derivative of the state vector with respect to time

$\mathbf{y}$ = output vector

$\mathbf{u}$ = input or control vector

$\mathbf{A}$ = system matrix

$\mathbf{B}$ = input matrix

$\mathbf{C}$ = output matrix

$\mathbf{D}$ = feedforward matrix

 MATLAB Commands & LabVIEW Structure:

There are many processes in which the two programs can be used to achieve them. Examples of these processes:

1. Store transfer function.
2. Cascade between two transfer function.
3. Parallel between two transfer function.
4. Feedback between two transfer function.
5. Store State Space.
6. Convert from transfer function to state space.
7. Convert from state space to transfer function.
8. Reduce a block diagram of multiple subsystems to a single block representing the transfer function.

To achieve these operations in the LabVIEW program. You must download *Control Design and Simulation library*.

The LabVIEW Control Design and Simulation Module helps you simulate dynamic systems, design controllers, and deploy control systems to real-time hardware.

➤ **Control Design and Simulation library.**

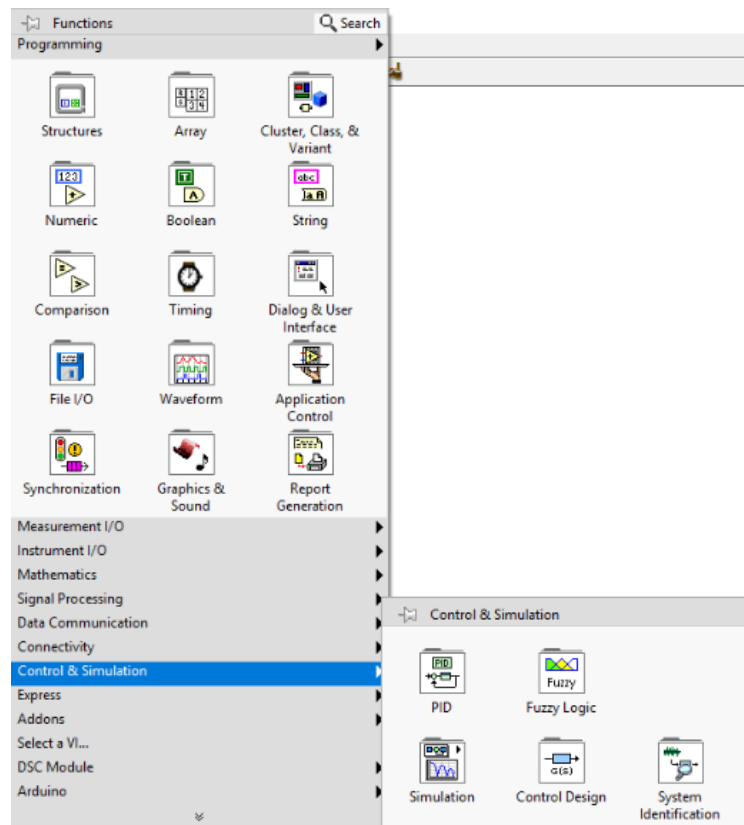


Figure 1 Control Design and simulation Module

With the **NI LabVIEW Control Design and Simulation Module**, you can analyze open-loop model behavior, design closed-loop controllers, simulate online and offline systems, and conduct physical implementations.

Create models from first principles using transfer function, state-space, or zero-pole-gain representation. With time and frequency analysis tools, such as time step response or Bode plot, you can interactively analyze open- and closed-loop behavior. Use built-in tools for both multiple input, multiple output (MIMO) and single input, single output (SISO) systems and take advantage of simulation capabilities to verify linear and nonlinear system dynamics.

1- Store transfer function by MATLAB and LabVIEW:

$$G(s) = \frac{s+2}{s^2+s+10}$$

○ By MATLAB:

```
>> num = [1 2];
>> den = [1 1 10];
>> G = tf(num,den)
```

```
G =

      s + 2
-----
s^2 + s + 10
```

Figure 2 To store TF by MATLAB

○ By LabVIEW

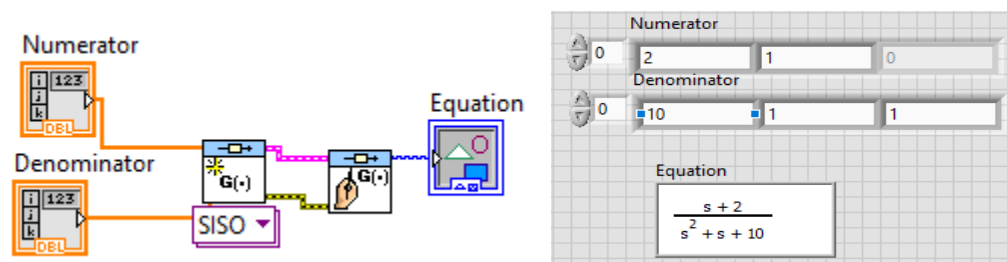


Figure 3 To store TF by LabVIEW

To achieve the required we follow the following:

- Block diagram>>Function>>Control and Simulation>>Control Design>>Module Construction>>Construct Transfer Function.
- Block diagram>>Function>>Control and Simulation>>Control Design>>Module Construction>>Draw Transfer Function.

2- Cascade between two transfer function.

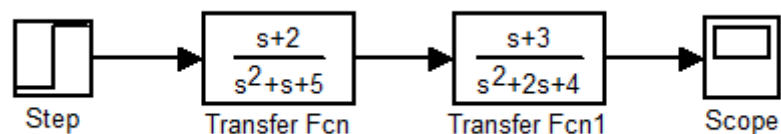


Figure 4 Cascade transfer function system

○ By MATLAB:

```
>> num1 = [1 2];
>> den1 = [1 1 5];
>> num2 = [1 3];
>> den2 = [1 2 4];
>> [num den] = series(num1,den1,num2,den2);
>> G = tf(num,den)
```

G =

$$\frac{s^2 + 5s + 6}{s^4 + 3s^3 + 11s^2 + 14s + 20}$$

Figure 5 Series TF

○ By MATLAB(Another method):

```
>> num1 = [1 2];
den1 = [1 1 5];
G1 = tf(num1,den1);
num2 = [1 3];
den2 = [1 2 4];
G2 = tf(num2,den2);
G3 = series(G1,G2)
```

G3 =

$$\frac{s^2 + 5s + 6}{s^4 + 3s^3 + 11s^2 + 14s + 20}$$

Continuous-time transfer function.

Figure 6 Another way for Series TF

○ BY LabVIEW:

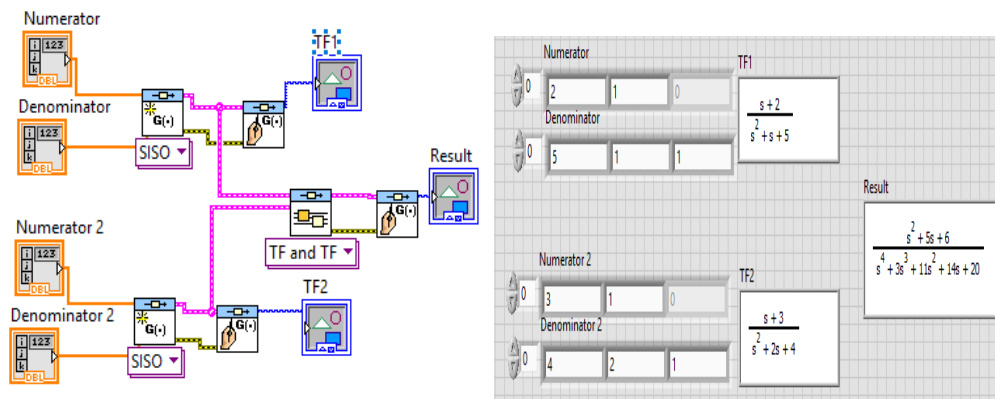


Figure 7 Series TF by LabVIEW

To achieve the required we follow the following:

- Block diagram>>Function>>Control and Simulation>>Control Design>>Module Interconnection>>CD Series.

3- Parallel between two transfer function.

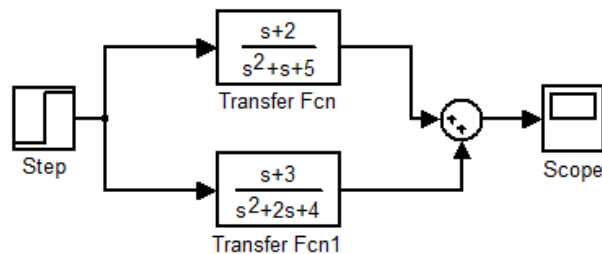


Figure 8 Parallel Transfer Function System

○ By MATLAB :

```
>> num1 = [1 2];
den1 = [1 1 5];
G1 = tf(num1,den1);
num2 = [1 3];
den2 = [1 2 4];
G2 = tf(num2,den2);
G3 = parallel(G1,G2)

G3 =

      2 s^3 + 8 s^2 + 16 s + 23
-----
s^4 + 3 s^3 + 11 s^2 + 14 s + 20

Continuous-time transfer function.
```

Figure 9 Parallel TF

○ BY LabVIEW:

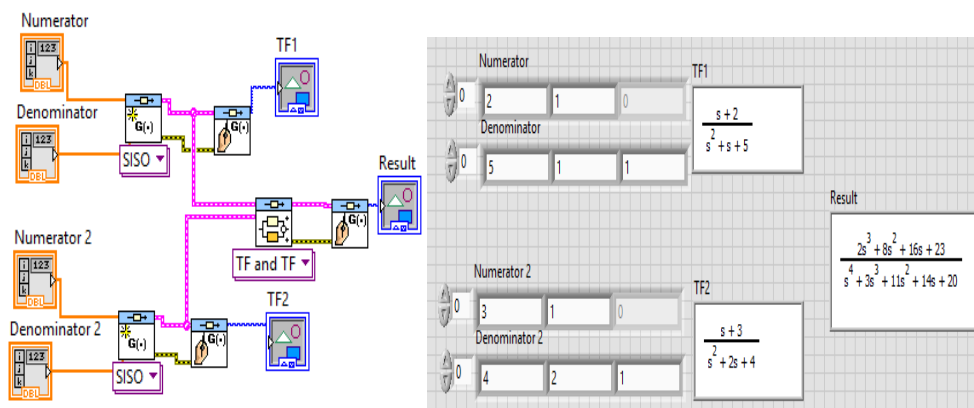


Figure 10 Parallel TF by LABVIEW

To achieve the required we follow the following:

- Block diagram>>Function>>Control and Simulation>>Control Design>>Module Interconnection>>CD Parallel.

4- Feedback between two transfer function.

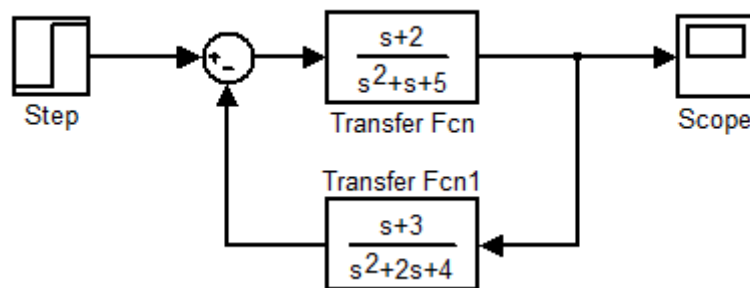


Figure 11 Feedback Transfer Function System

○ By MATLAB:

```
>> num1 = [1 2];
den1 = [1 1 5];
G1 = tf(num1,den1);
num2 = [1 3];
den2 = [1 2 4];
G2 = tf(num2,den2);
G3 = feedback(G1,G2)

G3 =

          s^3 + 4 s^2 + 8 s + 8
-----
      s^4 + 3 s^3 + 12 s^2 + 19 s + 26

Continuous-time transfer function.
```

Figure 12 Feedback TF

○ By LabVIEW:

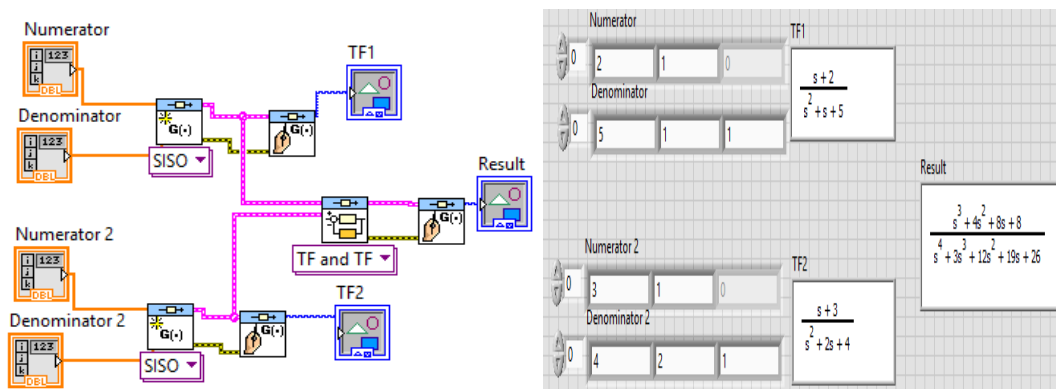


Figure 13 Feedback TF by LABVIEW

To achieve the required we follow the following:

- Block diagram>>Function>>Control and Simulation>>Control Design>>Module Interconnection>>CD Feedback

5- Store State Space by MATLAB and LabVIEW:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 0 \end{bmatrix}, B = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \\ C = \begin{bmatrix} 2 & 2 \end{bmatrix}, D = 0$$

Figure 14 State Space Model

○ By MATLAB:

```

a =
    x1    x2
    x1     1     2
    x2     3     0

b =
    u1
    x1     1
    x2     2

c =
    x1    x2
    y1     2     2

d =
    u1
    y1     0

>> A = [1 2 ; 3 0];
B = [1 ; 2];
C = [2 2];
D = 0;
G = ss(A,B,C,D)

```

Continuous-time state-space model.

Figure 15 Construct SS by MATLAB

○ By LabVIEW:

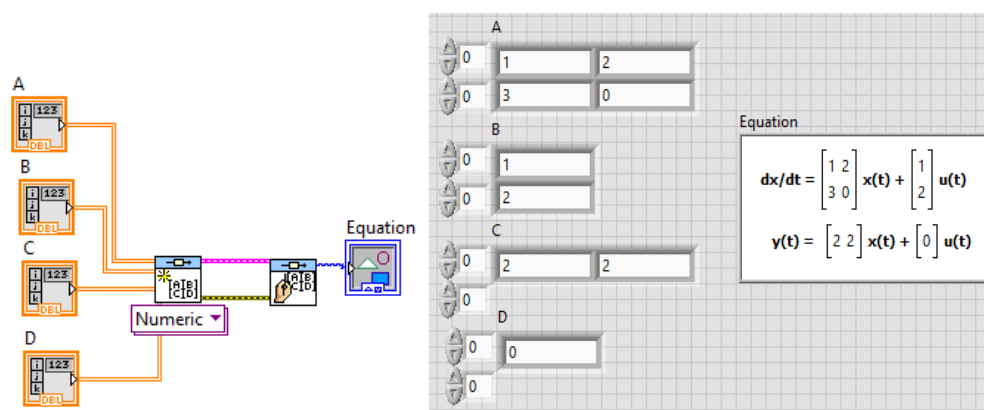


Figure 16 Construct SS by LABVIEW

To achieve the required we follow the following:

- Block diagram>>Function>>Control and Simulation>>Control Design>>Module Construction>>Construct State Space.
- Block diagram>>Function>>Control and Simulation>>Control Design>>Module Construction>>Draw State Space.

6- Convert from transfer function to state space.

For example, convert $G(s) = \frac{s+2}{s^2+s+10}$ to State space equivalent

○ By MATLAB:

```
>> num = [1 2];
    den = [1 1 10];
    [A,B,C,D] = tf2ss(num,den);
    G = ss(A,B,C,D)
```

Continuous-time state-space model.

a =

	x1	x2
x1	-1	-10
x2	1	0

b =

	u1
x1	1
x2	0

c =

	x1	x2
y1	1	2

d =

	u1
y1	0

Figure 17 Convert TF to SS

○ By LabVIEW:

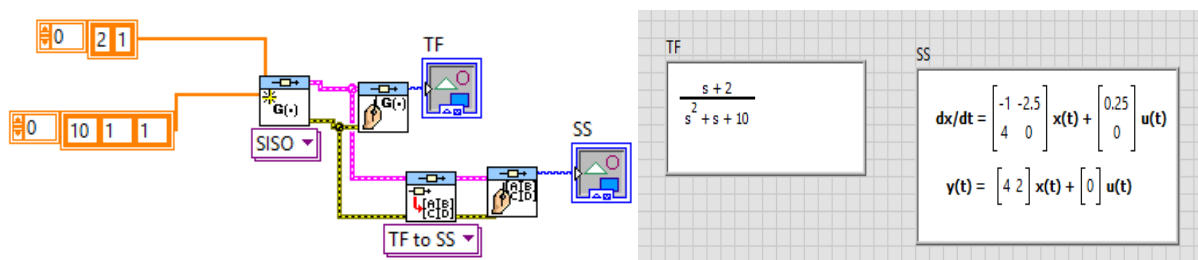


Figure 18 Convert TF to SS by LABVIEW

To achieve the required we follow the following:

- Block diagram>>Function>>Control and Simulation>>Control Design>>Model Conversion>>Convert transfer function to State Space.

7- Convert from state space to transfer function.

For example, convert to Transfer function equivalent

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 0 \end{bmatrix}, B = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \\ C = \begin{bmatrix} 2 & 2 \end{bmatrix}, D = 0$$

Figure 19 State Space Model

○ By MATLAB:

```
G =
      6 s + 10
      -----
      s^2 - s - 6

Continuous-time transfer function. G = tf(num,den)
```

```
>> A = [1 2 ; 3 0];
B = [1 ; 2];
C = [2 2];
D = 0;
[num den] = ss2tf(A,B,C,D)
```

Figure 20 Convert from SS to TF

○ By LabVIEW:

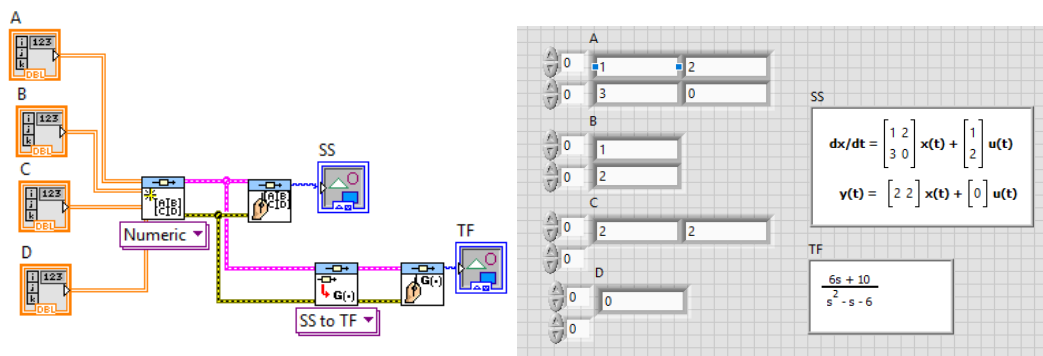


Figure 21 Convert from SS to TF by LABVIEW

To achieve the required we follow the following:

- Block diagram>>Function>>Control and Simulation>>Control Design>>Model Conversion>>Convert State Space to transfer function.

8- Reduce a block diagram of multiple subsystems to a single block representing the transfer function.

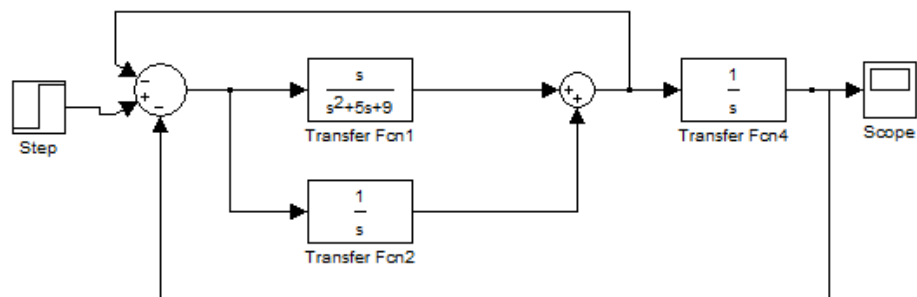


Figure 22 Multiple Subsystem

○ By MATLAB:

```
>> num1 = [1 0];
den1 = [1 5 9];
G1 = tf(num1,den1)
num2 = 1;
den2 = [1 0];
G2 = tf(num2,den2)

G3 = parallel(G1,G2);
G4 = feedback(G3,1);
G5 = series(G4,G2);
G6 = feedback(G5,1)
```

Continuous-time transfer function.

$$G6 = \frac{2s^2 + 5s + 9}{s^4 + 7s^3 + 16s^2 + 14s + 9}$$

Figure 23 Reducing to TF

○ By LabVIEW:

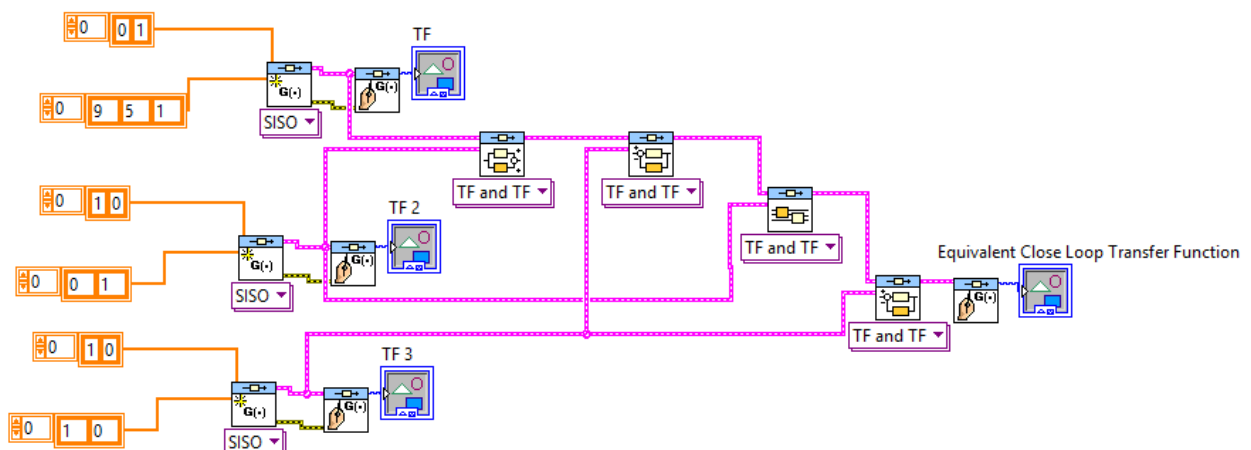


Figure 24 Reducing System to TF

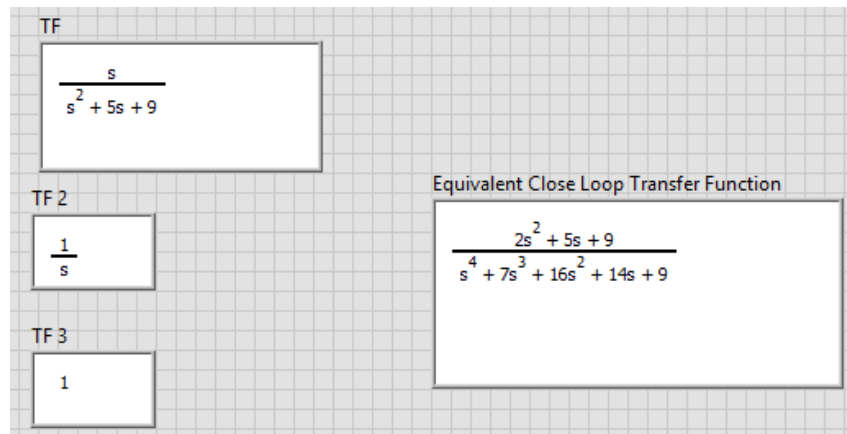


Figure 25 Front panel for reducing system to TF

Experiments 5

Time Response Analysis

Objectives:

- To study the transient response of first-order systems and second-order systems.
- To study how can be Find the settling time, peak time, percent overshoot, and rise time for an underdamped second-order system.
- To draw the time response from transfer function and state-space representation by MATLAB and LabVIEW.

Introduction:

What is Time Response?

If the output of control system for an input varies with respect to time, then it is called the **time response** of the control system. The time response consists of two parts:

- Transient response.
- Steady state response.

The response of control system in time domain is shown in the following figure.

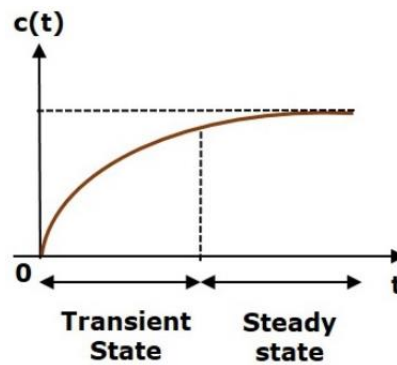


Figure 1 Time Response of the Control System

Here, both the transient and the steady states are indicated in the figure. The responses corresponding to these states are known as transient and steady state responses.

Mathematically, we can write the time response $c(t)$ as:

$$y(t) = y_t(t) + y_{ss}(t)$$

$y_t(t)$: Transient response

$y_{ss}(t)$: Steady state response

❖ Transient Response

After applying input to the control system, output takes certain time to reach steady state. So, the output will be in transient state till it goes to a steady state. Therefore, the response of the control system during the transient state is known as transient response.

✍ **Note:** If the input is step function the output or the response is called step time response and if the input is ramp, the response is called ramp time response.

❖ Steady state Response

The part of the time response that remains even after the transient response.

Step Time Response Specification:

We have defined two parameters associated with second-order systems, ζ and W_n . Other parameters associated with the underdamped response are rise time, peak time, percent overshoot, and settling time. These specifications are defined as follows.

1. **Rise time, T_r .** The time required for the waveform to go from 0.1 of the final value to 0.9 of the final value.
2. **Peak time, T_p .** The time required to reach the first, or maximum, peak.
3. **Percent overshoot, %OS.** The amount that the waveform overshoots the steady state, or final, value at the peak time, expressed as a percentage of the steady-state value.
4. **Settling time, T_s .** The time required for the transient's damped oscillations to reach and stay within 2% of the steady-state value.
5. **Peak time, T_P .** It is the time required for the underdamped step response to reach the first maximum peak.
6. **The DC gain** is the ratio of the steady state step response to the magnitude of a step input. For example if your input is step function with amplitude = 1 and found the step response output = 10 then the DC gain = $10/1 = 10$. In other words it is the value of the transfer function when $s=0$.

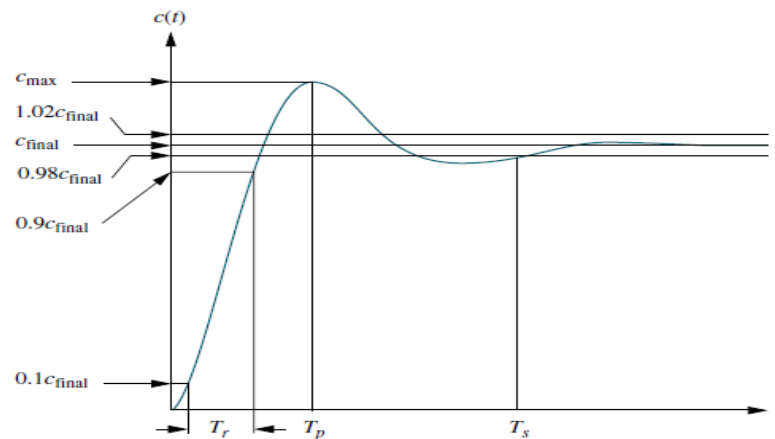


Figure 2 Time Response Specification

Types of Step Natural Response:

1. Overdamped Responses

Poles: Two real at $-\sigma_1, -\sigma_2$

real & distinct poles

$$c(t) = K_1 e^{-\sigma_1 t} + K_2 e^{-\sigma_2 t}$$

2. Underdamped responses

Poles: Two complex at $-\sigma_d \pm j\omega_d$

complex poles

$$c(t) = A e^{-\sigma_d t} \cos(\omega_d t - \phi)$$

3. Undamped responses

Poles: Two imaginary at $\pm j\omega_1$

imaginary poles

$$c(t) = A \cos(\omega_1 t - \phi)$$

4. Critically damped responses

Poles: Two real at $-\sigma_1$

real & repeated poles

$$c(t) = K_1 e^{-\sigma_1 t} + K_2 t e^{-\sigma_1 t}$$

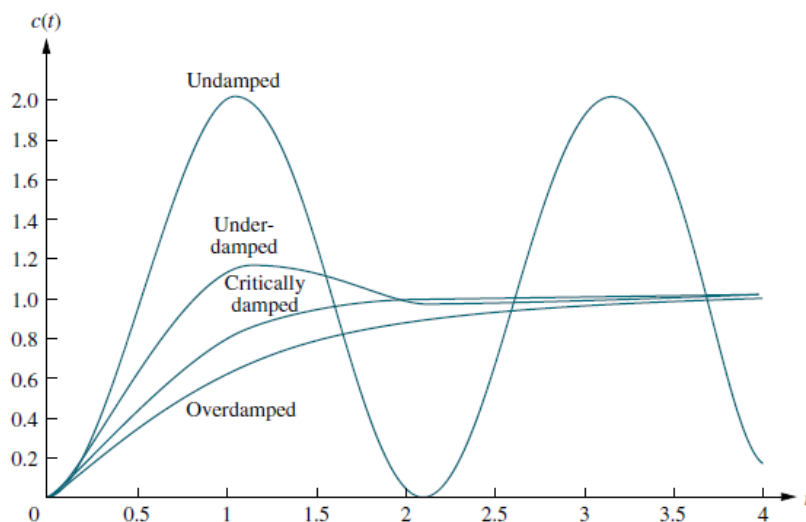


Figure 3 Types of Response

Characteristic of various systems:

1. **First Order system :**

The first order system can take the general form:

$$G(s) = \frac{b}{(s + a)} = \frac{K_{dc}}{(ts + 1)}$$

- **DC Gain:** is $K_{dc} = b/a$ by setting $s=0$ in the transfer function.

- **Time constant (t):**

It is the time required to reach 63% of the steady state value or final value for a step input or to decrease to 37% of the initial value and it is special for first order system only.

Time constant

$$t = \frac{1}{a}$$

Rise time

$$T_r = \frac{2.2}{a}$$

Settling time

$$T_s = \frac{4}{a}$$

The first order system has no overshooting but can be stable or not depending on the location of its pole.

The first order system has a single pole at -a. If the pole is on the negative real axis (LHP), then the system is stable. If the pole is on the positive real axis (RHP), then the system is not stable. The zeros of a first order system are the values of s which makes the numerator of the transfer function equal to zero.

2. **Second Order system :**

The general form of second order system is:

$$G(s) = \frac{a}{(s^2 + bs + c)} = \frac{K_{dc} \cdot \omega_n^2}{(s^2 + 2\zeta\omega_n s + \omega_n^2)}$$

▪ **Undamped Natural frequency ω_n :**

is the frequency of oscillation of the system without damping.

▪ **Damping Ratio $\zeta = b/2\omega_n$**

Note that the system has a pair of complex conjugate poles at:

$$S = -\zeta\omega_n \pm j\omega_n\sqrt{1-\zeta^2} = -\sigma \pm j\omega_d$$

ω_d : Damped frequency of oscillation

DC gain

$$K_{dc} = \frac{a}{c}$$

Percent overshoot

$$os\% = e^{\frac{\pi\zeta}{\sqrt{1-\zeta^2}}} * 100$$

Settling time

$$T_s = \frac{4}{\zeta\omega_n} = \frac{4}{\sigma}$$

Peak time

$$T_p = \frac{\pi}{\omega_n\sqrt{1-\zeta^2}}$$

Rise time

$$T_r \cong \frac{2.2}{\omega_n}$$

✚ How can be draw the step response by Matlab code, Simulink and LabVIEW for transfer function and state space representation.

For Example,

Transfer function:
$$G(s) = \frac{10}{s^2 + 2s + 10}$$

State space:
$$A = \begin{bmatrix} 0 & 1 \\ -9 & -2 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

$$C = [9 \quad 0], D = 0$$

Firstly// MATLAB Code:

- Transfer Function:

```
>> num = 10;
den = [1 2 10];
G = tf(num,den);
step(G)
grid;

>>
```

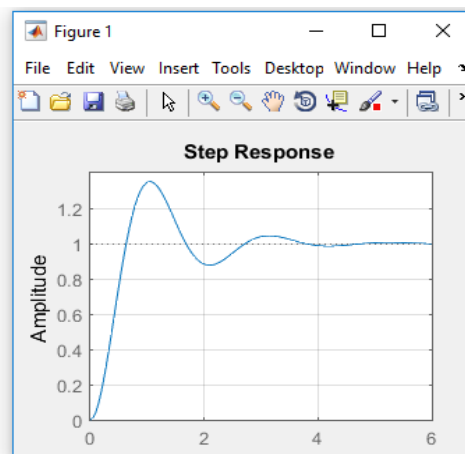


Figure 4 Construct TF by MATLAB code

- State Space:

```
>> A = [0 1 ; -9 -2];
B = [0 ; 1];
C = [9 0];
D = 0;
sys = ss(A,B,C,D);
step(sys)
grid;

>>
```

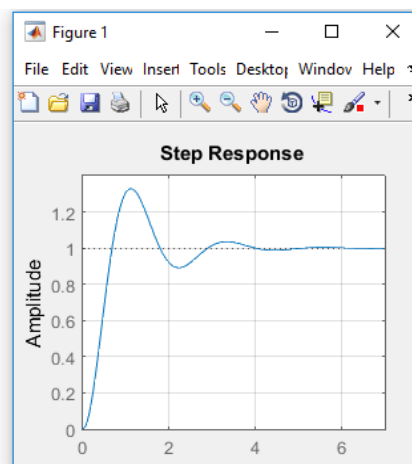


Figure 5 Construct SS by MATLAB code

Secondly // Simulink.

We can do the same work by Simulink in MATLAB.

❖ Components :

For most of the systems we will encounter, we only need to be concerned with a small fraction of Simulink's Component library. In particular, the components you should be familiar with are:

1. "Continuous" library:

Integrator—integrates a signal.

State-Space—used to add a system block in state-space form.

Transfer Function—used to add a system block in transfer function form.

2. "Math Operations" library:

Gain—a constant gain.

Sum—used to add two or more signals.

3. "Sinks" library:

Scope—used for viewing system output.

To workspace—used to transfer a signal to MATLAB.

4. "Sources" library:

Ramp—generates a ramp signal.

Sine Wave—generates a sinusoid.

Step—generates a unit step signal.

- Transfer Function:

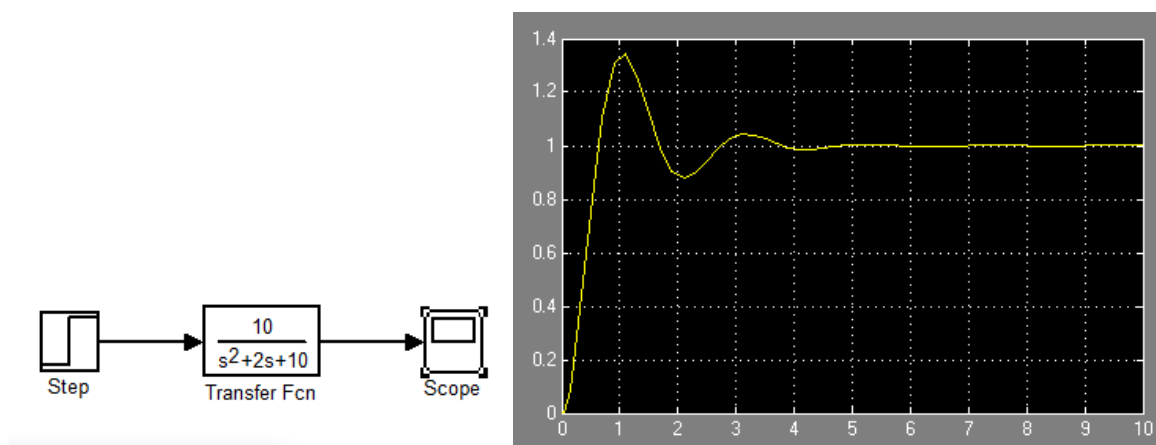


Figure 6 Construct TF by Simulink

- State Space:

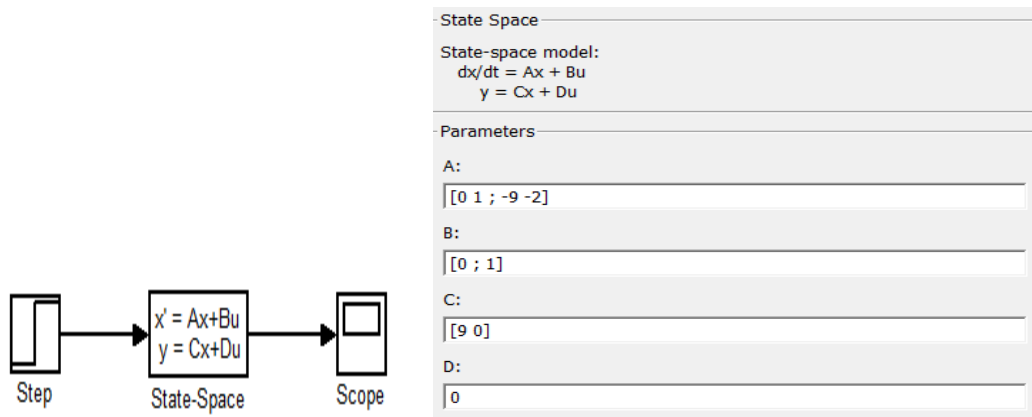


Figure 7 Construct SS by Simulink

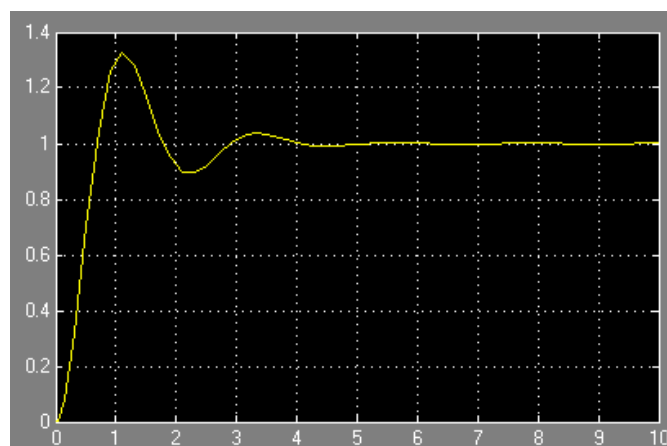


Figure 8 Step Response

Thirdly // LabVIEW.

Open Blank VI in LabVIEW then go to the block diagram window and open the function palette then select control design and simulation library.

Add control and simulation loop as area for your simulation then add the function generator and linear control system then display the output on the Sim-time waveform.

- Transfer Function:

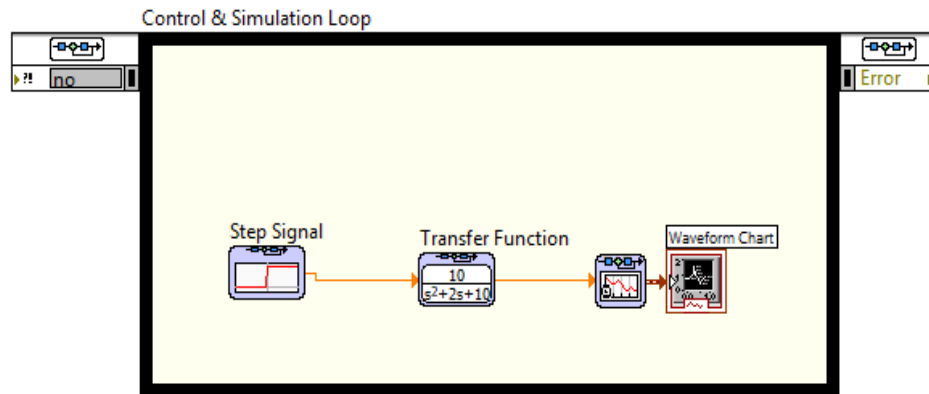


Figure 9 Construct TF by LABVIEW

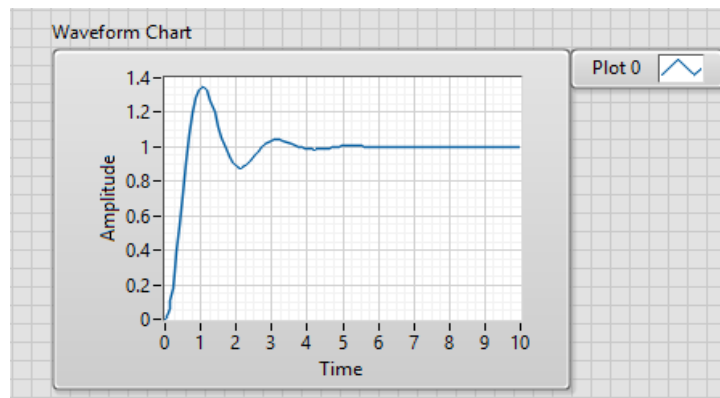


Figure 10 Step Response

- State Space:

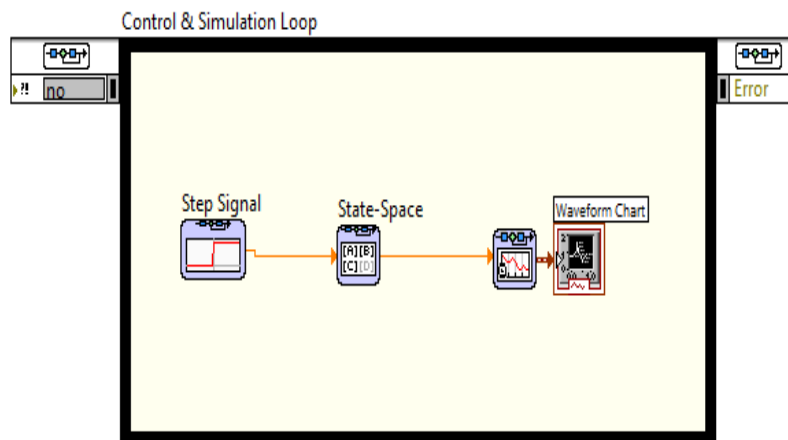


Figure 11 Construct SS by LABVIEW

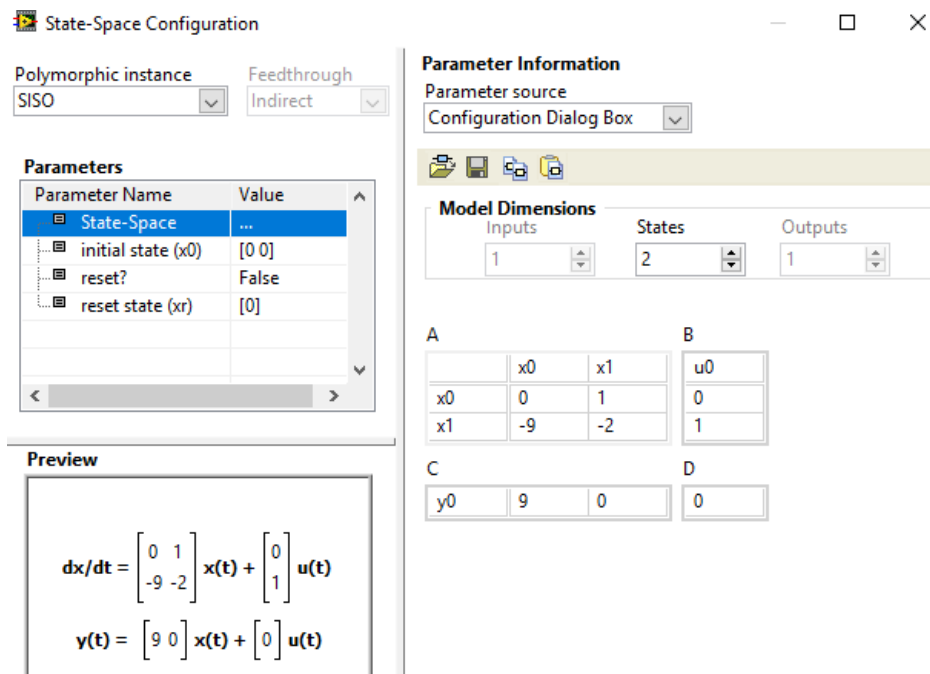


Figure 12 Configuration SS in LABVIEW

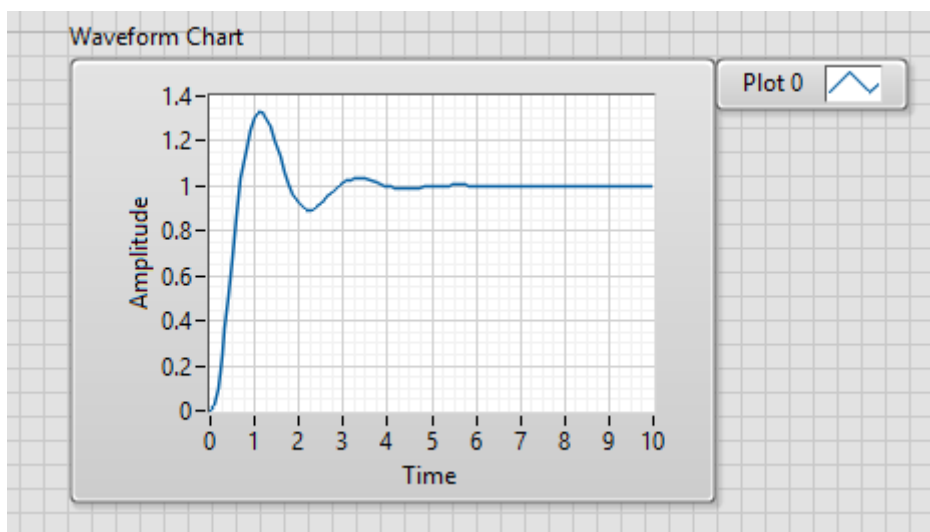


Figure 13 Step Response

Experiments 6

Implementing system using Op-Amp

Objectives:

- To study Operational Amplifier (Op-amp) Applications.
- To study what is all integral block diagram and Physical Realization of a system by op-amp.
- Implementing Transfer Function Using OP-Amps.
- To study how can be convert the block diagram to Op-Amps implementation.

Introduction:

The Op-Amp stand for Operational Amplifier, is basically a differential amplifier having a large voltage gain, very high input impedance and low output impedance. The op-amp has a "inverting" or (-) input and "non-inverting" or (+) input and a single output. The op-amp is usually powered by a dual polarity power supply in **the range of (+/-) 5 volts to (+/-) 15 volts.**

The circuit symbol for an op-amp is shown to the right, where:

- **V_+ : non-inverting input**
- **V_- : inverting input**
- **V_{out} : output**
- **V_{S+} : positive power supply**
- **V_{S-} : negative power supply**

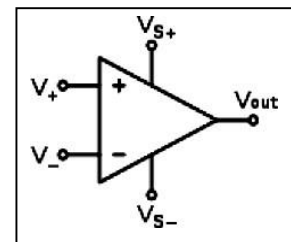
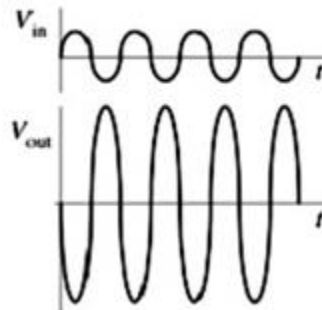
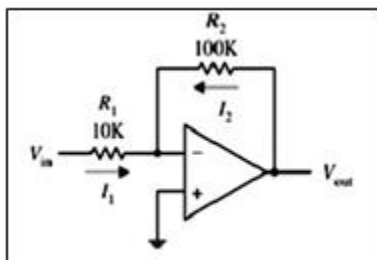


Figure 1 Operational Amplifier

Operational amplifiers can be used to perform mathematical operations on voltage signals such as inversion, addition, subtraction, integration, differentiation, and multiplication by a constant and comparison.

○ Operational Amplifier (Op-amp) Applications:

A. The inverting op-amp: (It can be used as K controller).



$$\text{Gain} = \frac{V_o}{V_i} = -\frac{R_2}{R_1}$$

Figure 2 inverting Op-Amp

B. The non-inverting op-amp: (It can be used as K controller).

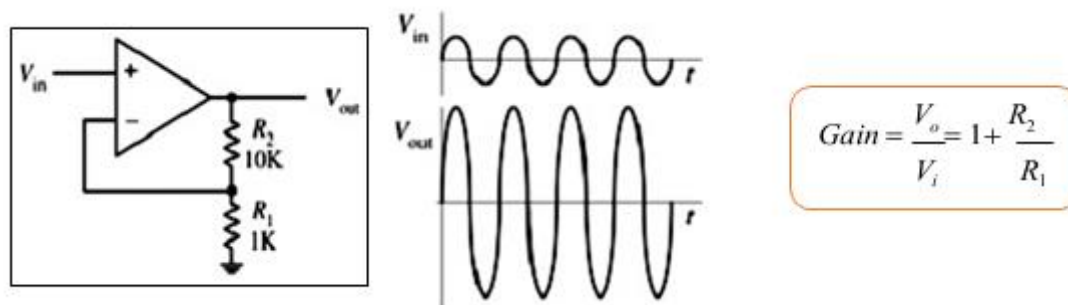


Figure 3 Non-inverting Op-Amp

C. The summer : (Adder)

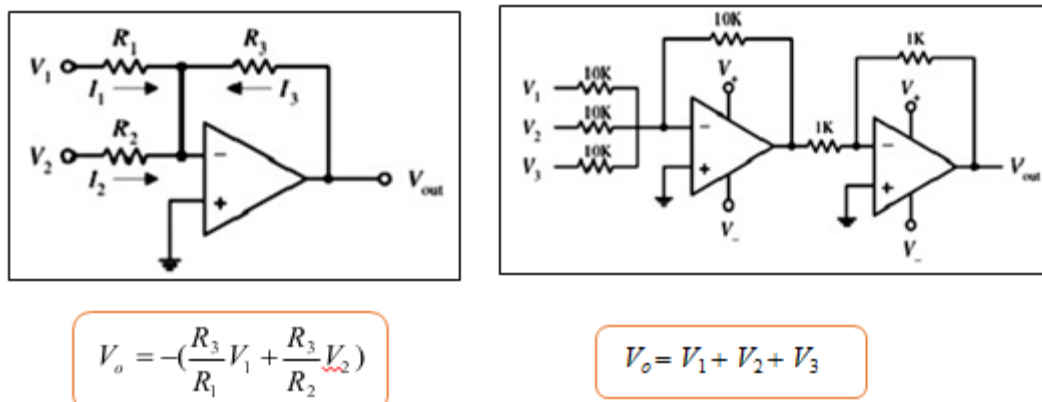


Figure 4 Summer Op-Amp

D. The differential amplifier: (Subtract)

$$V_o = \frac{R_2}{R_1} (V_2 - V_1)$$

If you set $R_1 = R_2$ then:

$$V_o = (V_2 - V_1)$$

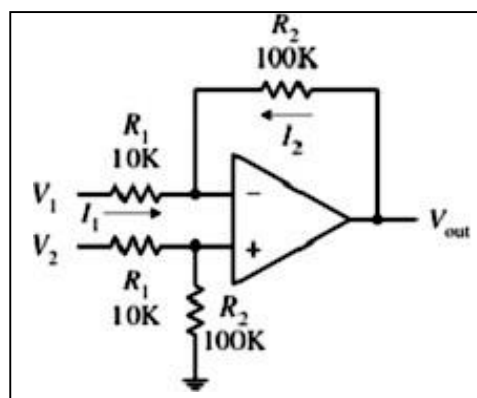


Figure 5 Differential Amplifier

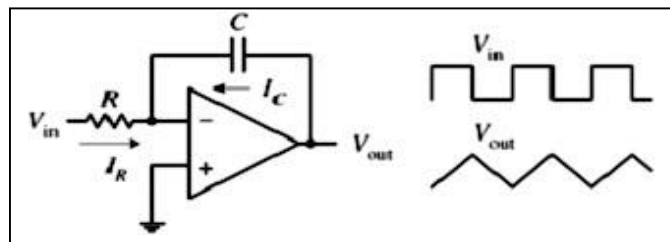
E. The integrator:

Figure 6 Integrator Op-Amp

$$V_o = - \frac{1}{RC} \int V_{in} dt$$

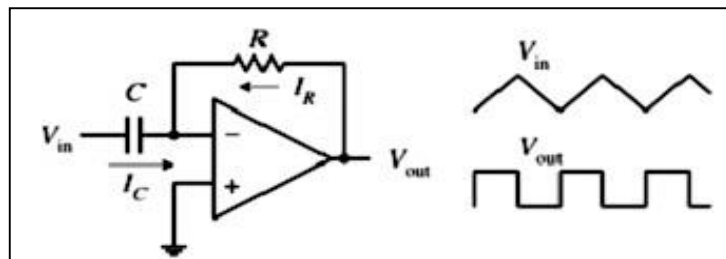
F. The differentiator:

Figure 7 Differentiator Op-Amp

$$V_o = -RC \frac{\partial V_{in}}{\partial t}$$

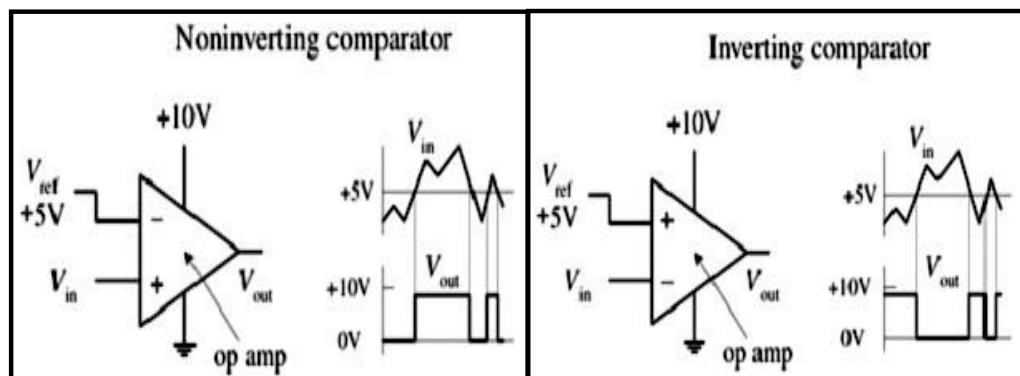
G. The comparator:

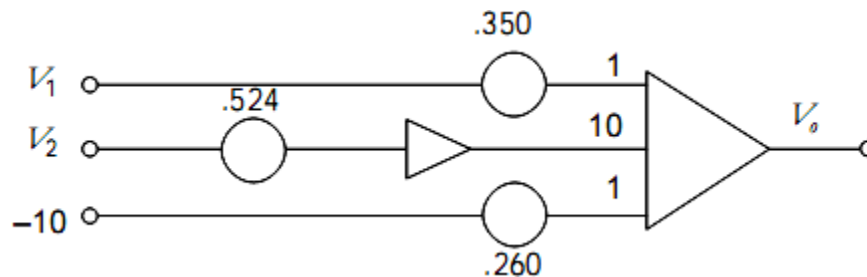
Figure 8 Comparator Op-Amp

V_{ref} : Reference voltage used as reference to compare between it and the input voltage.

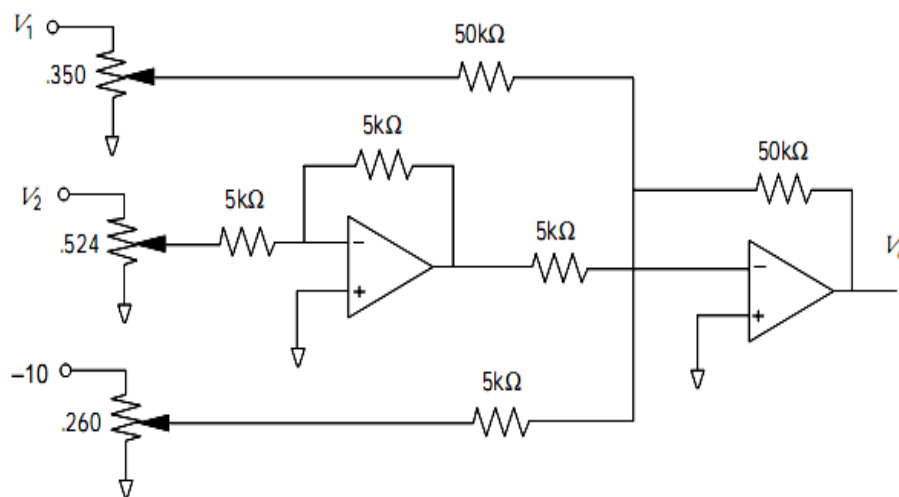
Example 1:

Determine the analog diagram and the circuit to implement the following equation

$$V_o = -0.35V_1 + 5.24V_2 + 2.6.$$



Note the inversion operation of all inputs that is because the summer inverts the output as we introduced before and the inverter must have a unity gain.

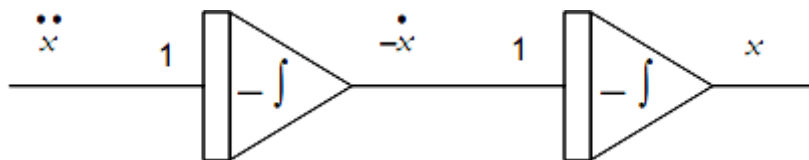


Example 2:

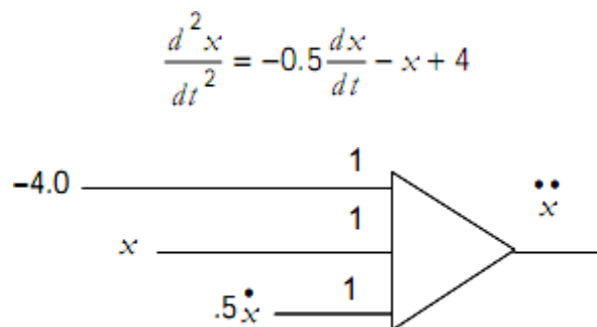
Determine the analog diagram and the circuit to implement the following system

$$\frac{d^2 x}{dt^2} + 0.5 \frac{dx}{dt} + x = 4$$

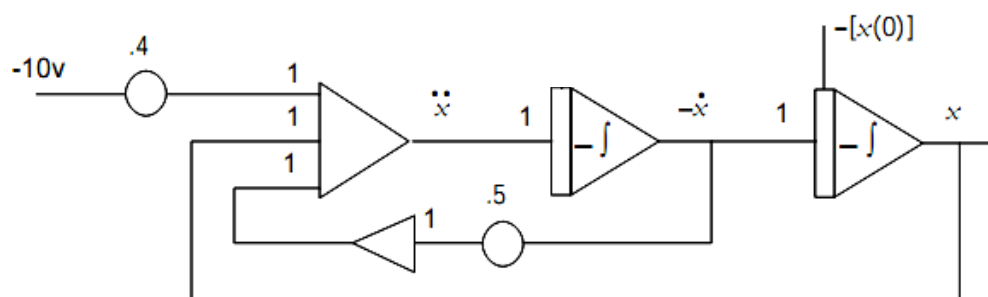
From the above equation we know that we will use two integrators at least as follows



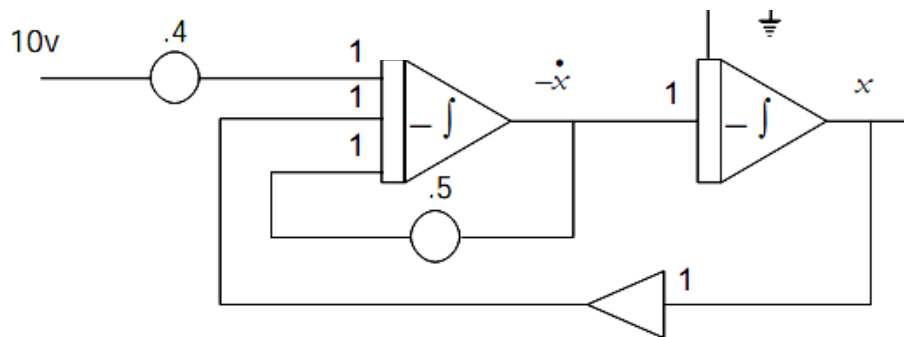
And by rearranging the equation, we conclude that



Don't forget that summer and integrators invert the output

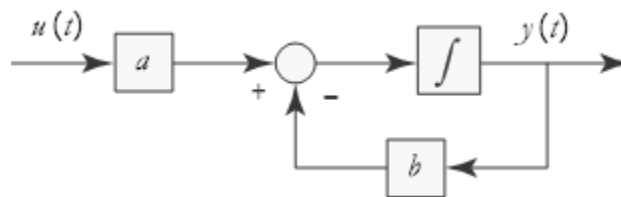


Because we know that integrator can do summation also, so we combine the two first blocks (summer, integrator) by one as follows, but we should invert the input to obtain the suitable output.



Note:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{a}{s+b},$$

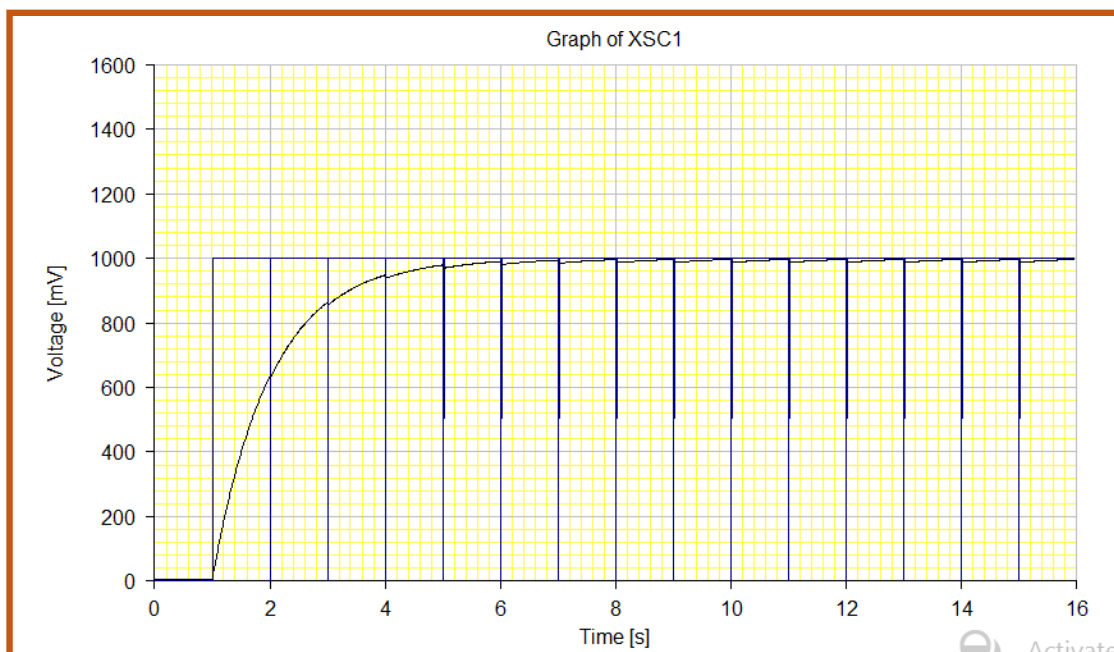
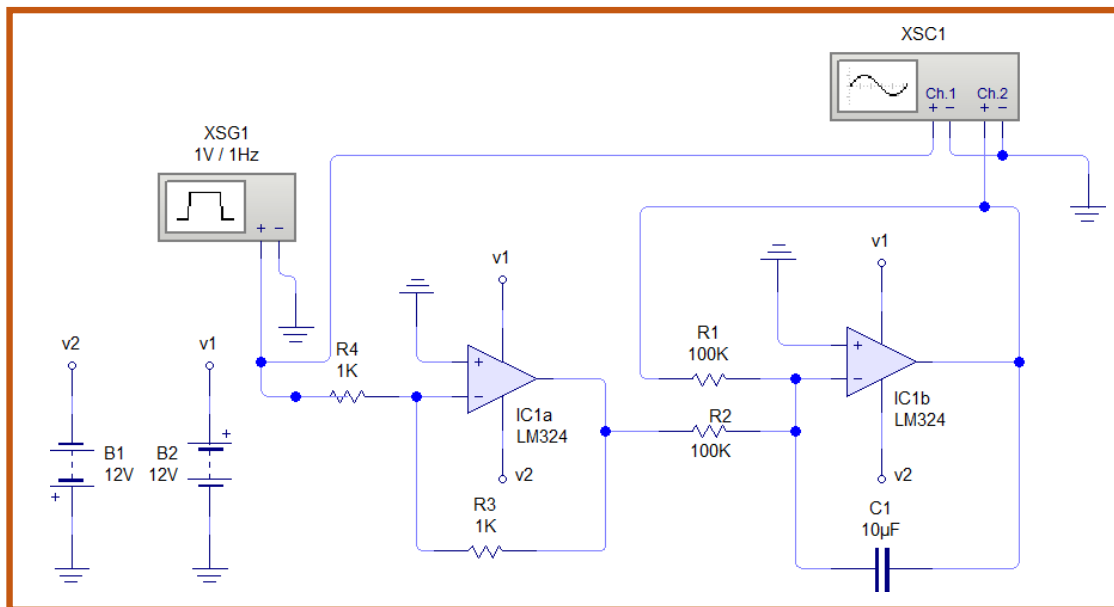


This is all integral block diagram, and we implement it by op-amp take care to invert the inputs.

Example 3:

Determine the circuit to implement the following system

$$G(s) = \frac{1}{s+1}$$



○ Implementing Transfer Function Using OP-Amps:

To implement the transfer function by using operation amplifiers there is some steps to do this, we will explain it as shown in the next example

Example: Given the T.F, implement it using by OP-AMPs

$$G(s) = \frac{Y(s)}{U(s)} = \frac{6}{s^2 + 2s + 12}$$

Firstly: Convert the T.F to State Space the following MATLAB code make this function:

```
num=[6];  
  
den=[1 2 12];  
  
G=tf(num,den)  
  
[a,b,c,d]=tf2ss(num,den)
```

Transfer function:

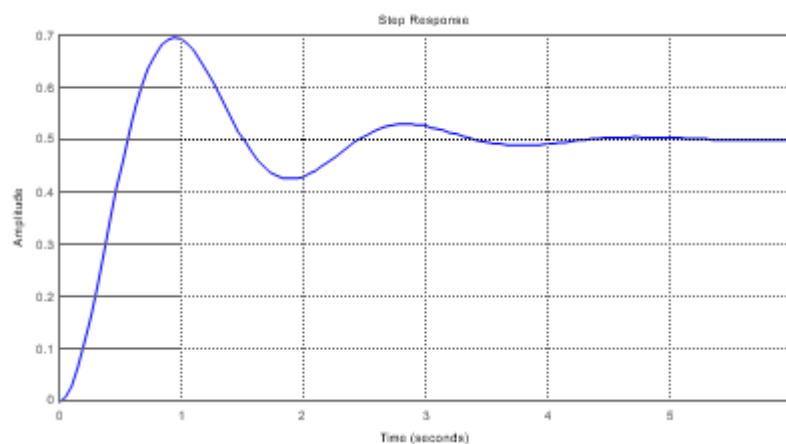
$$\frac{6}{s^2 + 2s + 12}$$

$$a = \begin{bmatrix} -2 & -12 \\ 1 & 0 \end{bmatrix}$$

$$b = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$c = \begin{bmatrix} 0 & 6 \end{bmatrix}$$

$$d = 0$$



The step response of the T.F is as shown in the following Figure.

Secondly: Find the Block diagram of state space by MATLAB Simulink, as shown in the following figure.

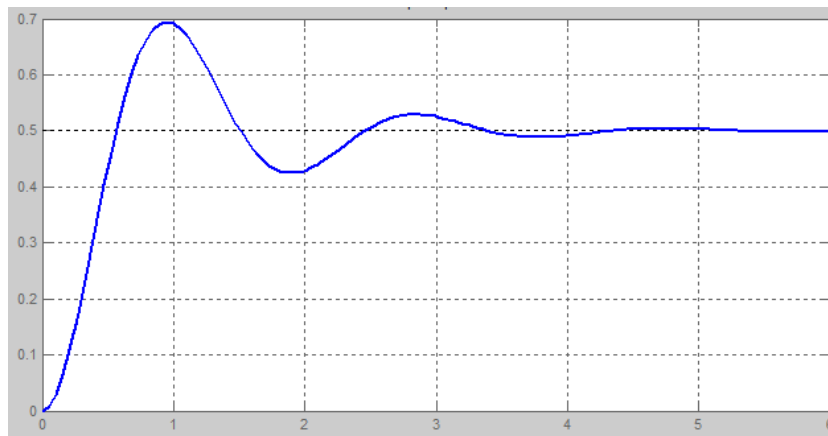
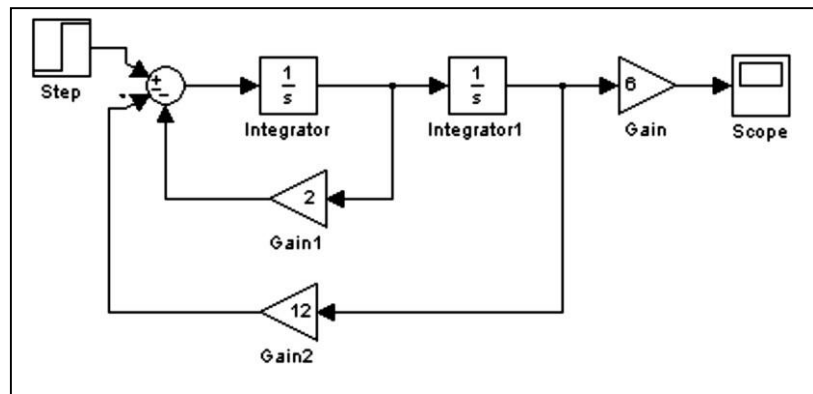
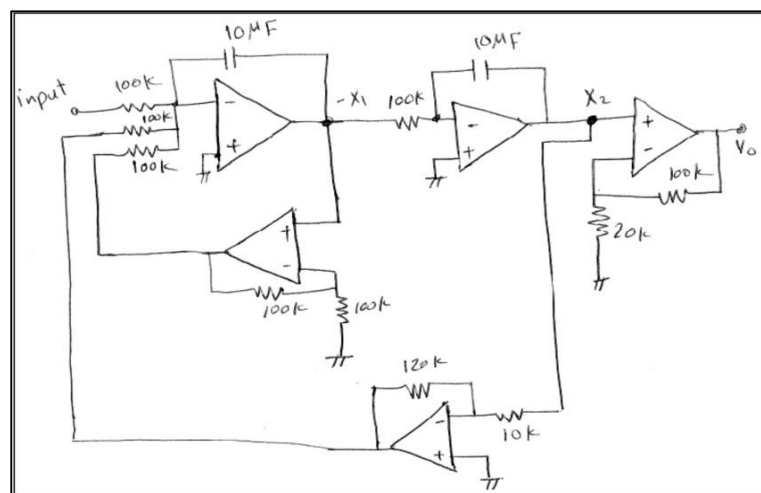


Figure 7 Step response for the block diagram

Thirdly: Convert the block diagram to Op-Amps implementation



Experiments 7

Root-Locus

Objectives:

- To study what is root-locus technique.
- To study how can be plot root-locus using MATLAB and LabVIEW.
- Design of system using root-locus technique.

Introduction:

Root locus, a graphical presentation of the closed-loop poles as a system parameter is varied, is a powerful method of analysis and design for stability and transient response. Feedback control systems are difficult to comprehend from a qualitative point of view, and hence they rely heavily upon mathematics. The root locus covered in this Lab is a graphical technique that gives us the qualitative description of a control system's performance that we are looking for and also serves as a powerful quantitative tool that yields more information than the methods already discussed.

The root locus can be used to describe qualitatively the performance of a system as various parameters are changed. For example, the effect of varying gain upon percent overshoot, settling time, and peak time can be vividly displayed. The qualitative description can then be verified with quantitative analysis.

- Consider a feedback system that has one parameter (gain) $K > 0$ to be designed.

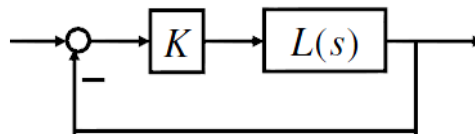


Figure 1 Feedback System

- **Root locus** graphically shows how poles of CL system varies as K varies from 0 to infinity.

○ Root-Locus Technique:

As system parameter k varies over a continuous range of values, the root locus diagram shows the trajectories of the closed-loop poles of the feedback system. Typically, the root locus method is used to tune the loop gain of a SISO control system by specifying a designed set of closed-loop pole locations.

Then, Root Locus technique produces a plot that shows the locations of poles of a closed loop system on the S-Plane as K varies and from this plot we can choose the suitable K that meet our specification conditions.

For example: Below we see two plots of root locus. The lines are represent the locations of the poles as K vary.

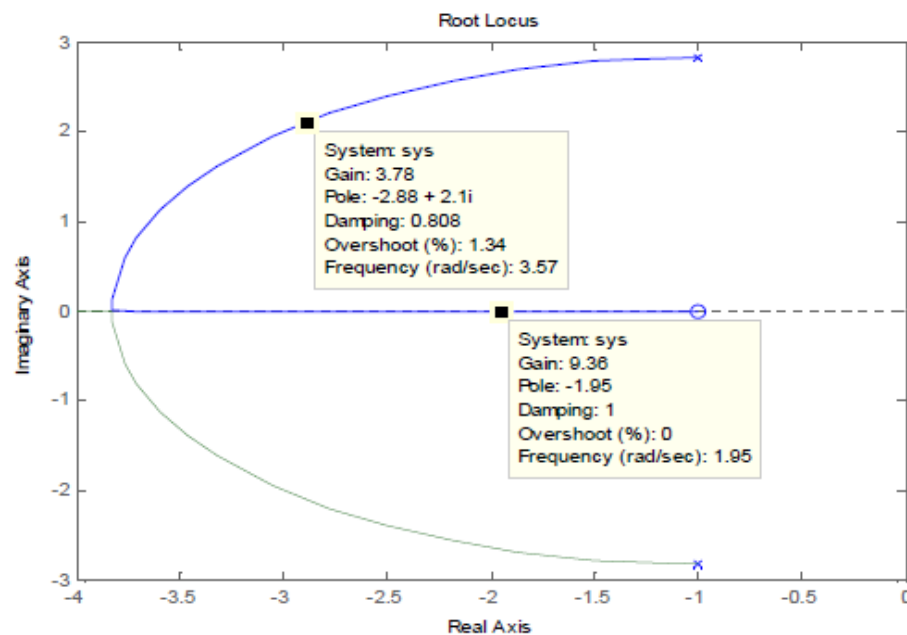


Figure 2 Plots of Root Locus.

As we note in the root locus graph, implemented by Mat lab, the plot tell us that when the gain $K = 3.78$ the response will have overshooting 1.34% and damping ratio 0.8 and when $K = 9.36$ the response will have no overshooting. Also the locus say that all values of K will not make the system unstable and will remain in the stable region.

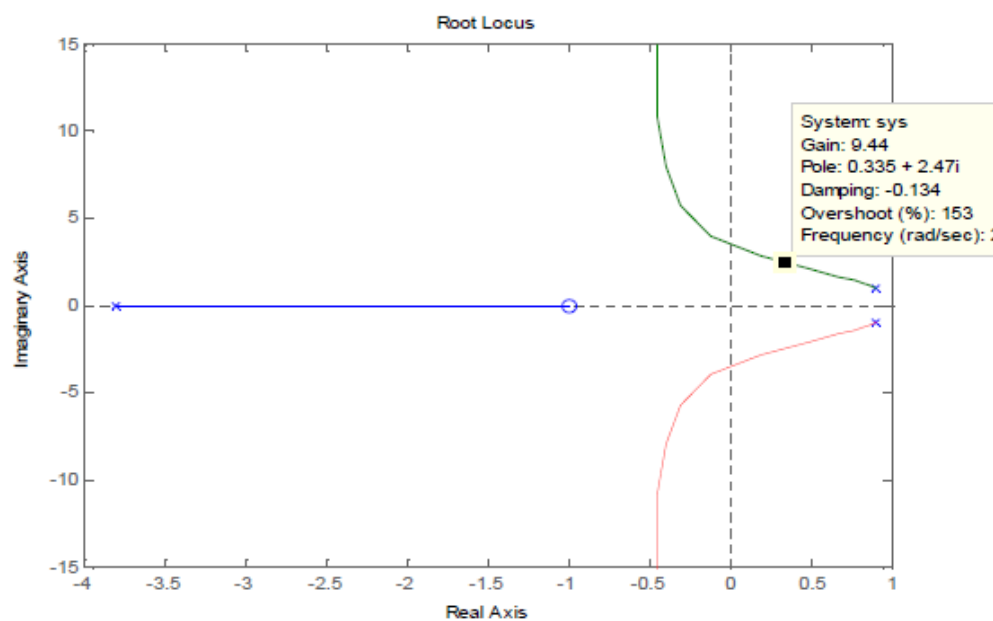


Figure 3 Using Proportional Controller

When the gain $K = 9.44$ the system will be unstable because the poles become in RHP , so we can conclude that the proportional controller may drive the system from the unstable mode to a stable one and vice versa.

○ How to plot root-locus using Mat lab

First thing you need to enter the open loop transfer function or the equivalent state space then in the command window write rlocus (sys) like the following example.

$$L(s) = \frac{s+5}{s^2+5s+6}$$

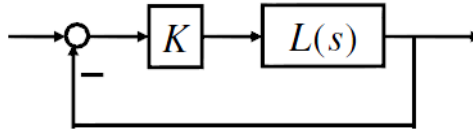


Figure 4 Feedback system with controller

```
num=[1 5]
den=[1 5 6]
sys=tf(num,den)
rlocus(sys)
```

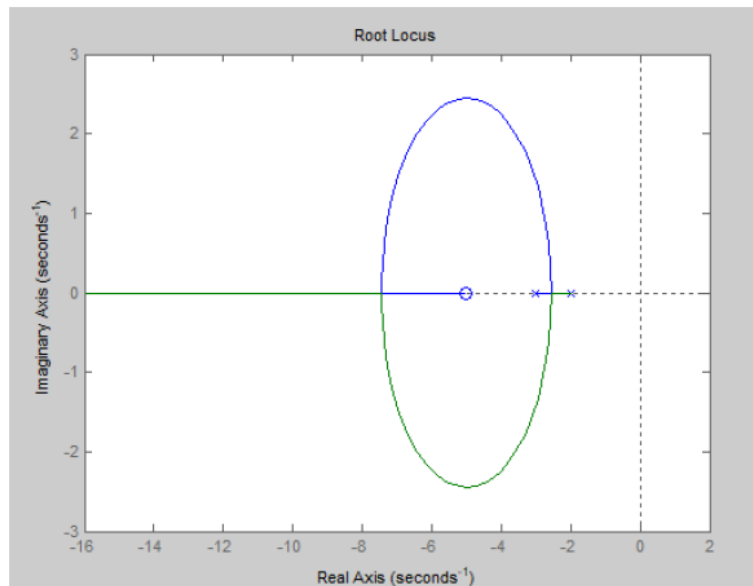


Figure 5 Root Locus using mat lab

○ How to plot root-locus using LabVIEW

In order to plot the root locus using LabVIEW you need to enter the system transfer function first like the previous part of this experiment the go to **control design >> dynamic characteristic** and choose **root locus**.

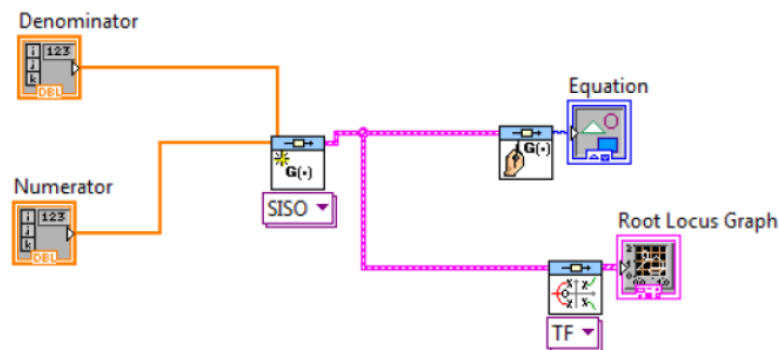


Figure 6 Front Panel

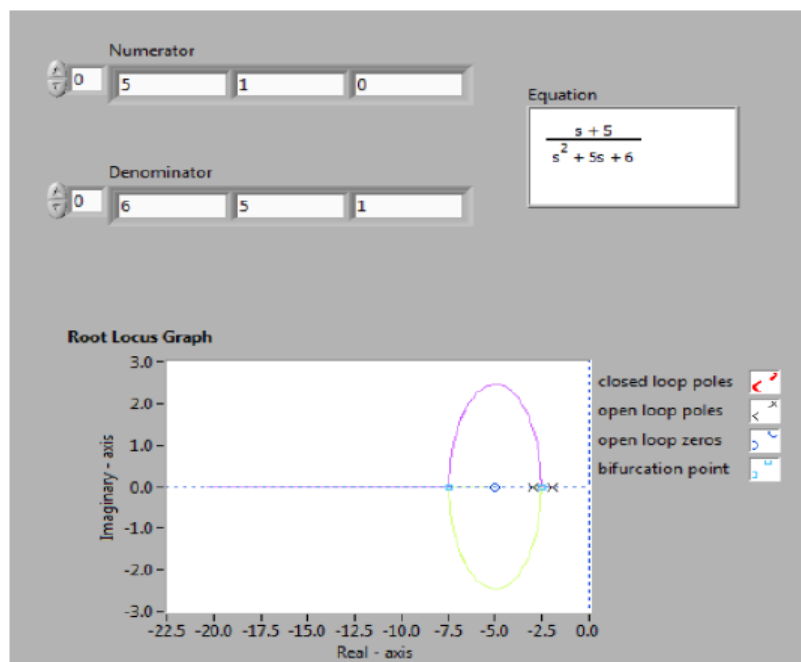


Figure 7 Block Diagram

○ Extra About root locus :

We knew that root locus plot the poles for an closed loop transfer function that will be controlled by Proportional controller and the system must have unity feedback path as in figure below.

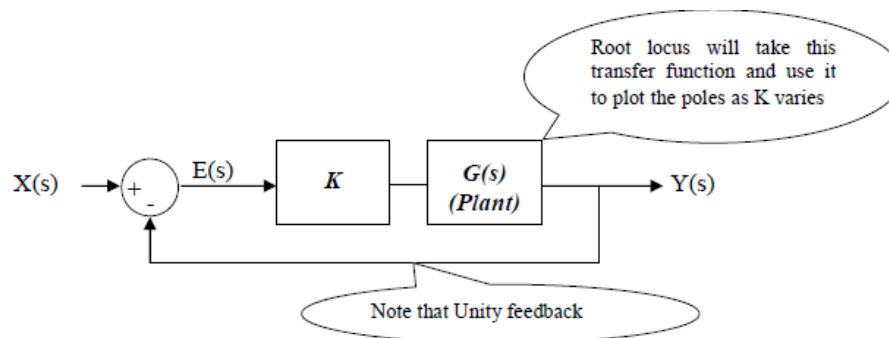


Figure 8 Unity Feedback System

But if the system does not has unity feedback as in figure below, so we should convert it to another has unity feedback by the solution below.

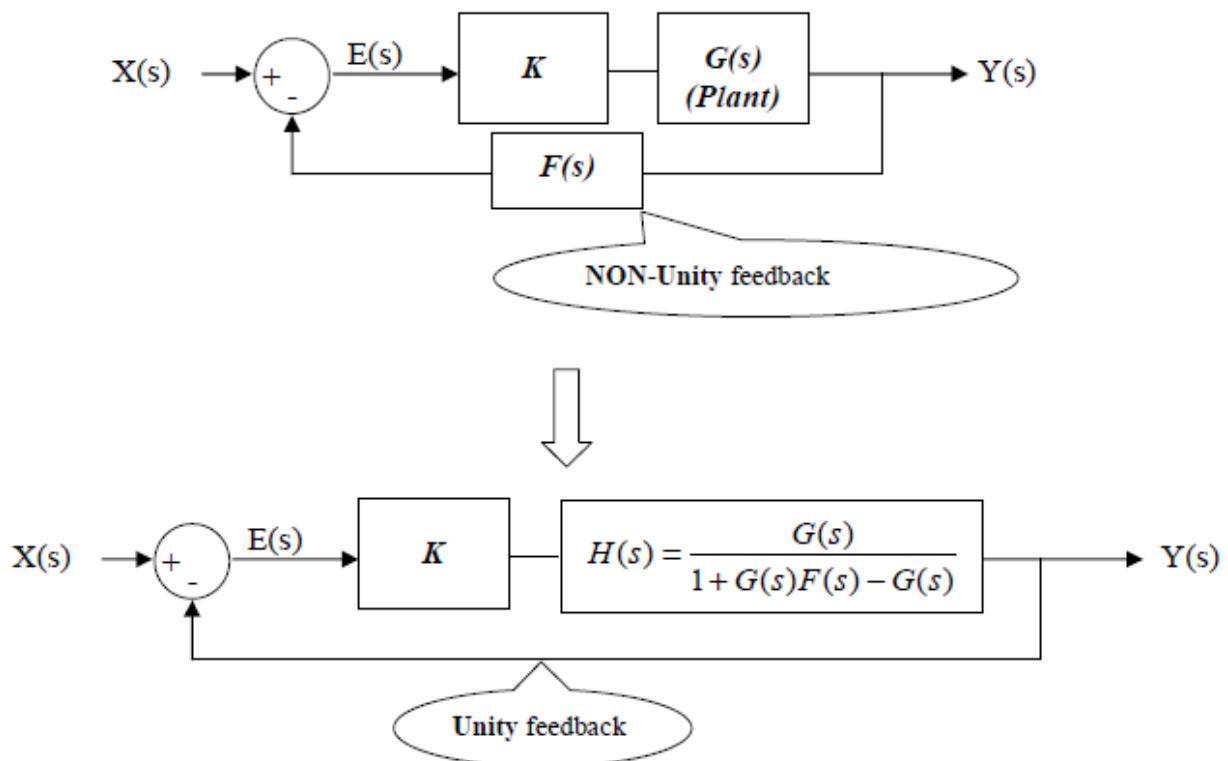


Figure 9 Non-unity Feedback System

Example 1: Using Mat lab plot the root locus for the T.F and design K controller to get 5% O.S

$$G(s) = \frac{3}{s(s+8)}$$

MATLAB Code:

```
num=[3]
den=[1 8 0]
sys=tf(num,den)
rlocus(sys)
```

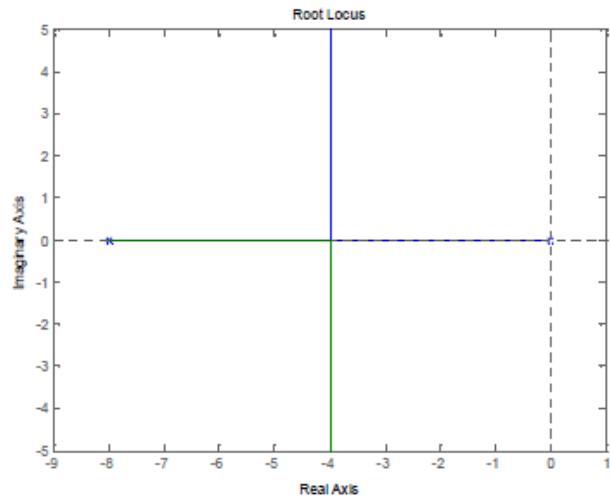


Figure 10 Root locus for EX1

From the plot of root locus we find that if $K = 11.2$ then the O.S will be 5 %
To verify draw the step response when $K = 11.2$:

MATLAB Code:

```
num=[3]
den=[1 8 0]
sys=tf(num,den)
k=11.2
sysf=feedback(k*sys,1)
step (sysf)
```

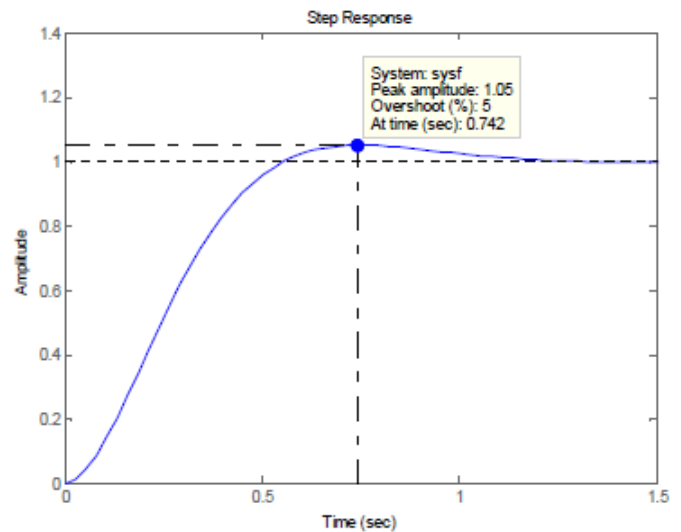


Figure 11 Root locus with Controller

Example 2: For the following T.F Plot the root locus of this system and obtain the value range of the proportional gain to make the system:

1. Stable.
2. Unstable.
3. Margin

$$G(s) = \frac{s+2}{s(s^2+0.6s+4)}$$

Mat lab Code:

From the Root Locus, we find that the system is:

- 1- Stable for $K < 1.77$
- 2- Margin stable for $K = 1.77$
- 3- Unstable for $K > 1.77$

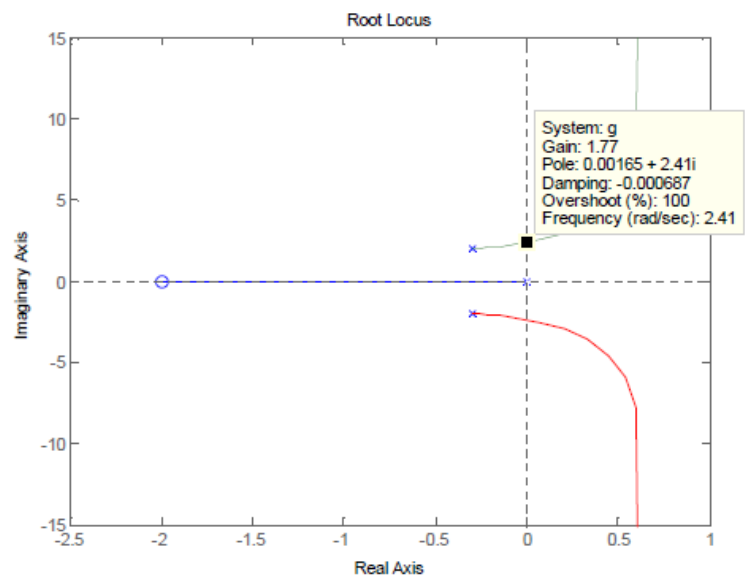


Figure 12 Root Locus for EX2

Experiments 8

Time Response Design

Objectives:

- To study what is the Time Response and how to find steady state error.
- To study Proportional Controller design.
- To study how can be design of the system using MATLAB and LabVIEW.

Introduction:

What is the Time Response?

It is an equation or a plot that describes the behavior of a system and contains much information about it with respect to time response specification as overshooting, settling time, peak time, rise time and steady state error.

Time response is formed by the transient response and the steady state response. Mathematically, we can write the time response $y(t)$ as

$$y(t) = y_t(t) + y_{ss}(t)$$

$y_t(t)$: Transient response

$y_{ss}(t)$: Steady state response

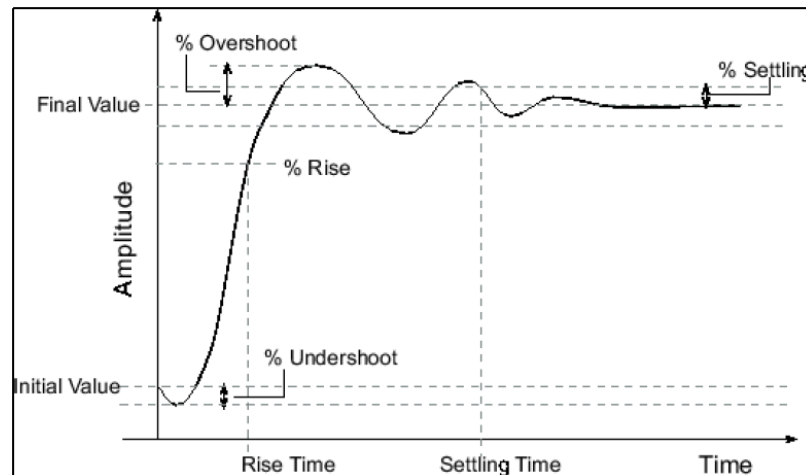


Figure 1 Time Response Specification

Control systems types:

Two types of control system:

1. Open loop system
2. Close loop system

Firstly: Open Loop System:

In the past, electrical control systems were basically manual or what is called an Open-loop System with very few automatic control or feedback features built in to regulate the process variable so as to maintain the desired output level or value.

an Open-loop system, also referred to as non-feedback system, is a type of continuous control system in which the output has no influence or effect on the control action of the input signal. In other words, in an open-loop control system the output is neither measured nor “fed back” for comparison with the input. Therefore, an open-loop system is expected to faithfully follow its input command or set point regardless of the final result.

Also, an open-loop system has no knowledge of the output condition so cannot self-correct any errors it could make when the preset value drifts, even if this results in large deviations from the present value.

The open loop transfer function is a transfer function that represents a system and relates the output $Y(s)$ to the input $X(s)$ as a ratio.

$$G(s) = \frac{Y(s)}{X(s)}$$

In Control Theory, the transfer function of the system that will be controlled is called a plant and sometimes denoted as $G(s)$.

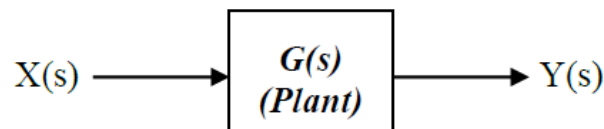


Figure 2 open loop system

For example of the open loop system, an electric clothes dryer. Depending upon the amount of clothes or how wet they are, a user or operator would set a timer (controller) to say 30 minutes and at the end of the 30 minutes the drier will automatically stop and turn-off even if the clothes were still wet or damp.

In this case, the control action is the manual operator assessing the wetness of the clothes and setting the process (the drier) accordingly.

So in this example, the clothes dryer would be an open-loop system as it does not monitor or measure the condition of the output signal, which is the dryness of the clothes. Then the accuracy of the drying process, or success of drying the clothes will depend on the experience of the user (operator).

Secondly: Closed Loop System

A **Closed-loop Control System**, also known as a *feedback control system* is a control system which uses the concept of an open loop system as its forward path but has one or more feedback loops (hence its name) or paths between its output and its input. The reference to “feedback”, simply means that some portion of the output is returned “back” to the input to form part of the systems excitation.

Closed-loop systems are designed to automatically achieve and maintain the desired output condition by comparing it with the actual condition. It does this by generating an error signal which is the difference between the output and the reference input. In other words, a “closed-loop system” is a fully automatic control system in which its control action being dependent on the output in some way.

For example of the closed loop system, consider our electric clothes dryer from the previous *Open-loop tutorial*. Suppose we used a sensor or transducer (input device) to continually monitor the temperature or dryness of the clothes and feed a signal relating to the dryness back to the controller.

The closed loop system with unity feedback.

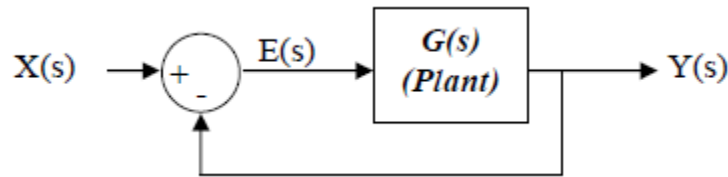


Figure 3 Unity Feedback System

Where $E(s) = \mathcal{L}\{e(t)\}$: the error signal between the input and output.

$$G(s) = \frac{Y(s)}{X(s)}$$

$$E(s) = X(s) - Y(s)$$

$$H(s) = \frac{Y(s)}{X(s)} = \frac{G(s)}{1 + G(s)}$$

○ Proportional Controller:

The closed loop system overcomes the disadvantage that existed in the open loop systems.

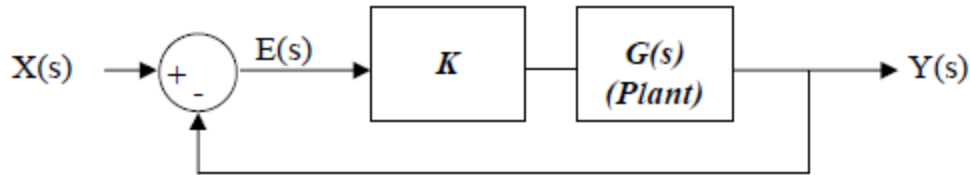


Figure 4 Closed Loop System

Where $E(s) = \mathcal{L}\{e(t)\}$: the error signal between the input and output.

$$G(s) = \frac{Y(s)}{X(s)}$$

$$E(s) = X(s) - Y(s)$$

$$H(s) = \frac{Y(s)}{X(s)} = \frac{K G(s)}{1 + K G(s)}$$

In this case K is called Proportional Gain or Proportional Controller.

The effect of proportional gain on the system specifications:

In this part we will be study the effect of proportional gain on the Overshoot, settling time, rise time, peak time and steady state error then study how can be choose the best design when change the value of K, we will make it by MATLAB and LabVIEW.

$$\text{Let } G(s) = \frac{1.5}{s^2 + 14s + 40.02}$$

Then the Closed loop transfer function is:

$$H(s) = \frac{Y(s)}{X(s)} = \frac{K G(s)}{1 + K G(s)} = \frac{1.5K}{s^2 + 14s + (40.02 + 1.5K)}$$

Now, let us assume some values of K and plot the step response and analyze the changes in the response specifications.

$$K = 0.1, 1, 10, 100, 1000$$

First method: By MATLAB Code

```
num = 1.5;
den = [1 14 40.02];
G = tf(num,den);
K = input('K = ');
sys = feedback(K*G,1);
step(sys)
stepinfo(sys)
```

➤ At $K = 0.1$

```
RiseTime: 0.6131
SettlingTime: 1.0995
SettlingMin: 0.0034
SettlingMax: 0.0037
Overshoot: 0
Undershoot: 0
Peak: 0.0037
PeakTime: 2.0407
```

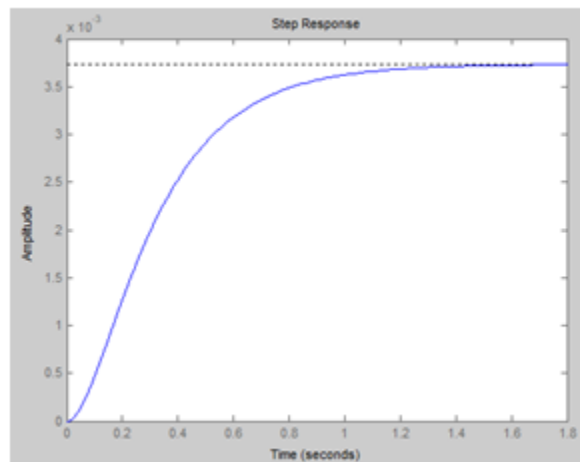


Figure 5 Step Response for $K=0.1$

➤ At $K = 1$

```
RiseTime: 0.5887
SettlingTime: 1.0521
SettlingMin: 0.0326
SettlingMax: 0.0361
Overshoot: 0
Undershoot: 0
Peak: 0.0361
PeakTime: 1.9311
```

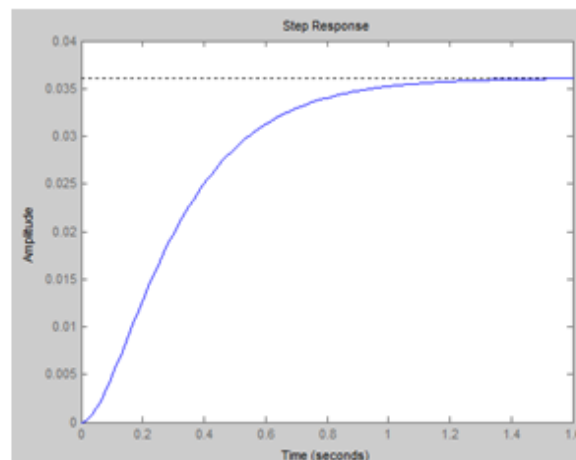


Figure 6 Step Response for $K=1$

➤ At $K = 10$

```
RiseTime: 0.4159
SettlingTime: 0.6996
SettlingMin: 0.2454
SettlingMax: 0.2727
Overshoot: 0.0127
Undershoot: 0
Peak: 0.2727
PeakTime: 1.2665
```

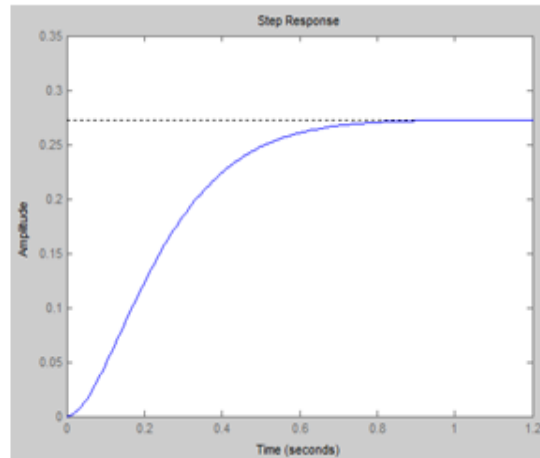


Figure 7 Step Response for $K=10$

➤ At $K = 100$

```
RiseTime: 0.1201
SettlingTime: 0.5798
SettlingMin: 0.7186
SettlingMax: 0.9132
Overshoot: 15.6869
Undershoot: 0
Peak: 0.9132
PeakTime: 0.2623
```

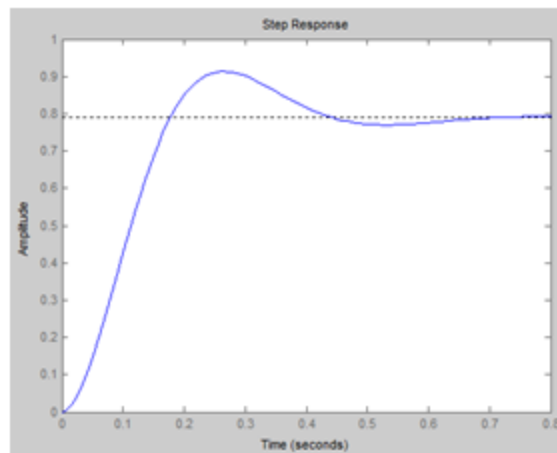


Figure 8 Step Response for $K=100$

➤ At $K = 1000$

```

RiseTime: 0.0307
SettlingTime: 0.5128
SettlingMin: 0.6633
SettlingMax: 1.5246
Overshoot: 56.5298
Undershoot: 0
Peak: 1.5246
PeakTime: 0.0824

```

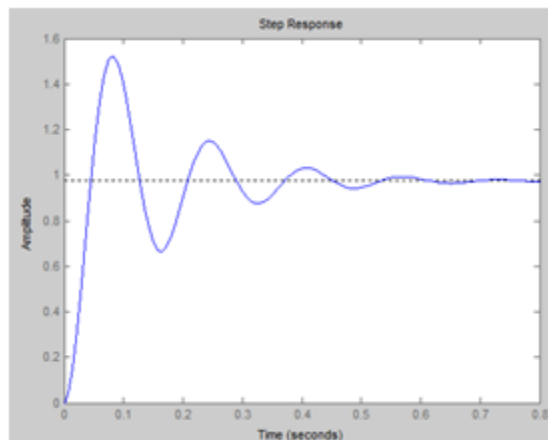


Figure 9 Step Response for $K=1000$

	$K = 0.1$	$K = 1$	$K = 10$	$K = 100$	$K = 1000$
T_s					
T_r					
OS%					
Y_{ss}					
S.S.E					

Conclusion:

The previous results say that the proportional controller has an effect on overshooting and settling time such that increasing the K gain will increase the overshooting and decrease the settling time and rise time. In the other hand decreasing the K gain will decrease the overshooting and increase the settling time and rise time.

Second method: By LabVIEW

- First step built the structure on the block diagram as shown in figure below.

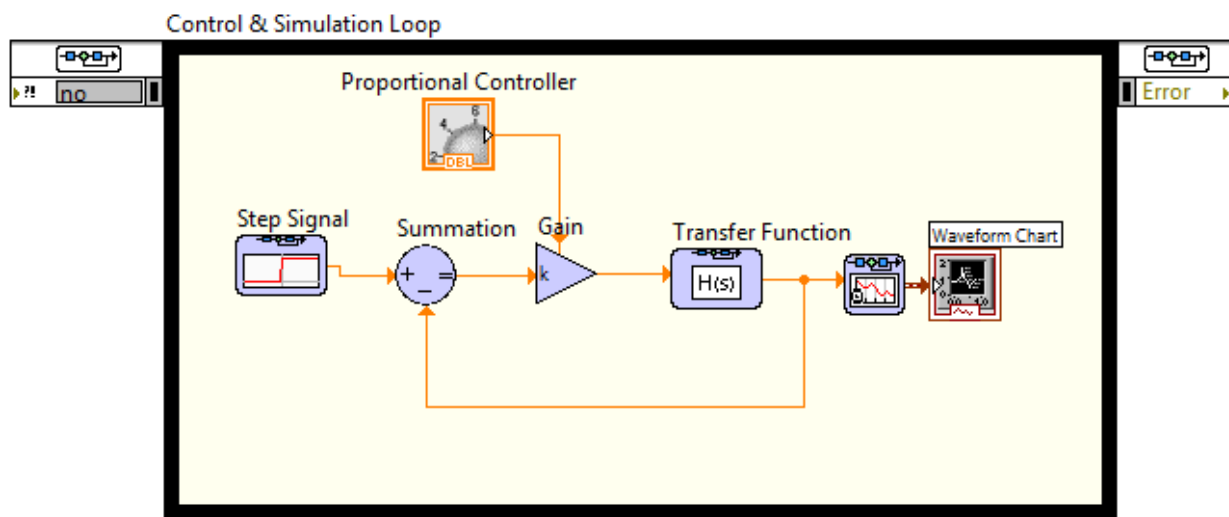


Figure 10 Block Diagram

- Put step time of step signal equal 0 as shown in figure below.

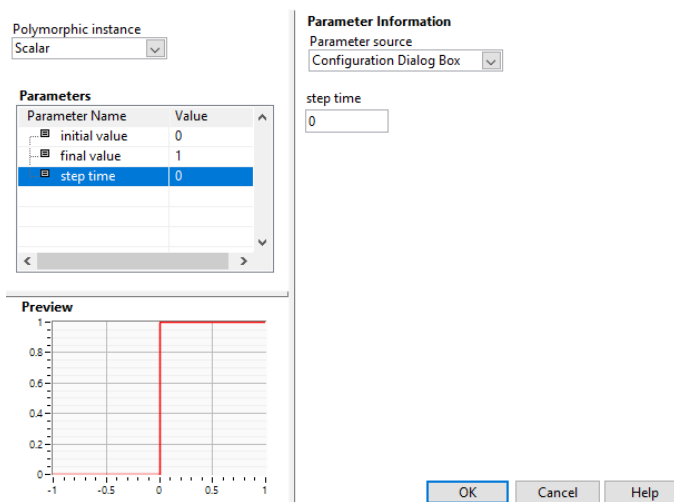


Figure 11 Step Time

- Change parameter source of the gain (K) from configuration dialog box to terminal as shown in figure.

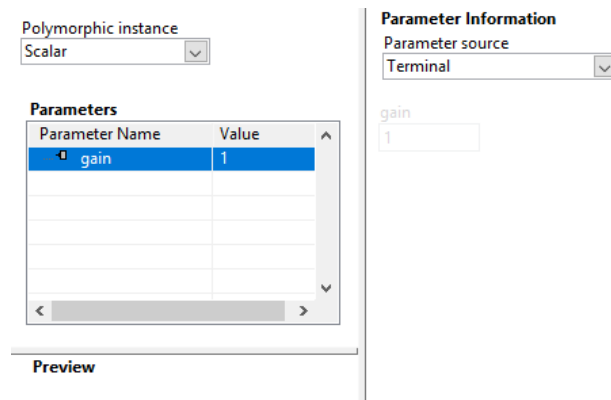


Figure 12 Specification

- Change the numerator and denominator of transfer function.

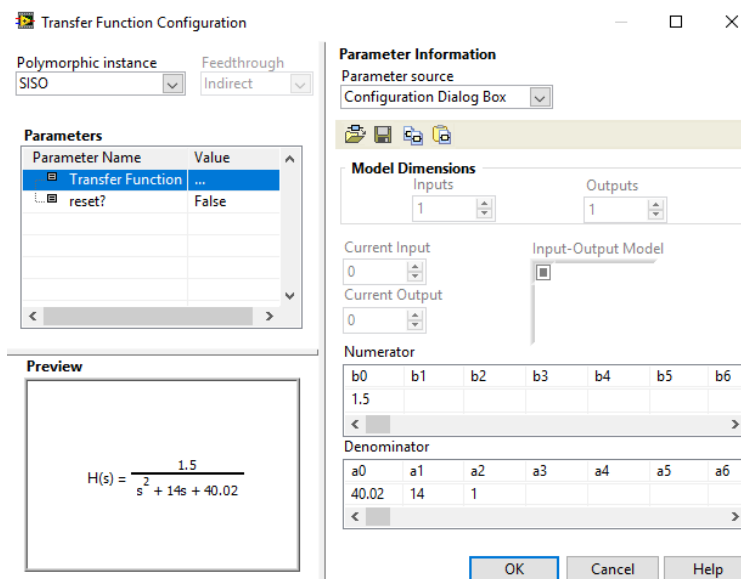


Figure 13 Numerator and denominator

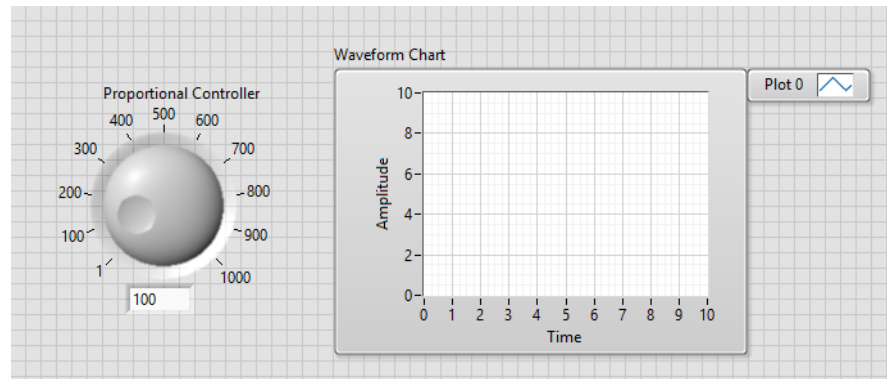
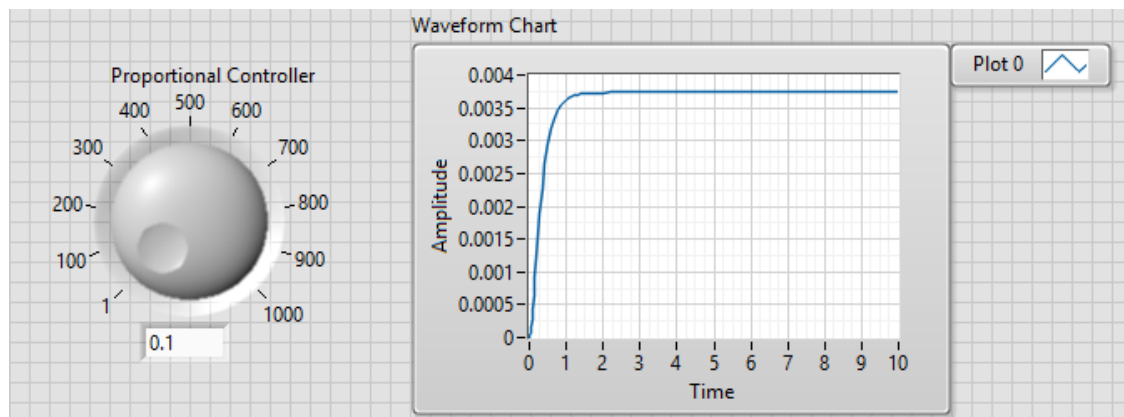


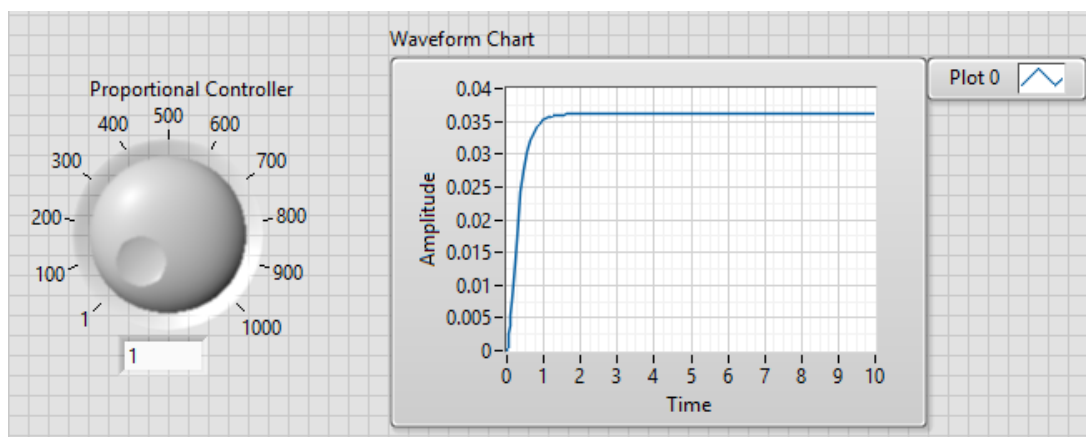
Figure 14 Front Panel

- Now we will vary in the value of proportional gain and observe the effect of increasing it on the step response of the system $G(s)$.

➤ At $K = 0.1$

Figure 15 Step Response for $K=0.1$

➤ At $K = 1$

Figure 16 Step Response for $K=1$

➤ At $K = 10$

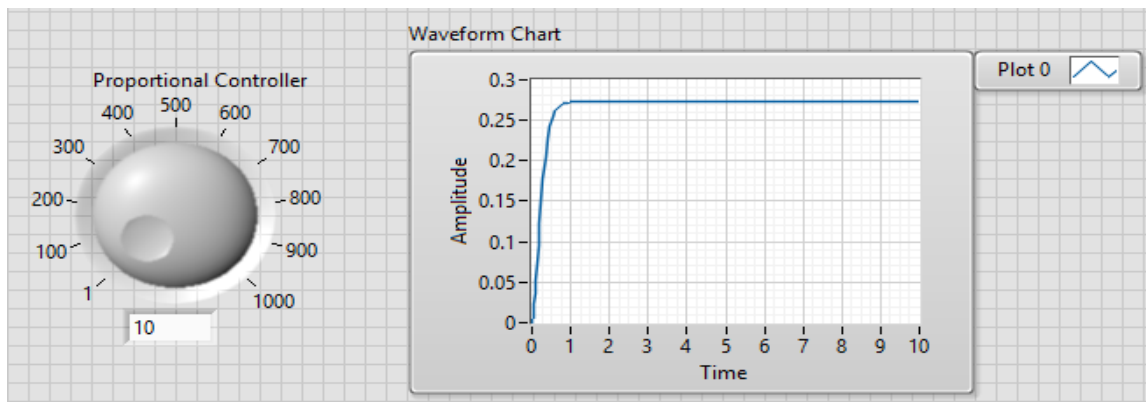


Figure 17 Step Response for $K=10$.

➤ $K = 100$

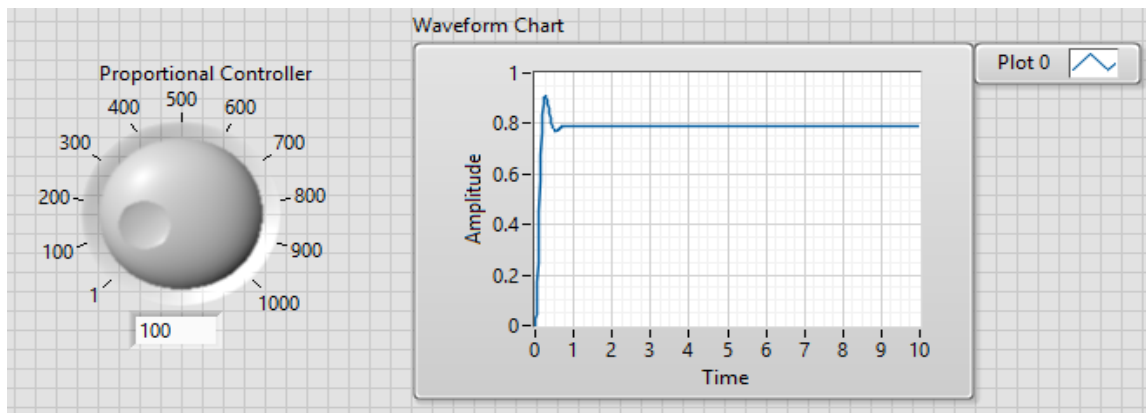


Figure 18 Step Response for $K=100$

➤ $K = 1000$

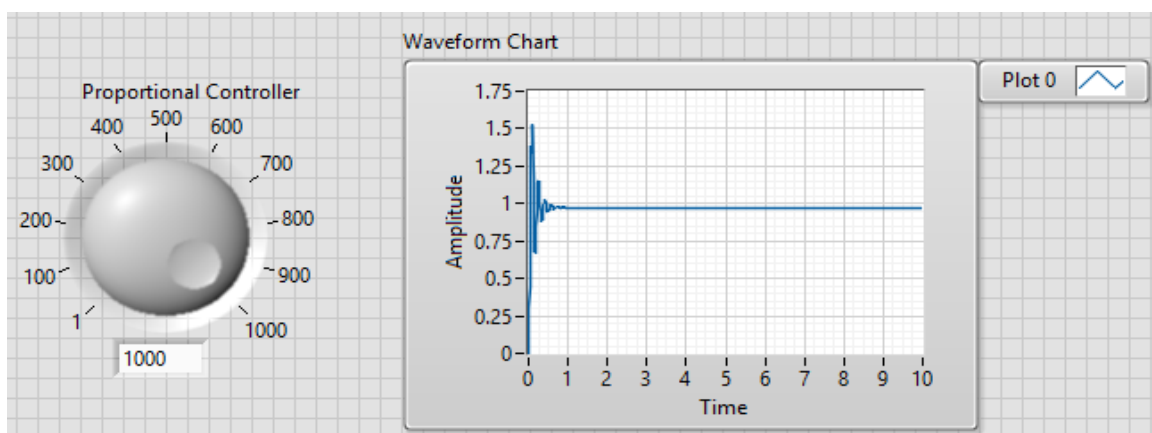


Figure19 Step Response for $K=1000$

Experiments 9

Supervisory Control and Data Acquisition (DSC)

Objectives:

- To study what is the SCADA (supervisory control and data acquisition)
- To study how can be design of the SCADA system in LabVIEW by insert Supervisory Control and Data Acquisition (DSC) library to LabVIEW.

Introduction:

The conventional methods for plant monitoring may be uneconomical if applied to power plant due to high cost, so integrated automation and control has become the solution for making it efficient and cost effective. Normally, an automated system improves the system efficiency, plant monitoring, productivity and the operation management of the plant.

The main objective of a supervisory system is to give the means to the human operator to control and to command a highly automated process. So, the supervision of industrial processes includes a set of tasks aimed at controlling a process and supervising its operation.

SCADA (supervisory control and data acquisition) generally refers to industrial control systems, computer systems that monitor and control industrial, laboratory, infrastructure or facility-based processes.

The LabVIEW Data logging and Supervisory Control (DSC) Module is add-on software for the LabVIEW programming environment. You can use this module to develop a distributed monitoring and control system with tags ranging from a few dozen to tens of thousands. It includes tools for logging data to a networked historical database, tracking real-time and historical trends, managing alarms and events, networking LabVIEW Real-Time targets and OPC devices into one complete system, and adding security to user interfaces.

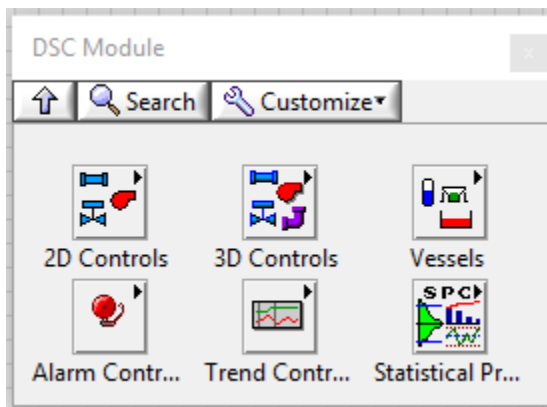


Figure 1: DSC Module in Front Panel

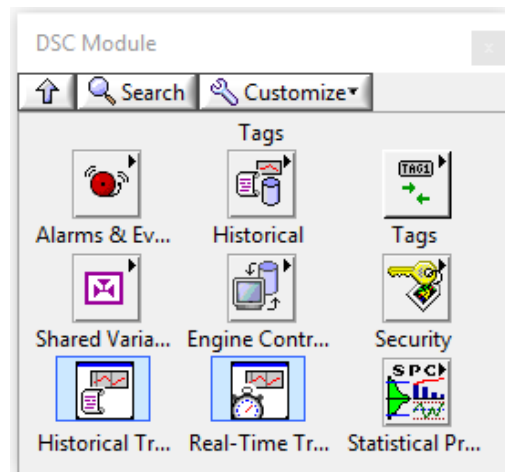


Figure 2: DSC Module in Block Diagram

Example1: Make Transmission belt that transfer of boxes from point to another point as shown in figure and control of moving box by toggle switch and push button to reset box to zero.

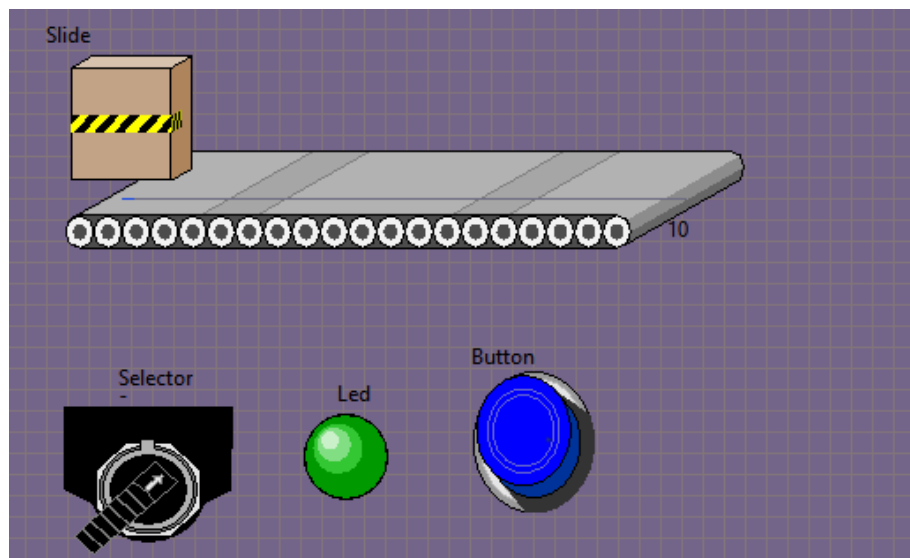


Figure 3 Front Panel

1- Selector

- Open blank project then from my computer >> new control.

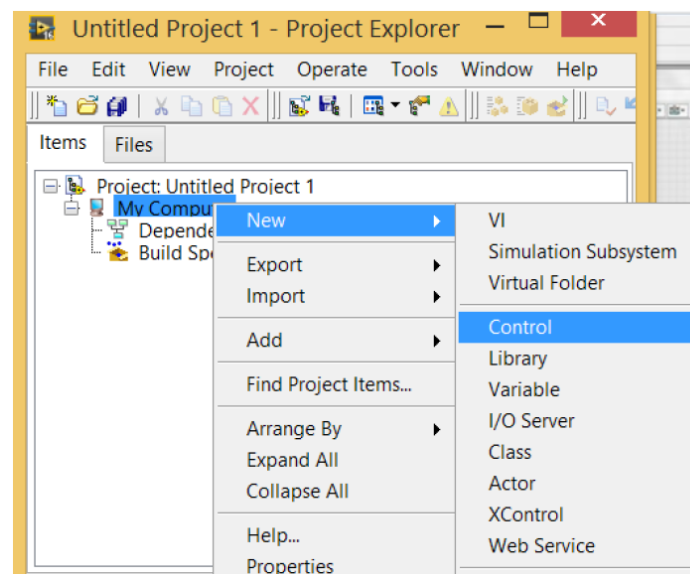


Figure 4 To create Control

- Add Toggle Switch

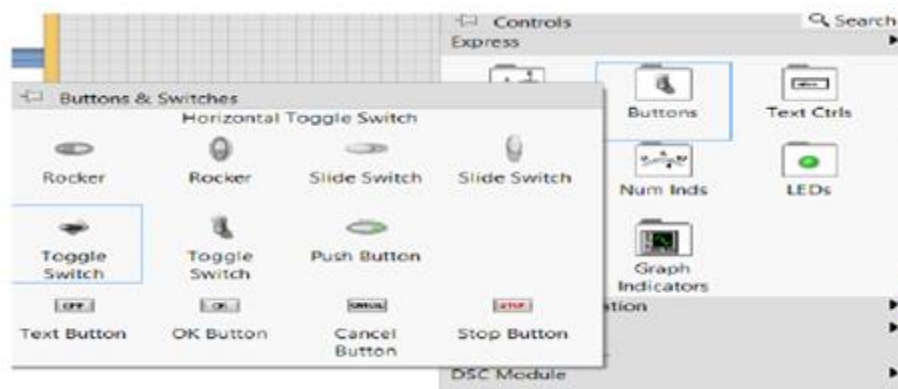


Figure 5 Toggle Switch

- From tools choose DSC module >>Image Navigator.

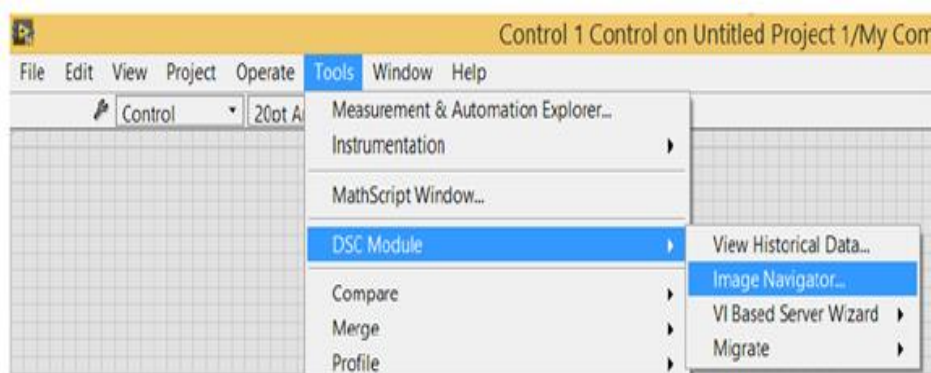


Figure 6 Image Navigator.

- Choose from 3D Pushbutton>>Selector switch left and Selector switch Right.

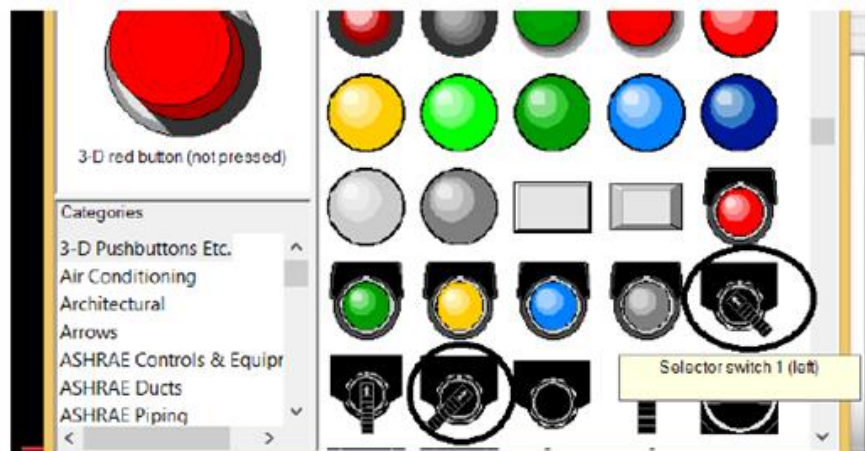


Figure 7 Adding Selector

- Press(ctrl+x) then Choose from Bar Change to Customize Mode.
- Go to toggle switch ,click right and choose import picture from clipboard.

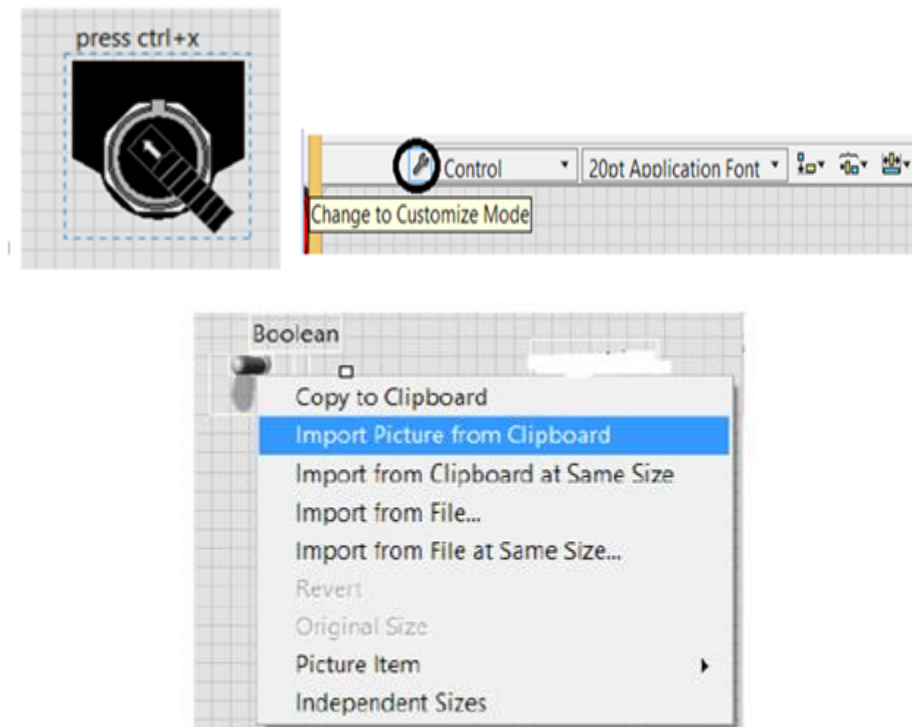


Figure 8 Customized Mode

- Then choose from Bar (Change to Edit Mode).

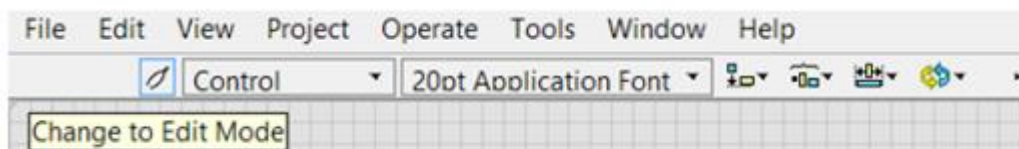


Figure 9 Edit Mode

- Now, go to toggle and click left button on it(selector will disappear)
- Then you must make these same steps for another selector switch.
- Save your work
- Led, button and slide are the same idea of selector.

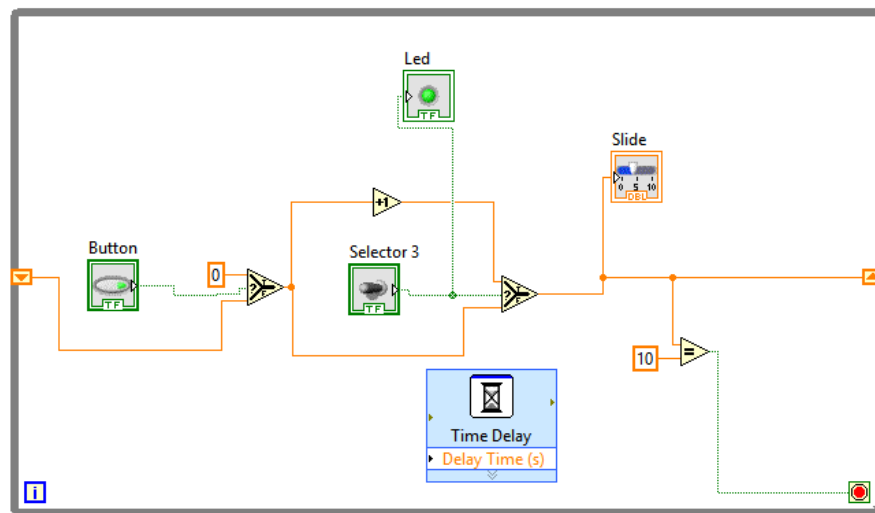


Figure 10 Final Front Panel

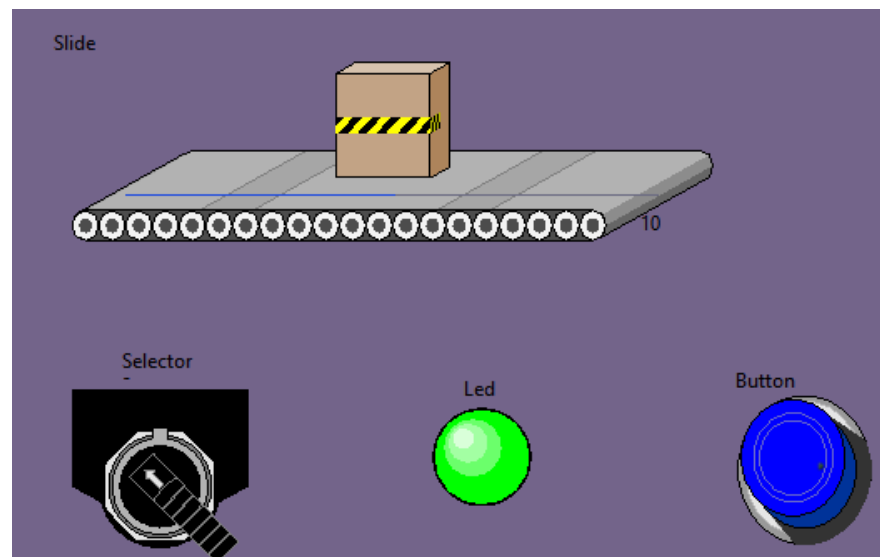


Figure 11 Final Block Diagram

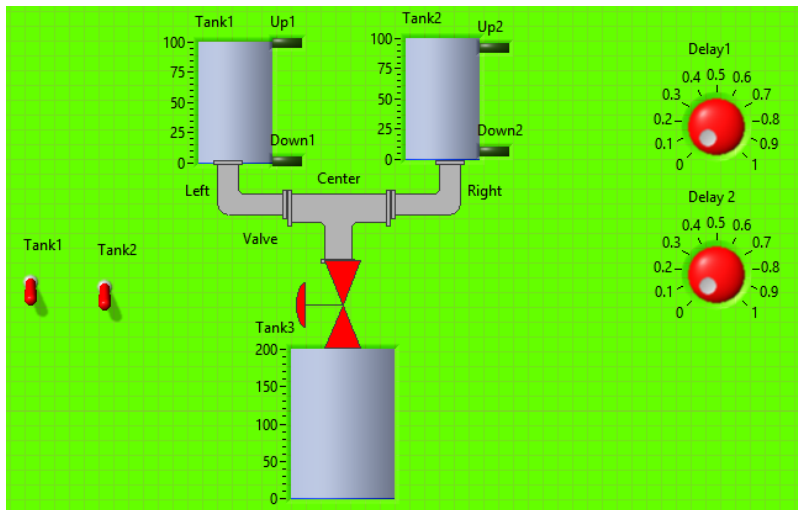
Exercise : Explain what this program does?

Figure 12 Front Panel

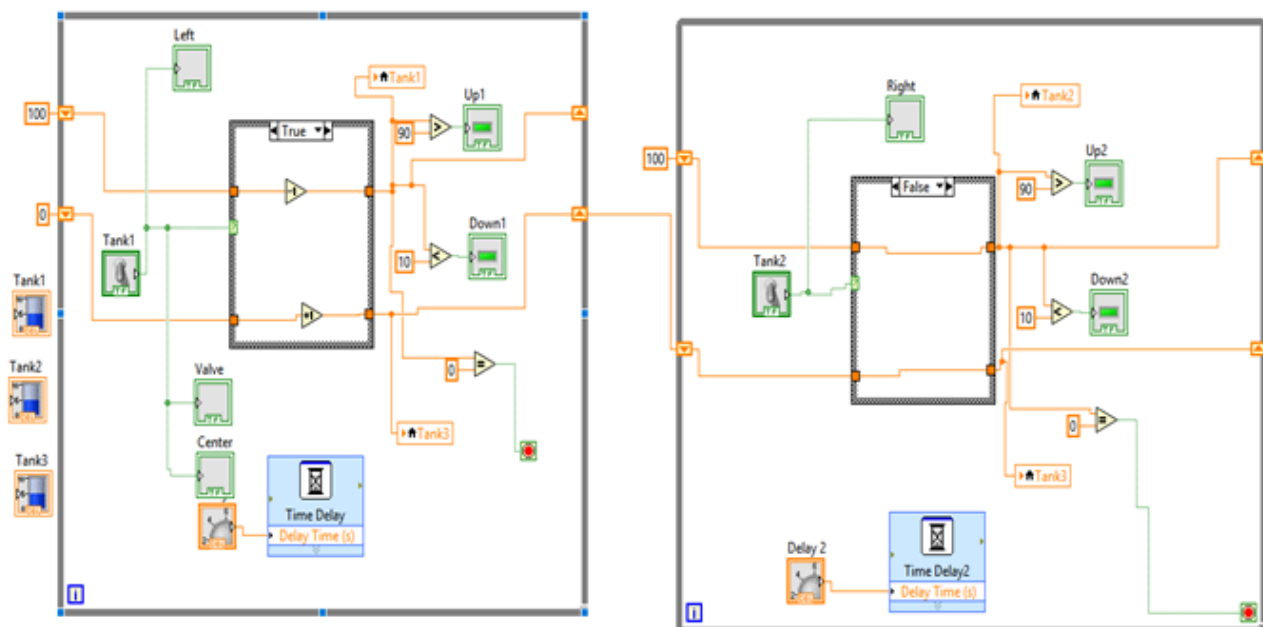


Figure 13 Block Diagram