Laboratory Manual

Control Technology

Electrical Engineering & Technology

Session: 2017-2021

Subject Code: ET-323

Program: Department of Electrical Engineering & Technology

Department of Engineering & Technology

University College of Engineering & Technology

University of Sargodha

LIST OF EXPERIMENTS:

LAB SESSION 01:

MATLAB FOR CONTROL SYSTEMS

OBJECTIVES:

This lab provides introduction to MATLAB and helps students developing following skills:

- Arithmetic Operations in MATLAB
- Use of Complex Numbers
- Array Indexing and addressing
- Matrices
- Control Flow in MATLAB
- Plotting
- Programming in MATLAB

THEORY:

WHAT IS MATLAB?

MATLAB stands for **MAT**RIX **LAB**ORATORY. MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

- Math and computation
- Algorithm development
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non-interactive language such as C or FORTRAN.

MATLAB features a family of application-specific solutions called toolboxes. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

This lab includes following sections:

1. MATLAB Basics
    A. Definition of Variables
    B. Definition of Matrices
    C. General Information
    D. M-files
2. Plotting
3. Loading and Saving Data

MATLAB BASICS

MATLAB is started by clicking the mouse on the appropriate icon and is ended by typing exit or by using the menu option. After each MATLAB command, the "return" or "enter" key must be depressed.

DEFINITION OF VARIABLES

Variables are assigned numerical values by typing the expression directly, for example, typing

```
a = 1+2
```

Yields: a = 3

The answer will not be displayed when a semicolon is put at the end of an expression, for example type

```
a = 1+2;
```

MATLAB utilizes the following arithmetic operators:

+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Power operator

A variable can be assigned using a formula that utilizes these operators and either numbers or previously defined variables. For example, since 'a' was defined previously, the following expression is valid

```
b = 2*a;
```

To determine the value of a previously defined quantity, type the quantity by itself:

b

yields: b = 6

There are several predefined variables which can be used at any time, in the same manner as user- defined variables:

i	sqrt(-1)
j	sqrt(-1)
pi	3.1416..

For example,

y= 2*(1+4*j)

yields: y= 2.0000 + 8.0000i

There are also a number of predefined functions that can be used when defining a variable.

Some common functions that are used in this text are:

abs	magnitude of a number (absolute value for real numbers)
angle	angle of a complex number, in radians
cos	cosine function, assumes argument is in radians
sin	cosine function, assumes argument is in radians
exp	exponential function

For example, with y defined as above,

c = abs(y)

yields: c = 8.2462

c = angle(y)

yields: c = 1.3258

With a=3 as defined previously,

```
c = cos(a)

yields: c =  -0.9900

c = exp(a)

yields: c =  20.0855
```

DEFINITION OF MATRICES

MATLAB is based on matrix and vector algebra; even scalars are treated as 1x1 matrices. Therefore, vector and matrix operations are as simple as common calculator operations. Vectors can be defined in two ways. The first method is used for arbitrary elements:

```
v = [1 3 5 7];
```

V creates a 1x4 vector with elements 1, 3, 5 and 7. Note that commas could have been used in place of spaces to separate the elements. Additional elements can be added to the vector:

```
v(5) = 8;
```

yields the vector v = [1 3 5 7 8]. Previously defined vectors can be used to define a new vector. For example, with v defined above

```
a = [9 10];

b = [v a];
```

creates the vector b = [1 3 5 7 8 9 10].

The second method is used for creating vectors with equally spaced elements:

```
t = 0:0.1:10;
```

creates a 1x101 vector with the elements 0, 0.1, 0.2, 0.3,...,10. Note that the middle number defines the increment. If only two numbers are given, then the increment is set to a default of 1:

```
k = 0:10;
```

creates a 1x11 vector with the elements 0, 1, 2, ..., 10.

A particular element of a matrix can be assigned:

```
M(1,2) = 5;
```

place the number 5 in the first row, second column.

Operations and functions that were defined for scalars in the previous section can also be used on vectors and matrices. For example,

```
a = [1 2 3];

b = [4 5 6];

c = a + b

yields: c =5 7 9
```

Functions are applied element by element. For example,

```
t = 0:10;

x = cos(2*t);
```

creates a vector x with elements equal to cos(2t) for t = 0, 1, 2, ..., 10.

Operations that need to be performed element-by-element can be accomplished by preceding the operation by a ".". For example, to obtain a vector x that contains the elements of x(t) = tcos(t) at specific points in time, you cannot simply multiply the vector t with the vector cos(t). Instead you multiply their elements together by adding a '.' In between:

```
t = 0:10;

x = t.*cos(t);
```

GENERAL INFORMATION

- MATLAB is case sensitive so "a" and "A" are two different names.
- Comment statements are preceded by a "%".
- On-line help for MATLAB can be reached by typing help for the full menu or typing help followed by a particular function name or M-file name. For example,*' help sin'* gives help on the sine function.
- The number of digits displayed is not related to the accuracy. To change the format of the display, type 'format short e' for scientific notation with 5 decimal places, 'format long e' for scientific notation with 15 significant decimal places and 'format bank' for placing two significant digits to the right of the decimal.
- The commands 'who' and 'whos' give the names of the variables that have been defined in the workspace.

- The command ‘length(x)’ returns the length of a vector x and ‘size(x)’ returns the dimension of the matrix x.

M-FILES

M-files are macros of MATLAB commands that are stored as ordinary text files with the extension "m", that is *filename*.m. An M-file can be either a function with input and output variables or a list of commands.

The following describes the use of M-files on a PC version of MATLAB. MATLAB requires that the M-file must be stored either in the working directory or in a directory that is specified in the MATLAB path list. For example, consider using MATLAB on a PC with a user-defined M-file stored in a directory called "\MATLAB\MFILES". Then to access that M-file, either change the working directory by typing cd\matlab\mfiles from within the MATLAB command window or by adding the directory to the path.

MATLAB M-files are most efficient when written in a way that utilizes matrix or vector operations.

Loops and if statements are available, but should be used carefully since they are computationally inefficient. An example of the use of the command ‘for’ is;

```
for k=1:10,

        x(k) = cos(k);

end
```

This creates a 1x10 vector x containing the cosine of the positive integers from 1 to 10. This operation is performed more efficiently with the commands

```
k = 1:10;

x = cos(k);
```

Which utilizes a function of a vector instead of a ‘for’ loop. An ‘if’ statement can be used to define conditional statement. An example is;

```
if (a<=2)

b=1;

elseif (a<=1)

b = 2;

else
```

```
b = 3;

end
```

The allowable comparisons between expressions are >=, <=, <, >, ==, and ~=.

Several of the M-files written for this textbook employ a user-defined variable which is defined with the command input. For example, suppose that you want to run an M-file with different values of a variable T. The following command line within the M-file defines the value:

```
T = input('Input the value of T: ')
```

Whatever comment is between the quotation a mark is displayed to the screen when the M-file is running, and the user must enter an appropriate value.

PLOTTING

Commands covered:

- plot
- xlabel
- ylabel
- title
- grid
- axis
- stem
- subplot

The command most often used for plotting is ‘plot’, which creates linear plots of vectors and matrices; plot(t,y) plots the vector t on the x-axis versus vector y on the y-axis. There are options on the line type and the color of the plot which are obtained using plot(t,y,'option'). The line type options are '-' solid line (default), '--' dashed line, '-.' dot dash line, ':' dotted line. The points in y can be left unconnected and delineated by a variety of symbols: ▪+ . * o x▪. The following colors are available options:

- red (r)
- blue (b)
- green (g)
- white (w)
- black (k)

For example, plot(t,y,'--') uses a dashed line, plot(t,y,'*') uses * at all the points defined in t and y without connecting the points, and plot(t,y,'g') uses a solid green line. The options can also be used together, for example, plot(t,y,'g:') plots a dotted green line.

To plot two or more graphs on the same set of axes, use the command plot(t1,y1,t2,y2), which plots y1 versus t1 and y2 versus t2.

To label your axes and give the plot a title, type;

```
xlabel('time (sec)')

ylabel('step response')

title('My Plot')
```

Finally, add a grid to your plot to make it easier to read. Type ‘grid’.

The problem that you will encounter most often when plotting functions is that MATLAB will scale the axes in a way that is different than you want them to appear. You can easily override the auto scaling of the axes by using the axis command after the plotting command:

```
axis([xmin xmax ymin ymax]);
```

Where xmin, xmax, ymin, and ymax are numbers corresponding to the limits you desire for the axes. To return to the automatic scaling, simply type ‘axis’.

For discrete-time signals, use the command ‘stem’ which plots each point with a small open circle and a straight line. To plot y[k] versus k, type

```
stem(k,y)
```

You can use stem(k,y,'filled') to get circles that are filled in.

To plot more than one graph on the screen, use the command subplot(mnp) which partitions the screen into an mxn grid where p determines the position of the particular graph counting the upper left corner as p=1. For example,

```
Subplot(2,1,1),

Subplot(2,1,2),
```

This has 2 rows, one column, and one plot in each row.

LOADING AND SAVING DATA

When using MATLAB, you may wish to leave the program but save the vectors and matrices you have defined. To save file to the working directory, type save filename where "filename" is name of your choice. To retrieve the data later, type load filename. Or simply save and open the file from the menu.

Help In MATLAB

Display help for MATLAB functions in Command Window

Syntax

- help
- help /
- help functionname
- help toolboxname
- help toolboxname/functionname
- help functionname>subfunctionname
- help classname.methodname
- help classname
- help **syntax**
- t = help('topic')
-

Description

- **help** lists all primary help topics in the Command Window. Each main help topic corresponds to a directory name on the MATLAB search path.
- **help** / lists all operators and special characters, along with their descriptions.
- **help functionname** displays M-file help, which is a brief description and the syntax for functionname, in the Command Window. The output includes a link to doc functionname, which displays the reference page in the Help browser, often providing additional information. Output also includes see also links, which display help in the Command Window for related functions. If functionname is overloaded, that is, appears in multiple directories on the search path, help displays the M-file help for the first functionname found on the search path, and displays a hyperlinked list of the overloaded functions and their directories. If functionname is also the name of a toolbox, help also displays a list of subdirectories and hyperlinked list of functions in the toolbox, as defined in the Contents.m file for the toolbox.
- **help toolboxname** displays the Contents.m file for the specified directory named toolboxname, where Contents.m contains a list and corresponding description of M-files in toolboxname--see the Remarks topic, Creating Contents Files for Your Own M-File Directories. It is not necessary to give the full pathname of the directory; the last component, or the last several components, are sufficient. If toolboxname is also a function name, help also displays the M-file help for the function toolboxname.
- **help toolboxname/functionname** displays the M-file help for the functionname that resides in the toolboxname directory. Use this form to get direct help for an overloaded function.
- **help functionname>subfunctionname** displays the M-file help for subfunctionname that is in functionname.
- **help classname.methodname** displays help for the method methodname of the fully qualified class classname. If you do not know the fully qualified class for the method, use class(obj), where methodname is of the same class as the object obj.
- **help classname** displays help for the fully qualified class classname.

- **help syntax** displays M-file help describing the syntax used in MATLAB commands and functions.
- **t = help('topic')** returns the help text for topic as a string, with each line separated by /n, where topic is any allowable argument for help.

LAB TASKS:

1. What is a variable? Define variables in MATLAB (integers, rational, complex).
2. Apply mathematical operations on the defined variables (addition, multiplication, subtraction, division).
3. Calculate absolute and angle of the complex variable defined.
4. What is a matrix? Define three matrices in MATLAB with dimensions 4x4, 4x3, 2x3.
5. Apply mathematical operations on matrices (addition, subtraction, multiplication).
6. What is a polynomial? How polynomials can be defined in MATLAB. Define three polynomials in MATLAB with highest degree 5, 7, 3.
7. Define any two functions and plot its graph in MATLAB.
8. Change the mark and color, adjust axis, and add axis labels and title.
9. Use the command subplot to plot the graphs of two functions in the same window.

CONCLUSION AND COMMENTS:

LAB SESSION 02:

TRANSFER FUNCTIONS IN MATLAB

OBJECTIVES:

This lab provides introduction to MATLAB and helps students developing following skills:

- Polynomials and transfer functions
- Plotting system time responses

THEORY:

This lab includes following section

- Presenting polynomials
- Evaluating polynomials
- Representing transfer functions
- Plotting system responses

PRESENTING POLYNOMIALS:

MATLAB® represents polynomials as row vectors containing coefficients ordered by descending powers. For example, the three-element vector

```
p = [p2 p1 p0];
```

represents the polynomial

$$p(x)=p_2x^2+p_1x+p_0.$$

Create a vector to represent the quadratic polynomial

$$p(x)=x^2-4x+4.$$

```
p = [1 -4 4];
```

Intermediate terms of the polynomial that have a coefficient of 0 must also be entered into the vector, since the 0 acts as a placeholder for that particular power of x.

Create a vector to represent the polynomial

$$p(x)=4x^5-3x^2+2x+33.$$

```
p = [4 0 0 -3 2 33];
```

EVALUATING POLYNOMIALS

After entering the polynomial into MATLAB® as a vector, use the `polyval` function to evaluate the polynomial at a specific value.

Use `polyval` to evaluate $p(2)$.

```
polyval(p,2)
ans = 153
```

Alternatively, you can evaluate a polynomial in a matrix sense using `polyvalm`. The polynomial expression in one variable, $p(x)=4x^5-3x^2+2x+33$, becomes the matrix expression

$$p(X)=4X^5-3X^2+2X+33I,$$

where `X` is a square matrix and `I` is the identity matrix.

Create a square matrix, `X`, and evaluate `p` at `X`.

```
X = [2 4 5; -1 0 3; 7 1 5];

Y = polyvalm(p,X)

Y = 3×3


154392     78561    193065
 49001     24104    59692
215378    111419     269614
```

REPRESENTING TRANSFER FUNCTIONS

Control System Toolbox™ software supports transfer functions that are continuous-time or discrete-time, and SISO or MIMO. You can also have time delays in your transfer function representation.

A SISO continuous-time transfer function is expressed as the ratio:

$$G(s) = \frac{N(s)}{D(s)}$$

of polynomials $N(s)$ and $D(s)$, called the numerator and denominator polynomials, respectively.

You can represent linear systems as transfer functions in polynomial or factorized (zero-pole-gain) form. For example, the polynomial-form transfer function:

$$G(s) = \frac{s^2-3s-4}{s^2+5s+6}$$

can be rewritten in factorized form as:

$$G(s) = \frac{(s+1)(s-4)}{(s+2)(s+3)}$$

The "`tf` " model object represents transfer functions in polynomial form. The "`zpk`" model object represents transfer functions in factorized form.

Create Transfer Function Using Numerator and Denominator Coefficients

This example shows how to create continuous-time single-input, single-output (SISO) transfer functions from their numerator and denominator coefficients using tf.

Create the transfer function $G(s) = \frac{s}{s^2 = 3s + 2}$

```
 num = [1 0];
den = [1 3 2];
G = tf(num,den);
```

num and den are the numerator and denominator polynomial coefficients in descending powers of *s*. For example, den = [1 3 2] represents the denominator polynomial $s^2 + 3s + 2$.

G is a tf model object, which is a data container for representing transfer functions in polynomial form.

Alternatively, you can specify the transfer function *G*(*s*) as an expression in *s*:

1. Create a transfer function model for the variable *s*.

```
s = tf('s');
```

2. Specify *G*(*s*) as a ratio of polynomials in *s*.

```
G = s/(s^2 + 3*s + 2);
```

Create Transfer Function Model Using Zeros, Poles, and Gain

This example shows how to create single-input, single-output (SISO) transfer functions in factored form using `zpk`.

Create the factored transfer function $G(s) = 5\frac{s}{(s+2)(s+1+i)(s+1-i)}$:

```
Z = [0];
P = [-1-1i -1+1i -2];
K = 5;
G = zpk(Z,P,K);
```

`Z` and `P` are the zeros and poles (the roots of the numerator and denominator, respectively). `K` is the gain of the factored form. For example, *G*(*s*) has a real pole at $s = –2$ and a pair of complex poles at $s = –1 \pm i$. The vector `P = [-1-1i -1+1i -2]` specifies these pole locations.

`G` is a `zpk` model object, which is a data container for representing transfer functions in zero-pole-gain (factorized) form.

PLOTTING TRANSFER FUNCTIONS:

`lsim` simulates the (time) response of continuous or discrete linear systems to arbitrary inputs. When invoked without left-hand arguments, `lsim` plots the response on the screen.

`lsim(sys,u,t)` produces a plot of the time response of the dynamic system model `sys` to the input history, `t`,`u`. The vector `t` specifies the time samples for the simulation (in system time units, specified in the `Time Unit` property of `sys`), and consists of regularly spaced time samples:

```
t = 0:dt:Tfinal
```

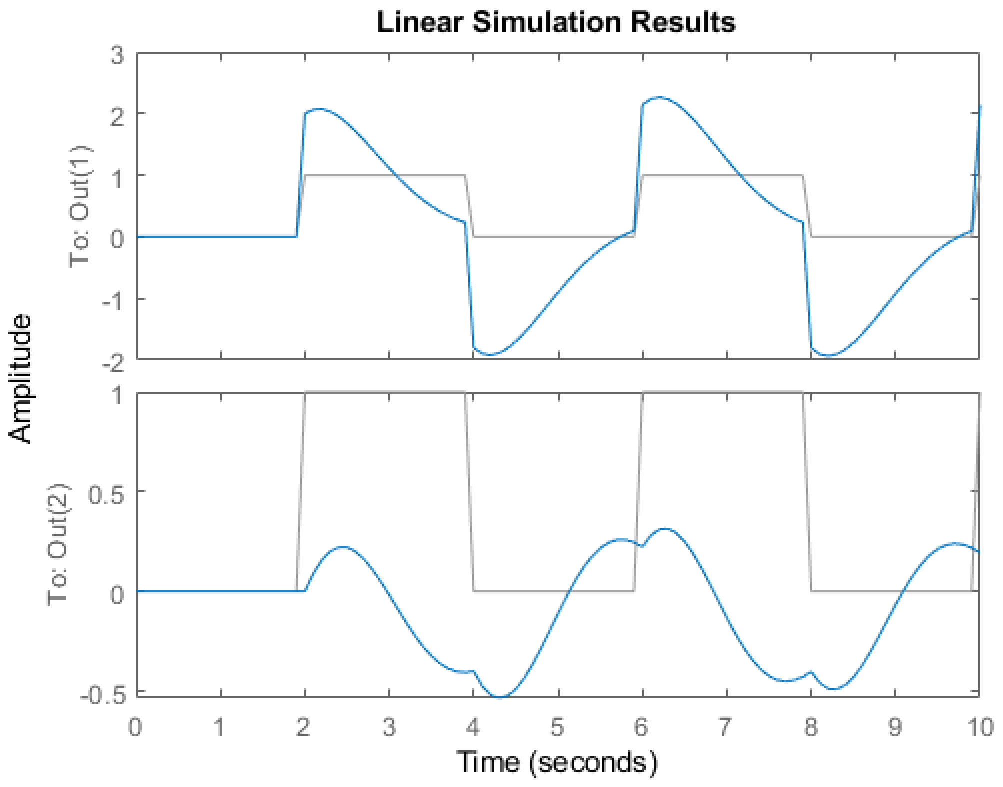
The input `u` is an array having as many rows as time samples (`length(t)`) and as many columns as system inputs. For instance, if `sys` is a SISO system, then `u` is a t-by-1 vector. If `sys` has three inputs, then `u` is a t-by-3 array. Each row `u(i,:)` specifies the input value(s) at the time sample `t(i)`. The signal `u` also appears on the plot.

The model `sys` can be continuous or discrete, SISO or MIMO. In discrete time, `u` must be sampled at the same rate as the system. In this case, the input `t` is redundant and can be omitted or set to an empty matrix. In continuous time, the time sampling `dt = t(2)-t(1)` is used to discretized the continuous model. If `dt` is too large (undersampling), `lsim` issues a warning suggesting that you use a more appropriate sample time, but will use the specified sample time.

```
H = [tf([2 5 1],[1 2 3]);tf([1 -1],[1 1 5])];
[u,t] = gensig('square',4,10,0.1);
```

Then simulate with `lsim`.

```
lsim(H,u,t)
```



LAB TASKS:

1. Define the following transfer functions in MATLAB.

$$sys1 = \frac{s^3 + 2s^2 + 4s + 5}{s^5 + 2s^3 + 4}$$

$$sys2 = \frac{s^2 + 4s + 5}{s^2 + 3s + 4}$$

$$sys3 = \frac{s^3 + 4s^2 + 2}{7s^6 + 5s^4 + 3s^2 + 1}$$

$$sys4 = 100.1\frac{(s + 2)}{(s + 1 + j)(s + 1 - j)(s + 5)}$$

$$sys5 = 70\frac{(s + 5)(s - 6)}{(s - 1)(s - 7)(s + 8)(s - 9)}$$

$$sys6 = \frac{s}{(s - 1)(s + 2)(s - 3)}$$

2. Find the resultant outputs when input is IMPULSE, STEP, RAMP, cos(10t) and cos(2t)+sin(4t).
3.

CONCLUSION AND COMMENTS:

LAB SESSION 03:

SIMULINK FOR CONTROL SYSTEMS

OBJECTIVES:

This lab provides introduction to Simulink and helps students developing following skills:

- Using library browser and components with in
- Creating a Simulink model
- Analysis of a Dynamic Control System Model

THEORY:

WHAT IS SIMULINK?

Simulink is a software package for modeling, simulating and analyzing dynamical systems. It supports linear and nonlinear systems, modeled in continuous time, sampled time, or a hybrid of the two. Simulink can easily accommodate multi rate systems as well. Simulink offers a friendly, graphical environment, in which you can model, systems in the form of block diagrams, by simply clicking and dragging blocks into a model window. You can run a model at the push of a button and modify it easily. The graphical nature of Simulink models makes them easy for others to read and understand.

Simulink has a comprehensive block library of sinks, sources, subsystems (linear, nonlinear, and time-varying), connectors, and powerful conditionally-enabled blocks. You can also customize your own blocks.

You can use all the facilities of MATLAB when running Simulink. You can invoke familiar MATLAB expressions and M-file functions as temporary utilities, for instance to control display and visualization. You can even encapsulate them as blocks and place them in Simulink models. Using Simulink, you can see data displayed in scopes as the simulation unfolds. This makes tracing and debugging a model, by quick, proof-of-concept demonstrations, much faster and easier than by working directly from the command line. Outside the simulation environment, Simulink serves as the primary link for targeting to chips, boards, and co-simulation platforms by means of automatic code generation.

Blocksets are comprehensive collections of Simulink blocks that extend the Simulink environment to solve particular classes of problems. Areas in which blocksets are available include digital signal processing, communications, embedded target for TI C6000, Xilinx and many others.

WHAT IS THE COMMUNICATIONS BLOCKSET?

The Communications Blockset is a collection of Simulink® blocks for designing and simulating communication systems. With the Communications Blockset, you can design models in the form of block diagrams, using simple click-and-drag mouse operations. You can run simulations on a model at the push of a button, and change parameters while the simulation is running. The Communications Blockset contains ready-to-use blocks to model various processes within communication systems, including

- Signal generation
- Source coding
- Error-correction
- Interleaving
- Modulation/demodulation
- Transmission along a channel
- Synchronization

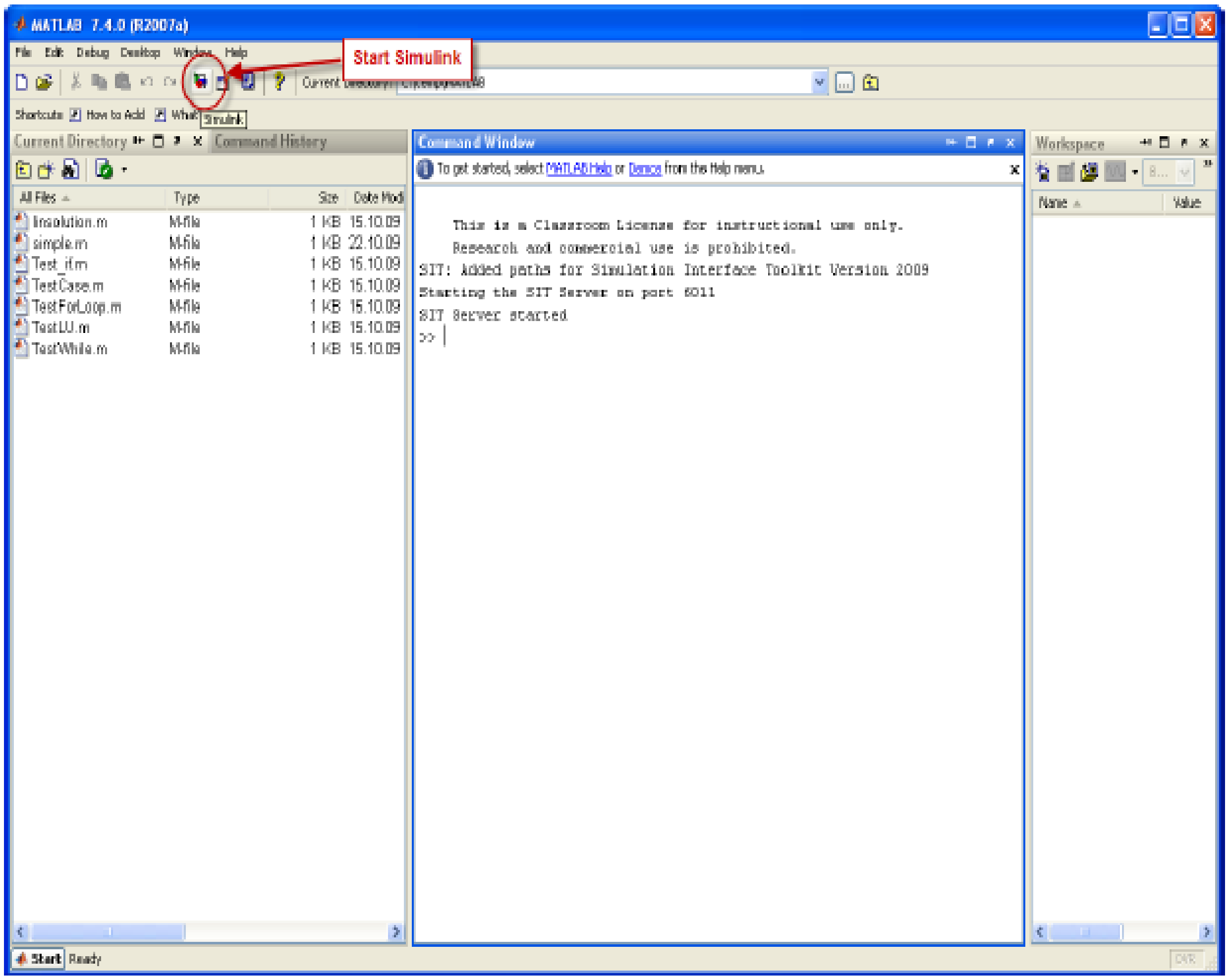
In addition, you can create specialized blocks, to implement your own algorithms.
All the power of MATLAB® is available to you when you use the Communications Blockset. You can run simulations from the command line and invoke MATLAB expressions and M-files.

WHAT BACKGROUND DO I NEED?

Ideally, you should know a little something about MATLAB and the easy, supportive way it leads you from elementary calculations, all the way to powerful, matrix-intensive algorithm. Development and execution. But if you are a newcomer to MATLAB and Simulink, just keep reading here. This manual will carry you through the basics. Everything you need to know and do here is easy – and fun! If you want to learn more about MATLAB and Simulink, beyond what is covered in this manual, you can take a look at Getting Started with MATLAB or Using Simulink. You can read either of these on-line by selecting the Help tab in the MATLAB Command Window. These references are also available on the Math Works Web site at (http://www.mathworks.com/)
The topics in this manual illustrate the techniques you need to use Simulink for modeling and simulating Communication systems. If you are a Communication novice, just read on Simulink can illuminate Communication concepts in very exciting ways.

STARTING SIMULINK

Unlike MATLAB, there is no special command window for Simulink. It works in the background whenever you build and run models. Before using Simulink, you must first start MATLAB.

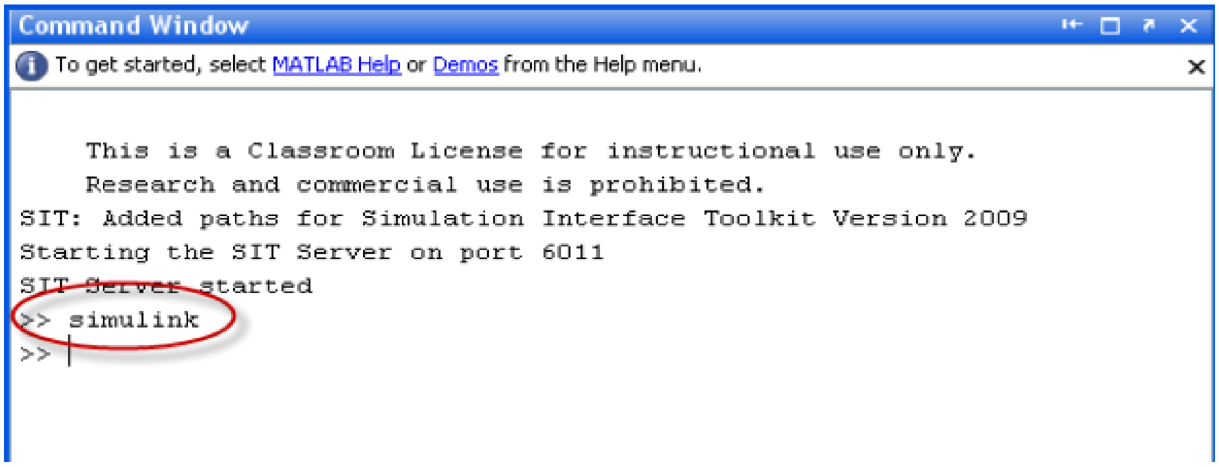


All Files	Type	Size	Date Modi
linsolution.m	M-file	1 KB	15.10.09
simple.m	M-file	1 KB	22.10.09
Test_if.m	M-file	1 KB	15.10.09
TestCase.m	M-file	1 KB	15.10.09
TestForLoop.m	M-file	1 KB	15.10.09
TestLU.m	M-file	1 KB	15.10.09
TestWhile.m	M-file	1 KB	15.10.09

To get started, select MATLAB Help or Demos from the Help menu.

```
    This is a Classroom License for instructional use only.
    Research and commercial use is prohibited.
SIT: Added paths for Simulation Interface Toolkit Version 2009
Starting the SIT Server on port 6011
SIT Server started
>> 
```

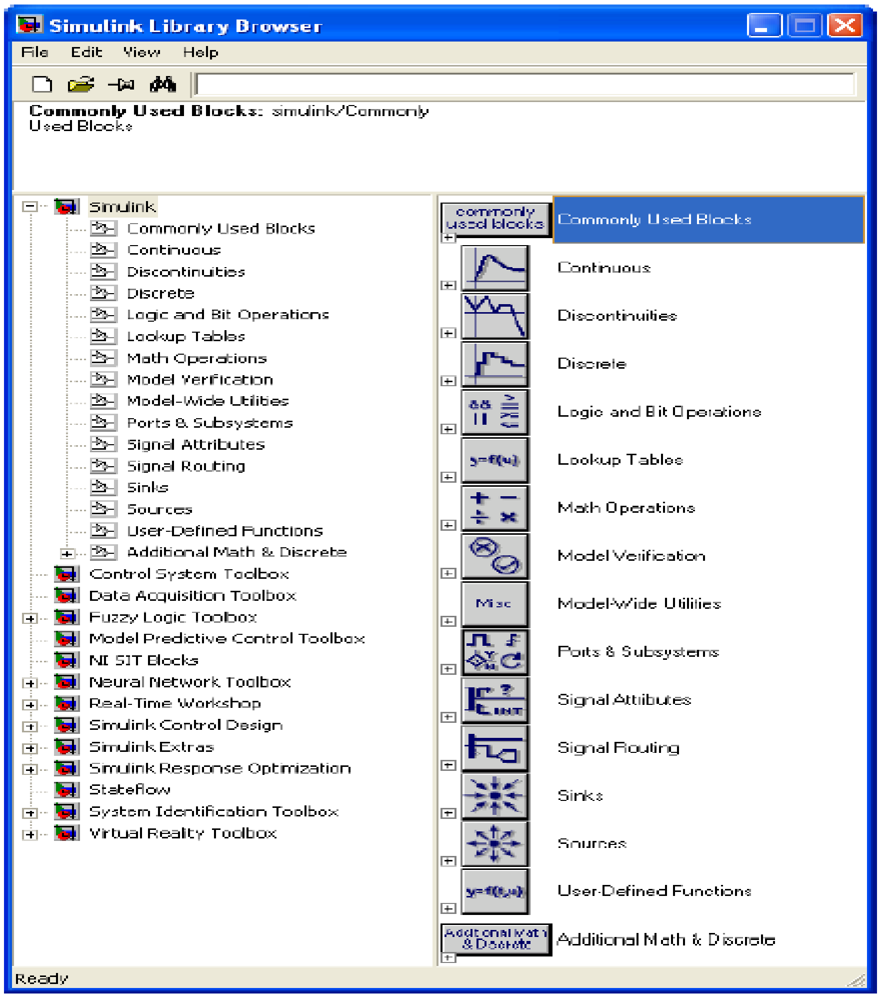
How to Start Simulink from icon available in MATLAB Toolbar?



To get started, select MATLAB Help or Demos from the Help menu.

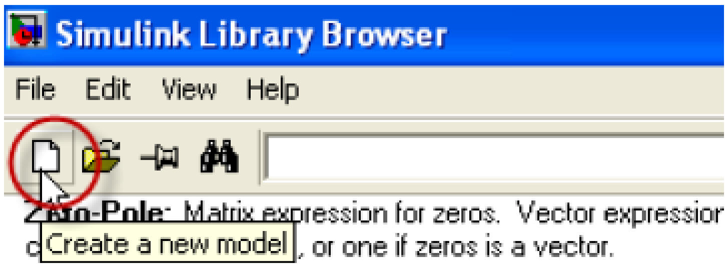
```
    This is a Classroom License for instructional use only.
    Research and commercial use is prohibited.
SIT: Added paths for Simulation Interface Toolkit Version 2009
Starting the SIT Server on port 6011
SIT Server started
>> simulink
>> 
```

How to start Simulink by writing Simulink in command prompt?



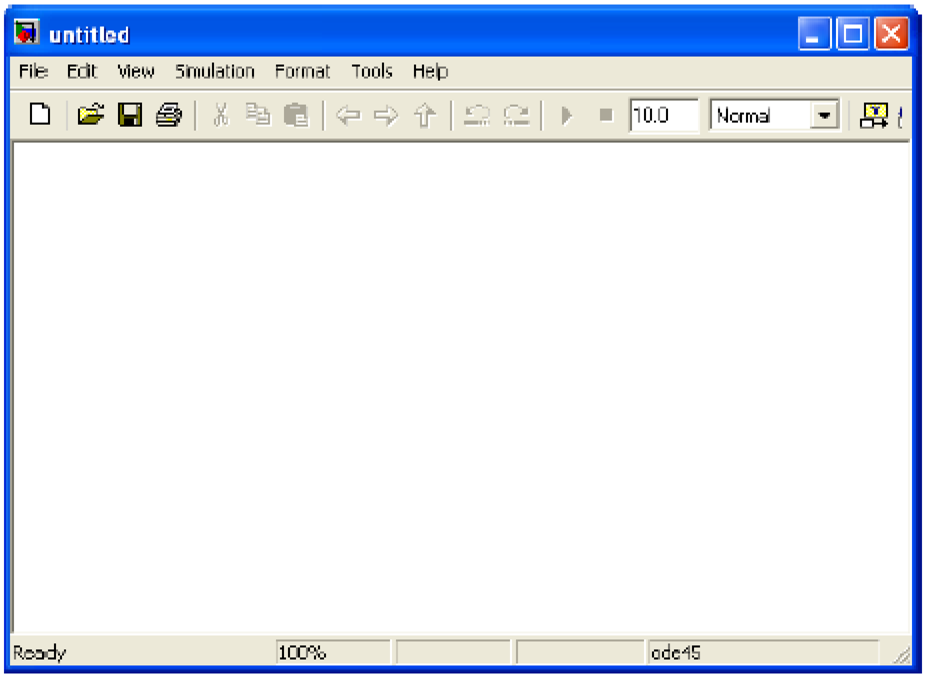
Simulink Library

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.
Click the New icon on the Toolbar in order to create a new Simulink model.


Simulink Library Browser
File Edit View Help
Zero-Pole: Matrix expression for zeros. Vector expression
Create a new model
, or one if zeros is a vector.

How to Start Creating Model in Simulink?

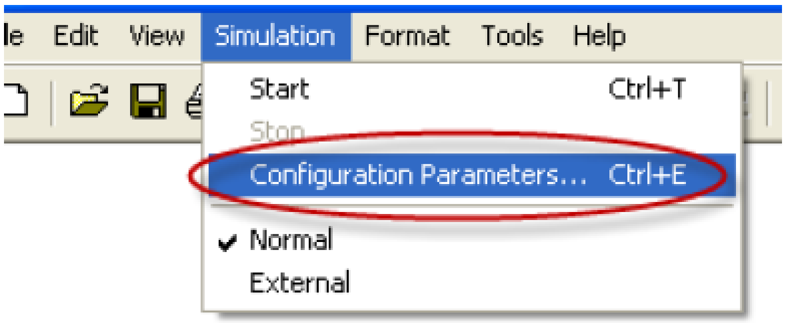
The following window appears. You may now drag the blocks you want to use from the Simulink Library Browser to the model surface (or right-click on a block and select “Add to…”)



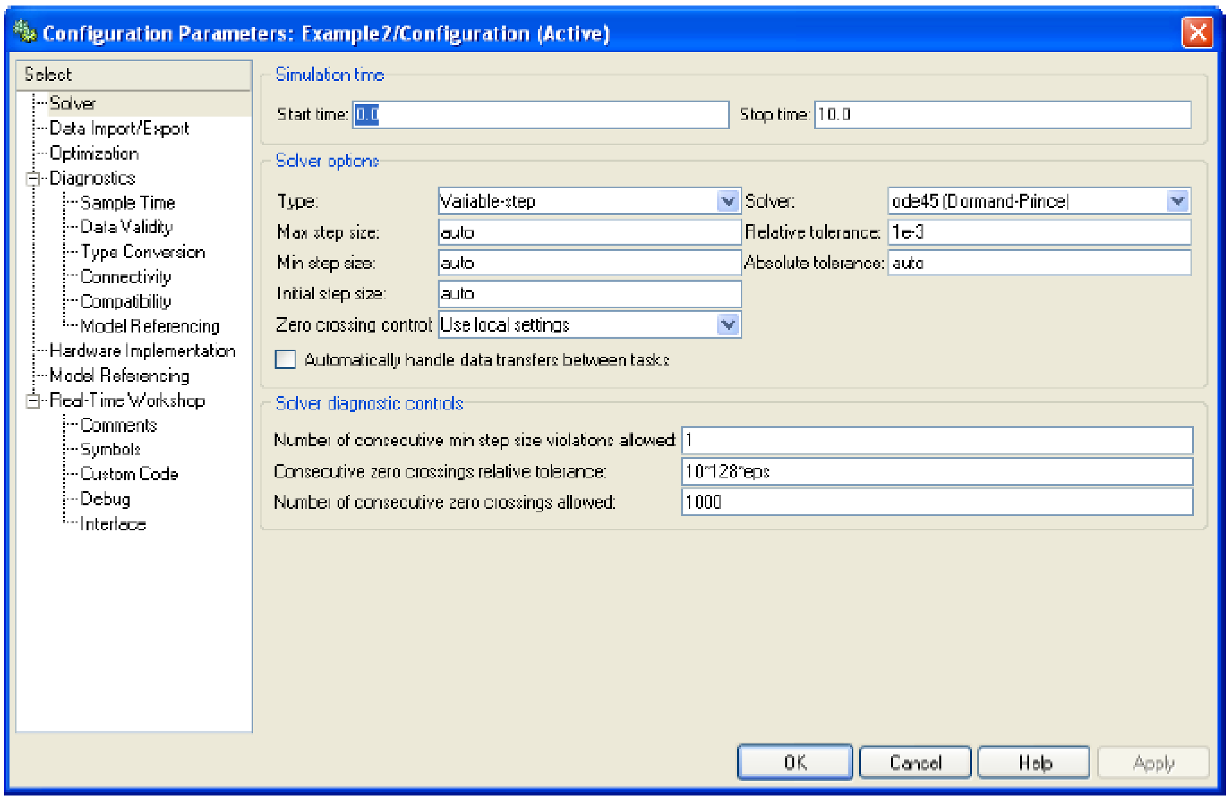
Model Surface for creating Simulink Model

There are lots of parameters you may want to configure regarding your simulation. Select "Configuration Parameters…" in the Simulation menu. After that following window appear. Here you set important parameters such as:

- Start and Stop time for the simulation
- What kind of Solver to be used (ode45, ode23 etc.)
- Fixed-step/Variable-step


le Edit View Simulation Format Tools Help
Start Ctrl+T
Stop
Configuration Parameters… Ctrl+E
Normal
External

Simulation configuration in Simulink



Configuration Parameters: Example2/Configuration (Active)

Simulation time

Start time: 0.0 Stop time: 10.0

Solver options

Type: Variable-step Solver: ode45 (Dormand-Prince)

Max step size: auto Relative tolerance: 1e-3

Min step size: auto Absolute tolerance: auto

Initial step size: auto

Zero crossing control: Use local settings

☐ Automatically handle data transfers between tasks

Solver diagnostic controls

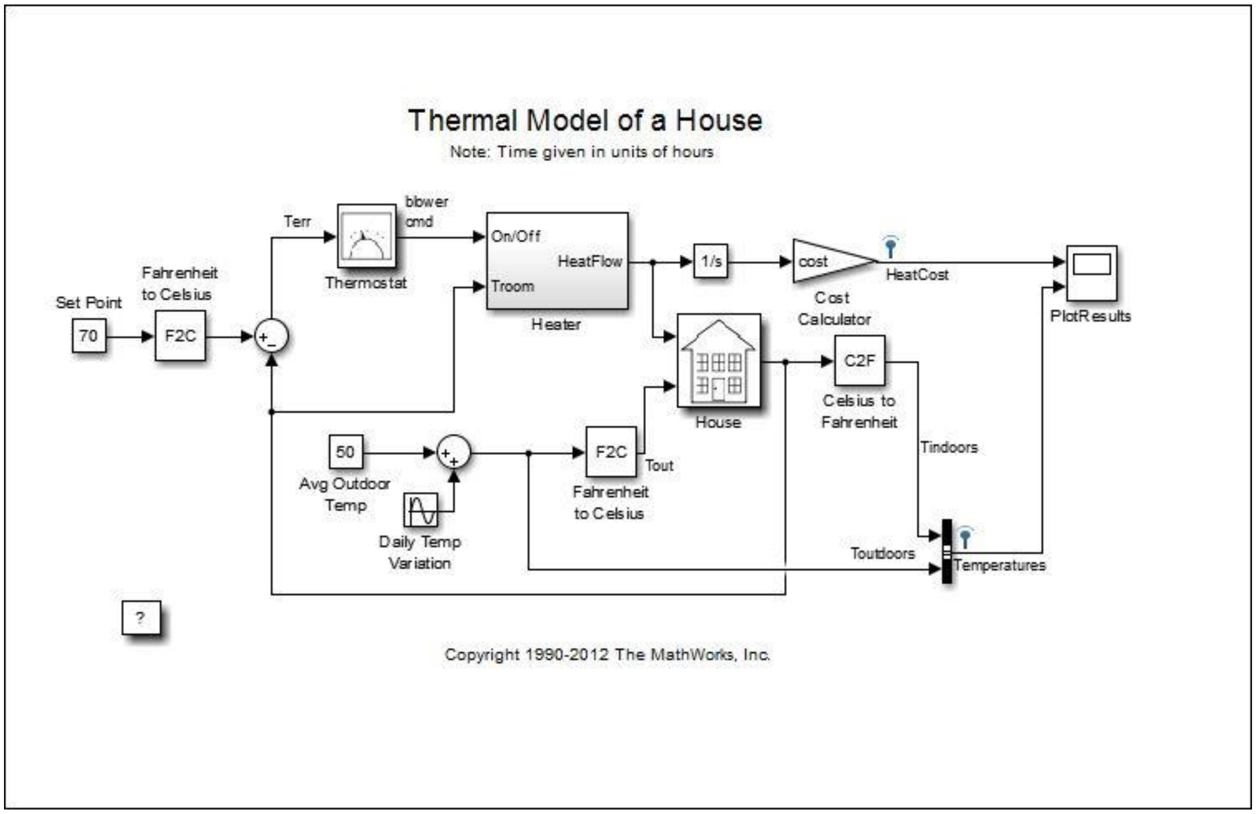
Number of consecutive min step size violations allowed: 1

Consecutive zero crossings relative tolerance: 10*128*eps

Number of consecutive zero crossings allowed: 1000

Configuration window in Simulink

STUDYING EXAMPLE MODEL IN SIMULINK:



MODEL INITIALIZATION

This model calculates heating costs for a generic house. When the model is opened, it loads the information about the house from the sldemo_househeat_data.m file. The file does the following:

- Defines the house geometry (size, number of windows)
- Specifies the thermal properties of house materials
- Calculates the thermal resistance of the house
- Provides the heater characteristics (temperature of the hot air, flow-rate)
- Defines the cost of electricity (0.09$/kWhr)
- Specifies the initial room temperature (20 deg. Celsius = 68 deg. Fahrenheit).

MODEL COMPONENTS

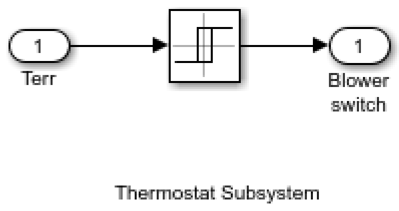
Set Point

"Set Point" is a constant block. It specifies the temperature that must be maintained indoors. It is 70 degrees Fahrenheit by default. Temperatures are given in Fahrenheit, but then are converted to Celsius to perform the calculations.

Thermostat

"Thermostat" is a subsystem that contains a Relay block. The thermostat allows fluctuations of 5 degrees Fahrenheit above or below the desired room temperature. If air temperature drops below 65 degrees Fahrenheit, the thermostat turns on the heater. See the thermostat subsystem below.

Open the Thermostat subsystem



The "Thermostat" Subsystem

Heater

"Heater" is a subsystem that has a constant air flow rate, "M dot", which is specified in the sldemo_househeat_data .m file. The thermostat signal turns the heater on or off. When the heater is on, it blows hot air at temperature T Heater (50 degrees Celsius = 122 degrees Fahrenheit by default) at a constant flow rate of M dot (1kg/sec = 3600kg/hr by default). The heat flow into the room is expressed by the Equation 1.

Equation 1

$$\frac{dQ}{dt} = (T_{heater} - T_{room}) \cdot Mdot \cdot c$$

$$\frac{dQ}{dt} = \text{ heat flow from the heater into the room}$$

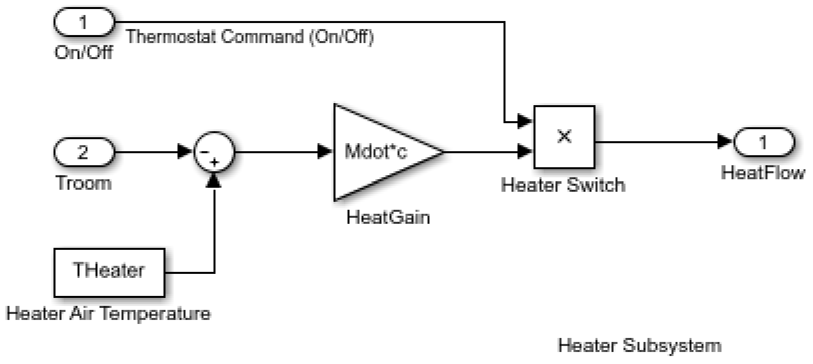
$$c = \text{ heat capacity of air at constant pressure}$$

$$Mdot = \text{ air mass flow rate through heater (kg/hr)}$$

$$T_{heater} = \text{ temperature of hot air from heater}$$

$$T_{room} = \text{ current room air temperature}$$

Open the Heater subsystem



The Heater Subsystem

Cost Calculator

"Cost Calculator" is a Gain block. "Cost Calculator" integrates the heat flow over time and multiplies it by the energy cost. The cost of heating is plotted in the "Plot Results" scope.

House

"House" is a subsystem that calculates room temperature variations. It takes into consideration the heat flow from the heater and heat losses to the environment. Heat losses and the temperature time derivative are expressed by Equation 2.

Equation 2

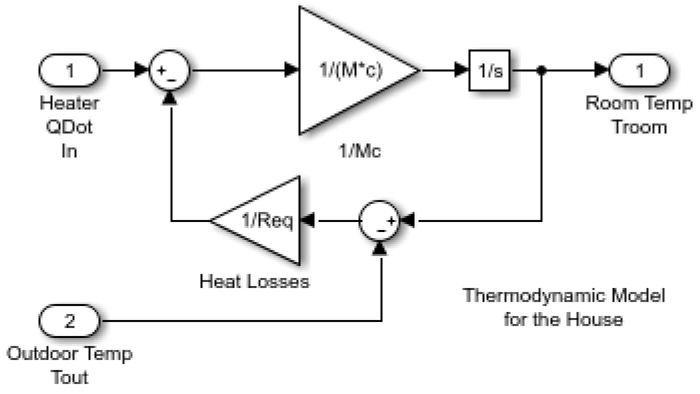
$$\left(\frac{dQ}{dt}\right)_{losses} = \frac{T_{room} - T_{out}}{R_{eq}}$$

$$\frac{dT_{room}}{dt} = \frac{1}{M_{air} \cdot c} \cdot \left(\frac{dQ_{heater}}{dt} - \frac{dQ_{losses}}{dt}\right)$$

$M_{air} =$ mass of air inside the house

$R_{eq} =$ equivalent thermal resistance of the house

Open the House subsystem

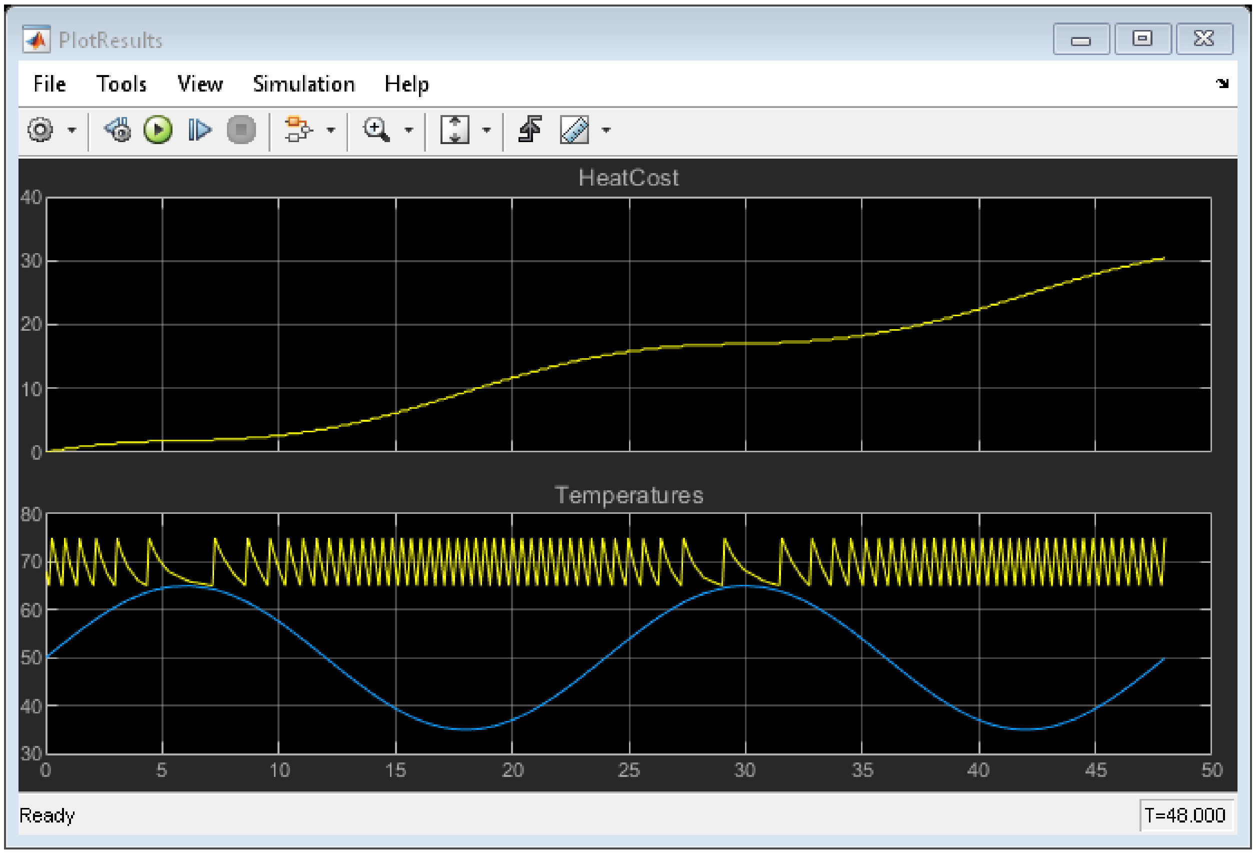


The House Subsystem

Modeling the Environment

We model the environment as a heat sink with infinite heat capacity and time varying temperature Tout. The constant block "Avg Outdoor Temp" specifies the average air temperature outdoors. The "Daily Temp Variation" Sine Wave block generates daily temperature fluctuations of outdoor temperature. Vary these parameters and see how they affect the heating costs.

RUNNING THE SIMULATION AND VISUALIZING THE RESULTS:



LAB TASKS:

1. Study any example model already defined in Simulink examples.
2. Build a simple model using integrator, source (sine wave, step, impulse, ramp) and sink (oscilloscope).
3. Find the resultant outputs and discuss.

CONCLUSION AND COMMENTS:

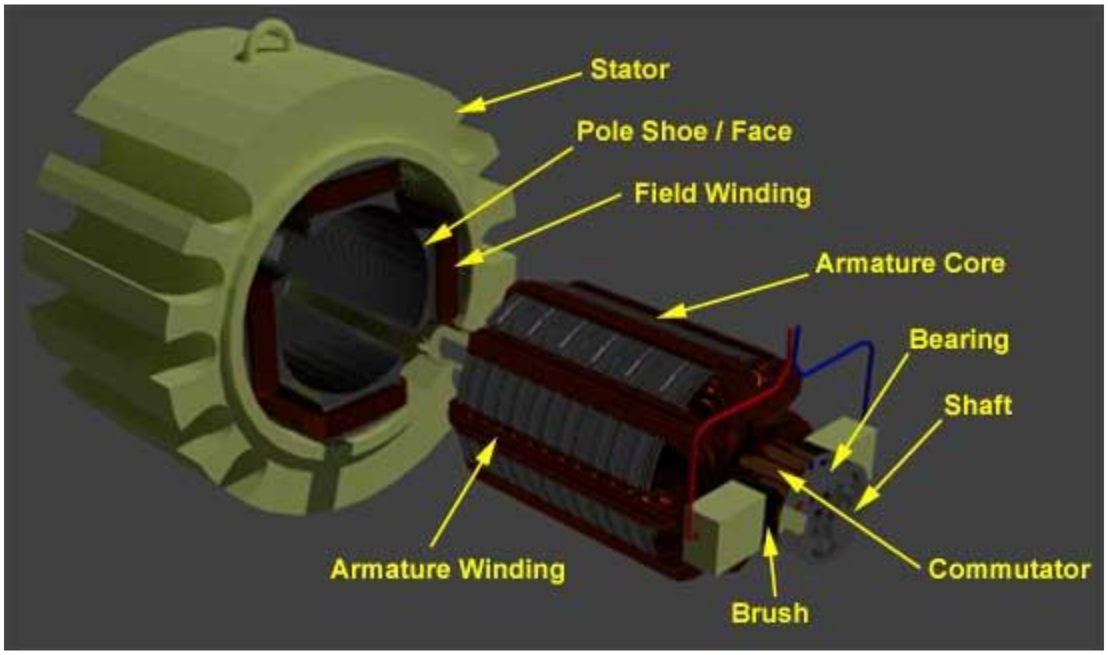
LAB SESSION 04:

DIFFERENTIAL EQUATION MODELING AND ANALYSIS OF ARMATURE CONTROLLED DC MOTOR SPEED

OBJECTIVES:

- To understand DC Motor Speed characteristics
- Model and analyze the system differential equations and Transfer function of armature controlled DC motor
- Determine the open-loop step response using **ltiview** in MATLAB

THEORY:



Main parts of the DC motor include the following:

- Stator – The static part that houses the field windings and receives the supply
- Rotor – The rotating part that brings about the mechanical rotations.
- Yoke of dc motor.
- Poles of dc motor.
- Field winding of dc motor.
- Armature winding of dc motor.
- Commutator of dc motor.
- Brushes.
-

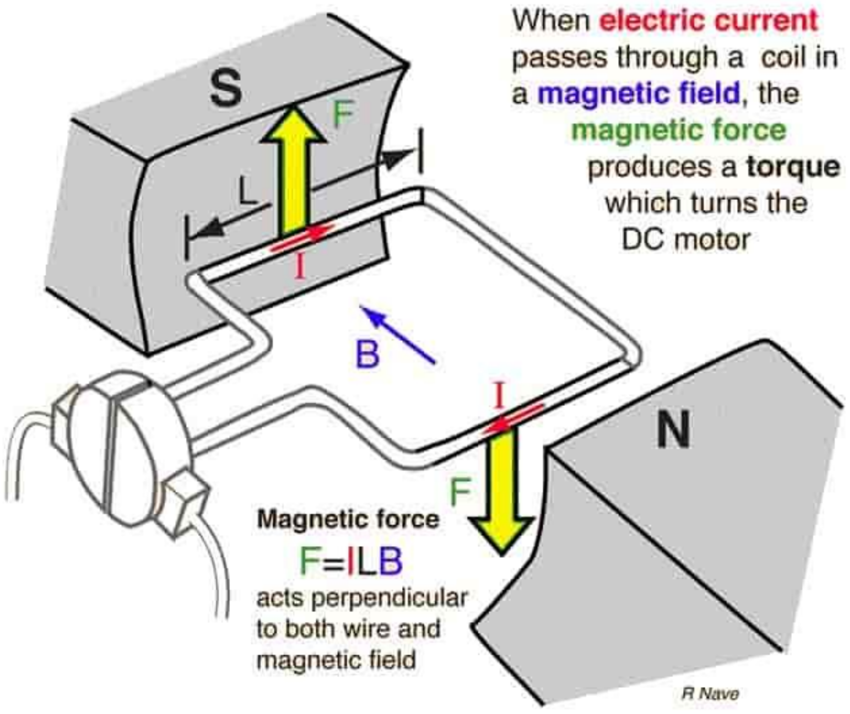
All these parts put together configures the total construction of a dc motor.

DC MOTOR WORKING WITH DIAGRAM

DC motor working is based on the principle that when a current carrying conductor is placed in a magnetic field, the conductor experiences a mechanical force. The direction of this force is given by Fleming's left-hand rule and magnitude is given by

F = BIL Newtons

According to Flemings left-hand rule when an electric current passes through a coil in a magnetic field, the magnetic force produces a torque which turns the DC motor. The direction of this force is perpendicular to both the wire and the magnetic field.



S
F
L
I
B
I
N
F
When electric current passes through a coil in a magnetic field, the magnetic force produces a torque which turns the DC motor
Magnetic force
F=ILB
acts perpendicular to both wire and magnetic field
R Nave

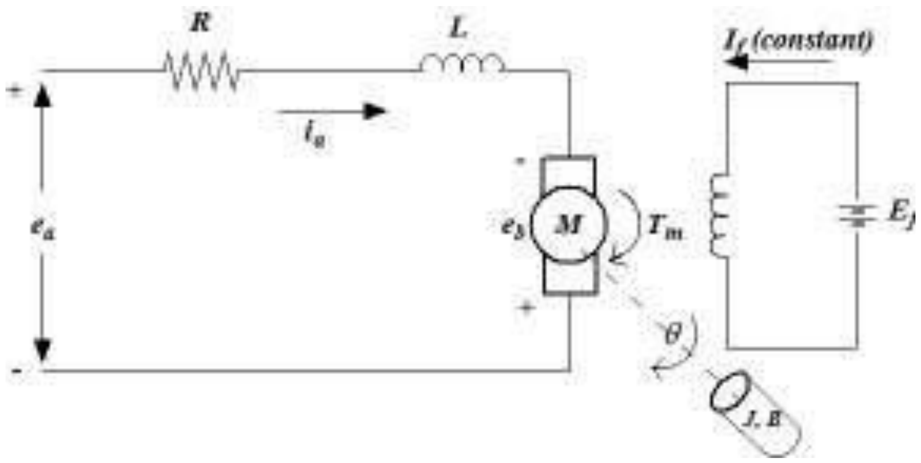
Armature Controlled DC Motor

The speed of a dc motor can be controlled by varying the voltage applied to the armature of a dc motor. A separately excited dc motor with variable armature voltage finds application as a drive motor in a variable speed drive. The variable armature voltage is provided by a phase controlled rectifier. The torque developed by the dc motor

$T_d = KØi_a$

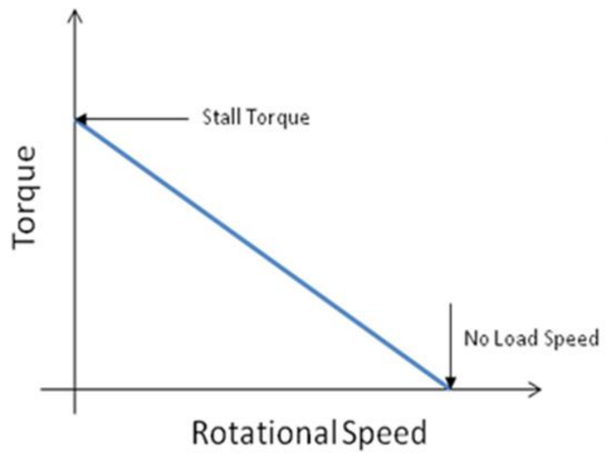
Where;

Ø is air gap flux.
Ia is armature current
K is a constant



DC MOTOR EQUATIONS

DC motors are relatively simple machines: when the load on the motor is constant, speed is proportional to supply voltage. And when supply voltage is constant, speed is inversely proportional to the load on the motor. This second relationship—between speed and load (or torque)—is typically shown on the motor’s torque-speed curve.



The inverse relationship between speed and torque means that an increase in the load (torque) on the motor will cause a decrease in speed. This can be demonstrated by the DC motor torque equation:

$$T = \frac{V - \omega \cdot k}{R} \cdot k$$

T = motor torque
V = supply voltage
ω = rotational speed
k = motor constant
R = resistance

Because the torque-speed curve is a straight line, it's simple to find the torque that the motor can produce at a given speed, or conversely, to find the motor's speed for a given load (torque) on the shaft. Recall the equation for a straight line:

$$y = m \cdot x + b$$

Where:

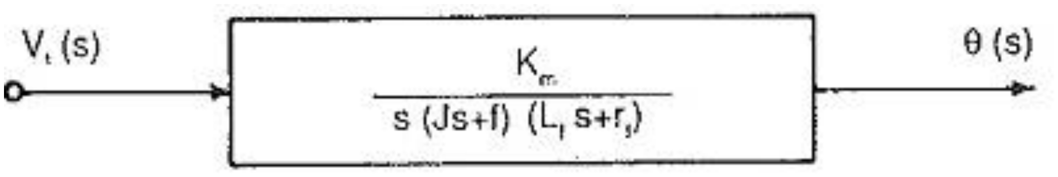
y = value of y axis variable, to be determined
m = slope of the line; change in y divided by change in x
x = value of x axis variable, given
b = y intercept; point at which the line crosses the y axis

Using this equation for the torque-speed curve, we can find the motor's torque at a given speed. In this case, the variables in the line equation represent the following:

$$x = \frac{y - b}{m}$$

y = torque to be determined
m = change in torque divided by change in speed
x = given speed
b = stall torque (value where the line crosses the y axis)
The line equation can also be rearranged to find the motor's speed at a given torque:

TRANSFER FUNCTION IN LAPLACE DOMAIN



Transfer function

3(c & d) *Block diagram of field controlled dc motor [Armature current constant]*

LAB TASKS:

1. Write MATLAB script file and determine the exact expression for $P(s) = \frac{K}{(Js+b)(Ls+R)+K^2}$ by plugging in the motor constants?
   J = 0.01
   b = 0.1
   K = 0.01
   R = 1
   L = 0.5

2. Find the step response using the command ltiview. Show the characteristics of the graph Peak response, Settling time, Rise time and Steady state. Locate the Poles and Zeros using Pole /Zero plot option.

CONCLUSION AND COMMENTS:

LAB SESSION 05:

MAGNETIC LEVITATION SYSTEM MODEL AND ANALYSIS

OBJECTIVES:

- Introduction of GML (Googol Magnetic Levitation) Control System and Applications
- System Modeling of Magnetic Levitation
- MATLAB Simulink Simulation

THEORY:

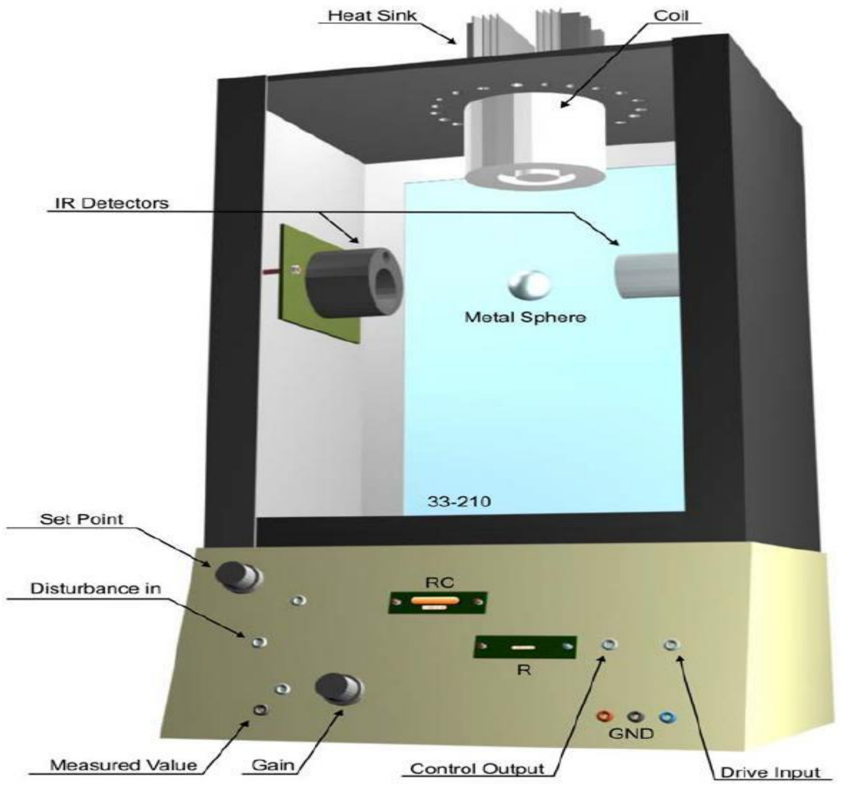
OVERVIEW OF MAGNETIC LEVITATION SYSTEM WITH DIAGRAM:

The Maglev station consist of an electromagnetic coil, a metal sphere, op-amp based analogue controller unit with changeable elements, a built-in current-voltage converter/driver and several connection nodes to achieve external connections with DAQ equipment. Station, itself can be operated with the suitable RC network configurations placed on op-amp connection nodes without using a DAQ connected to a computer, this type of operation will be called "standalone operation" .

Description of Elements

Name	Description
Coil	The coil generates and applies the electromagnetic force to the sphere in the opposite direction to gravity.
Heat Sink	Heat Sink prevents the coil from damages caused by the temperature created by the copper loss in the windings.
Metal Sphere	Metal Sphere is made of stainless steel, its diameter is 25 mm and it has a weight of 20 grams.
IR Detectors	IR Detectors is designed to measure the distance between the zero-point and the sphere.
Set Point	This potentiometer adjusts the desired set-point value of the sphere when the system used in standalone mode.
Disturbance In	This input can be used to examine the effect of the disturbances over the controller.
Measured Value	This node gives the inverted value of the measured sphere position in volts (To calculate the real position in meters; a linear conversion is

	needed.)
Gain	The gain potentiometer adjusts the overall gain of the controller when the system used in standalone mode.
Control Output	Control Output is the output of the built-in controller; in standalone mode it is directly connected to drive input.
Drive Input	Drive input converts the given voltage level to the current level applied to the coil windings. (It has a constant convert ratio between voltage and current)



Magnetic Levitation System

MAGNETIC LEVITATION APPLICATIONS IN REAL LIFE

- Transportation engineering (magnetically levitated trains, flying cars, or personal rapid transit (PRT), etc.)
- Environmental engineering (small and huge wind turbines: at home, office, industry, etc.)

- Aerospace engineering (spacecraft, rocket, etc.)
- Military weapons engineering (rocket, gun, etc.),
- Nuclear engineering (the centrifuge of nuclear reactor),
- Civil engineering including building facilities and air conditioning systems (magnetic bearing, elevator, lift, fan, compressor, chiller, pump, gas pump, geothermal heat pumps, etc.),
- Biomedical engineering (heart pump, etc.),
- Chemical engineering (analyzing foods and beverages, etc.),
- Electrical engineering (magnet, etc.),
- Architectural engineering and interior design engineering including household and administrative appliances (lamp, chair, sofa, bed, washing machine, room, toys (train, levitating spacemen over the space ship, etc.), stationery (pen), etc.),
- Automotive engineering (car, etc.),
- Advertising engineering.

LAB TASKS:

1. Develop a Simulink model for magnetic levitation system
2. Select transfer function block from continuous block library.
3. Transfer function is $\frac{77.8421}{0.0311s^2-30.5250}$
4. Change the block name to GML transfer function
5. Change the color of the block
6. Drag step block from the sources block
7. Drag the scope from sinks
8. Connect all the blocks
9. Plot the step response and record the plot generated
10. Write Matlab script for the system and calculate responses and then compare.

CONCLUSION AND COMMENTS:

LAB SESSION 06:

BALL AND BEAM MODELING AND ANALYSIS

OBJECTIVES:

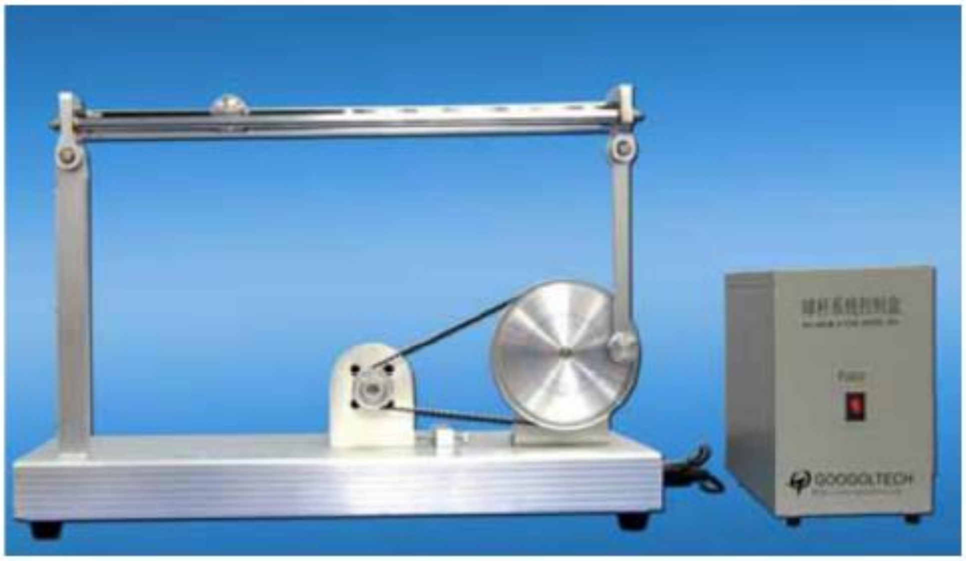
- Understanding the Mechanical Model of Ball and Beam
- Determining Transfer Function of ball and Beam
- Mechanical Model of Ball and beam in SIMULINK

THEORY:

The **ball and beam** system consists of a long beam which can be tilted by a servo or electric motor together with a ball rolling back and forth on top of the beam. It is a popular textbook example in control theory. The significance of the ball and beam system is that it is a simple system which is open-loop **unstable**. Even if the beam is restricted to be very nearly horizontal, without active feedback, it will swing to one side or the other, and the ball will roll off the end of the beam. To stabilize the ball, a control system which measures the position of the ball and adjusts the beam accordingly must be used.
The ball-beam system is a frequently encountered example of a nonlinear dynamical system. While the ideal ball-beam system is indeed nonlinear, its practical implementation in the lab has additional non-linearity, including:

- Deadband,
- Backlash introduced by the DC motor and belt pulley,
- Discrete position sensing,
- Uneven rolling surface.

These effects are traditionally hard to measure and to model. However they may drastically influence the behavior of the system under control. How to build a robust control for such systems is one of the most important questions of modern theory.

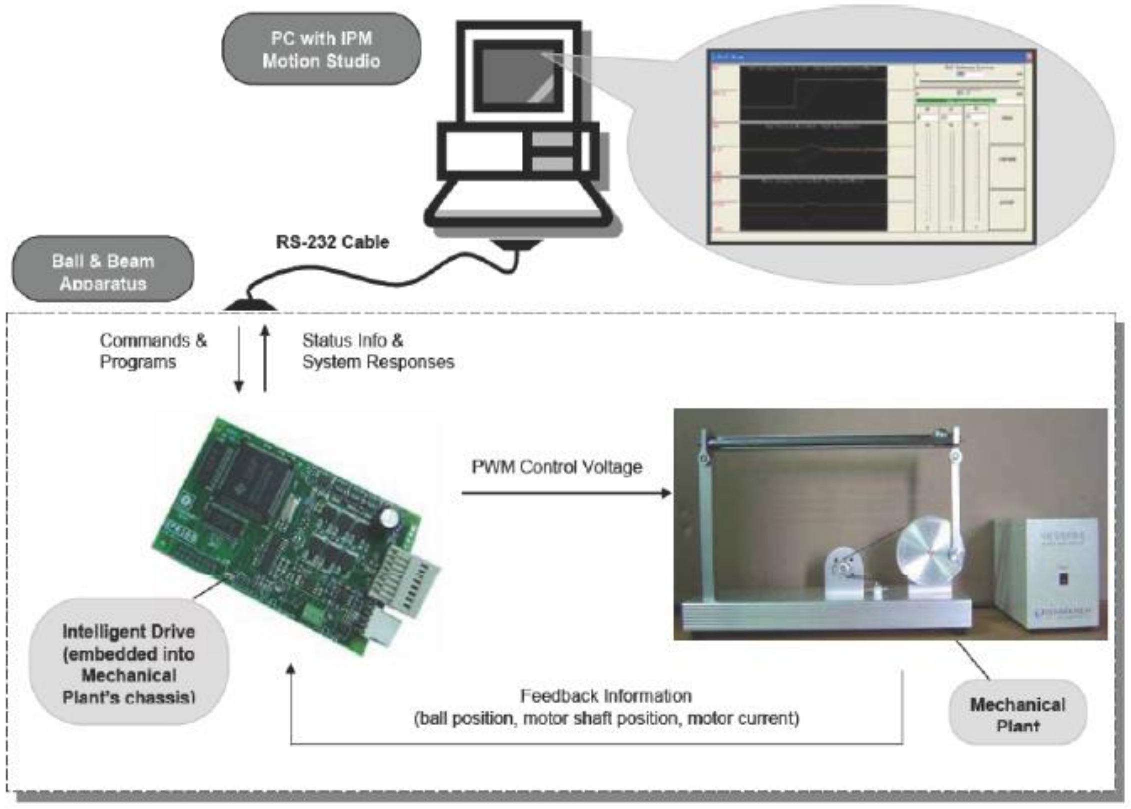


BALL&BEAM FUNCTIONAL COMPONENTS:

Ball & Beam apparatus that includes

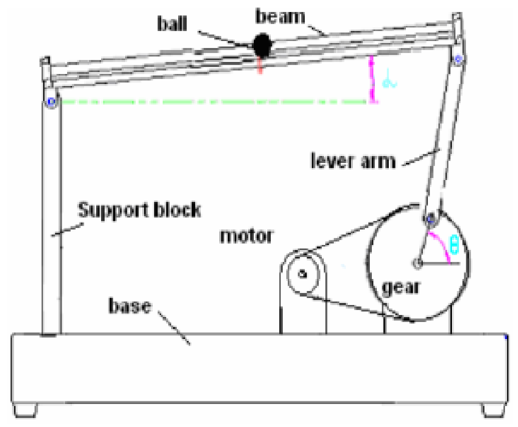
- mechanical plant with built-in DC servo motor and DC voltage supply,
- IPM100 intelligent servo drive system,
- PC with the control program.

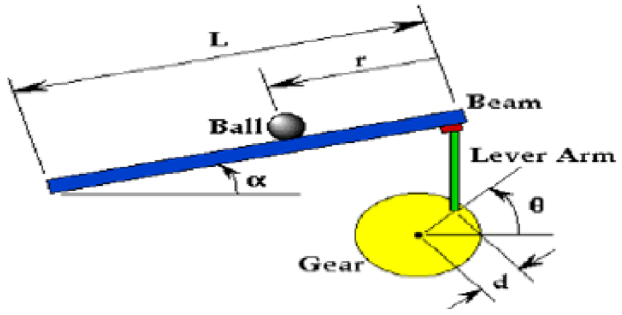
The details of functional blocks are discussed in the following sections.


PC with IPM Motion Studio
RS-232 Cable
Ball & Beam Apparatus
Commands & Programs
Status Info & System Responses
PWM Control Voltage
Intelligent Drive (embedded into Mechanical Plant's chassis)
Feedback Information (ball position, motor shaft position, motor current)
Mechanical Plant

MECHANICAL ASSEMBLY OF BALL & BEAM SYSTEM

The mechanical plant consists of a base, a beam, a ball, a lever arm, a belt pulley, a support block and a motor as shown in figure below.
The ball can roll freely along the whole length of the beam. The beam is connected to the fixed support block at one side and to the movable lever arm at another one. In turn the motion of the lever arm is controlled by the DC brush motor through belt pulley. The motor is equipped with built-in rotary **optical incremental encoder** (1000P/R) that provides feedback information about current actual rotary position of the motor shaft. In the slot along the beam there is a **linear potentiometer sensor** that senses current linear actual position of the ball on the beam. Both measured positions are fed back to the control system to organize a closed loop control.
As the servo gear turns by an **angle theta**, the lever arm changes the **angle of the beam by alpha**.
When the beam is moved away from horizontal position, gravity causes the ball to roll along the beam.
The purpose is to design and implement a control algorithm that moves the shaft of the motor in such
a way that the desired position of the ball is stabilized on the beam.
The purpose is to design and implement a control algorithm that moves the shaft of the motor in such a way that the desired position of the ball is stabilized on the beam.





INTELLIGENT DRIVE

The motion of the motor's shaft is governed by **IPM100 intelligent drive**. This is a high precision, fully digital servo drive with embedded intelligence and **100W power amplifier** suitable for brushless/brush motors. Based on feedback information from sensors it computes and then applies appropriate **PWM modulated voltage** to the motor windings in such a way that a sufficient torque moves the motor shaft according the user programmed control algorithm. The drive is called "intelligent" because besides built-in power amplifier for control signal amplification and PWM modulation.
It has an **on-board Digital Signal Processor (DSP)**, memory and other digital logic which offer advanced motion control and PLC functionality. Real-time trajectory generation, closed loop servo control, handling commands from host computer and processing I/O signals are executed on-board according to the stored programs and on-line commands from host PC. This embedded intelligence provides a true real-time control performance independent of any delays caused by PC's non-real time Operating System. Physically IPM100 is located inside the electric control box and communicates with the upper-level **PC through RS-232 interface**. The DC voltage to the drive is provided by the DC power supply also inside the electric control box.

PC-BASED CONTROL SOFTWARE

The user programs the drive with the high-level **Technosoft Motion Language (TML)** in IPM Motion Studio running on host PC. The code and on-line commands are downloaded to the drive for execution through PC's COM port.
IPM Motion Studio is an advanced and intuitive-based Windows Integrated Development Environment for the set-up and analysis of motion control applications with IPM drives.
With this platform it is easy to:

- Identify motor, sensor and load parameters
- Tune and adjust the servo-drive control loops
- Build flexible motion control algorithms/programs using TML
- Analyze and evaluate the behavior of the system

The user can define the motor commands to be applied to the motor. The Motion Wizard tool can help to generate all TML instructions in a graphical way without need to write any actual TML code.

Specific selection and definition dialogs can be opened, viewed and edited depending on the command type to be generated. The advanced **graphics tools include Data Logger, Control Panel, and Watches for TML parameters, registers and memory**. They can be used to perform real-time analysis of the behavior of the motion system.
It is advisable for users to read “IPM100 Motion Studio User Manual” for more details on IPM operation before assuming any practical steps with Ball & Beam.

THE BALL AND BEAM PACKAGE

It is important to inspect the GBB1004 package immediately upon receiving it for any mechanical damage, though it has been factory verified. If any damage has been found, please DO NOT use the product and contact us immediately.
The package includes the following items (details please refer to the packing list):

- Mechanical plant
- Electric control box with power supply and intelligent IPM100 servo drive inside
- Steel balls
- Power cable
- 9-pin and 4-pin cables
- CD with software and documentation

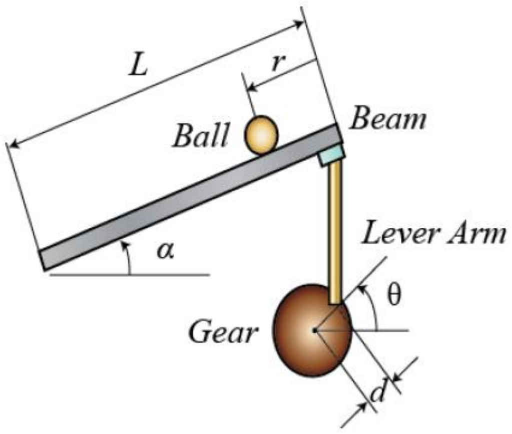
System Model Assumptions: For this problem, we will assume that the ball rolls without slipping and friction between the beam and ball is negligible. The constants and variables for this example are defined as follows:

MECHANICAL MODELING OF BALL AND BEAM

- Physical setup
- System parameters
- Design criteria
- System equations
-

PHYSICAL SETUP

A ball is placed on a beam, see figure below, where it is allowed to roll with 1 degree of freedom along the length of the beam. A lever arm is attached to the beam at one end and a servo gear at the other. As the servo gear turns by an angle, the lever changes the angle of the beam by. When the angle is changed from the horizontal position, gravity causes the ball to roll along the beam. A controller will be designed for this system so that the ball's position can be manipulated.



SYSTEM PARAMETERS

For this problem, we will assume that the ball rolls without slipping and friction between the beam and ball is negligible. The constants and variables for this example are defined as follows:

PHYSICAL CONSTANTS OF BALL AND BEAM SETUP

M	mass of the ball	0.11 kg
R	radius of the ball	0.015 m
d	lever arm offset	0.03 m
g	gravitational acceleration	9.8 m/s^2
L	length of the beam	1.0 m
J	ball's moment of inertia	9.99e-6 kgm^2
r	ball position coordinate	
alpha α	beam angle coordinate	
theta	servo gear angle	
T_s	Settling time	< 3 seconds
%Overshoot		< 5%

SYSTEM EQUATIONS

The second derivative of the input angle actually affects the second derivative of r. However, we will ignore this contribution. The Lagrangian equation of motion for the ball is then given by the following:

$$0 = \left(\frac{J}{R^2} + m\right)\ddot{r} + mg\sin\alpha - mr\dot{\alpha}^2$$

Linearization of this equation about the beam angle, α = 0 (or assuming α very small) gives us the following linear approximation of the system:

$$\left(\frac{J}{R^2}+m\right)\ddot{r} = -mg\,\alpha$$

The equation which relates the beam angle α to the angle of the gear θ can be approximated as linear by the equation below:

$$\alpha = \frac{d}{L}\theta$$

Substituting this into the previous equation, we get:

$$\left(\frac{J}{R^2}+m\right)\ddot{r} = -mg\,\frac{d}{L}\theta$$

TRANSFER FUNCTION OF BALL AND BEAM SYSTEM

Taking the Laplace transform of the equation above, the following equation is found:

$$\left(\frac{J}{R^2}+m\right)R(s)s^2 = -mg\,\frac{d}{L}\theta(s)$$

Rearranging we find the transfer function

$$P(s) = \frac{R(s)}{\theta(s)} = \frac{1}{s^2}\frac{-mgd}{L(\frac{J}{R^2}+m)} \quad \left[\frac{m}{rad}\right]$$

It should be noted that the above plant transfer function is a double integrator. As such it is marginally stable and will provide a challenging control problem.

LAB TASKS:

1. Determine the transfer function using the equation $P(s) = \frac{1}{s^2}\frac{-mgd}{L(\frac{J}{R^2}+m)}$ and

   parameters m=0.111, R=0.015, g=-9.8, L=1.0, d=0.03, J=9.99e-6
2. Determine the step response.
3. Determine the pole/ zero location of the obtained transfer function
4. Build a Simulink model for the given equation of the ball and beam system in Simulink
5. Determine the step response.

CONCLUSION AND COMMENTS:

LAB SESSION 07:

STABILITY OF A SYSTEM

OBJECTIVE:

To evaluate the stability of Linear Time Invariant Systems (LTI)

THEORY:

Stability is the most important system specification. If a system is unstable, transient response and steady-state errors are moot points. An unstable system cannot be designed for a specific transient response or steady-state error requirement. What, then, is stability? There are many definitions for stability, depending upon the kind of system or the point of view. In this section, we limit ourselves to linear, time-invariant systems. Computer can be used to determine the stability by calculating the root locations of the denominator of the closed-loop transfer function. Today some hand-held calculators can evaluate the roots of a polynomial. There is, however, another method to test for stability without having to solve for the roots of the denominator.

STABLE SYSTEM

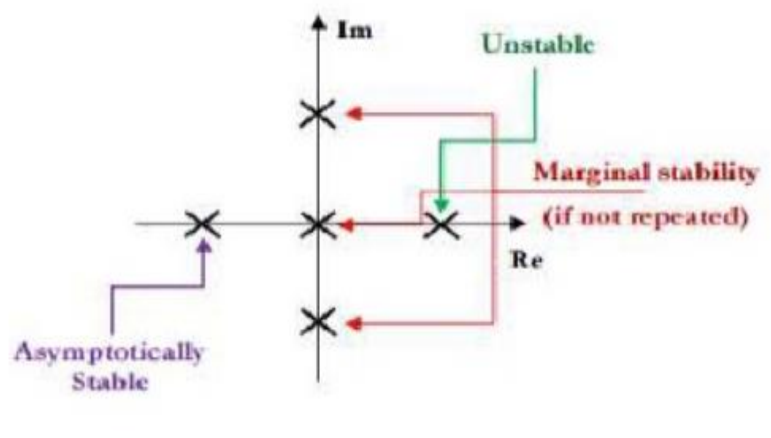
All the roots of the characteristic polynomial lie in the left half plane$(Re(s) < 0)$.

MARGINALLY STABLE SYSTEM

No solution grows unbounded but some do not decay$(Re(s) \leq 0)$.

UNSTABLE SYSTEM

All the roots of the characteristic polynomial lie in the Right half plane$(Re(s) > 0)$.



Stability of a system

POLES AND ZEROS OF SYSTEM:

To obtain the poles and zeros of the system use the MATLAB command "poles" and "zeros" respectively. You can also use MATLAB command "pzmap" to obtain the same.

<u>LAB TASKS:</u>

1. Determine the order of the following systems.

   $$\mathrm{sys1} = \frac{\mathrm{s} + 10}{(\mathrm{s}+1)(\mathrm{s}+3)(\mathrm{s}+5)}$$

   $$\mathrm{sys2} = \frac{100}{(\mathrm{s}+1)(\mathrm{s}-2)}$$

   $$\mathrm{sys3} = \frac{100}{(s+2i)(s-2i)(s+1)}$$

   $$\mathrm{sys4} = \frac{10}{(s+3)(s-5)}$$

2. Plot pole/zeros of the systems and comment on the stability of the system.

CONCLUSION AND COMMENTS:

LAB SESSION 08:

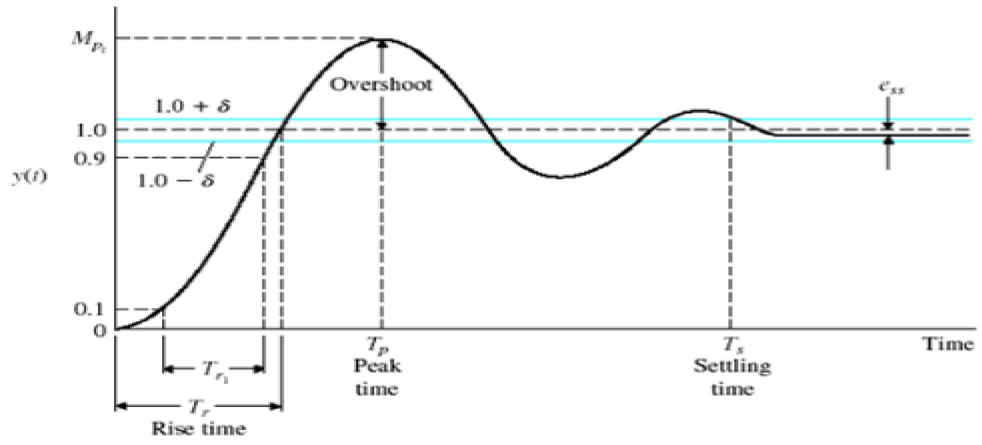
TO MEASURE TIME-DOMAIN PERFORMANCE OF THE SYSTEM

OBJECTIVE:

- To Measure performance of the system through time domain performance parameters using MATLAB

THEORY:

The output response of a system is the sum of two responses: the *forced response* and the *natural response.* Although many techniques, such as solving a differential equation or taking the inverse Laplace transform, enable us to evaluate this output response, these techniques are laborious and time-consuming. Productivity is aided by analysis and design techniques that yield results in a minimum of time. If the technique is so rapid that we feel we derive the desired result by inspection, we sometimes use the attribute *qualitative* to describe the method. The use of poles and zeros and their relationship to the time response of a system is such a technique. Learning this relationship gives us a qualitative "handle" on problems. The concept of poles and zeros, fundamental to the analysis and design of control systems, simplifies the evaluation of a system's response.



Performance parameters of a step response

The performance parameters depicted in the above can be described as follows.

a) Rise Time
Rise time is the time taken by a signal to change from a specified low value to a specified high value.

b) Peak Time

The peak time is the time required for the response to reach the highest peak of the overshoot. Peak time is inversely proportional to the amount of overshoot.

c) Settling time

The time required for the system's output to settle within a percentage of the input amplitude (which is usually taken as 2%) is called settling time. Settling time T_s, is calculated as

$$T_s = \frac{4}{\zeta\omega_o}$$

d) Overshoot

Overshoot is the maximum peak value of the response curve measured from the desired response of the system. It is calculated as

$$P.O = \frac{M_p - fv}{fv} \times 100\%$$

Where M_p is the highest peak magnitude of output and f_v is the final value of the output.

e) Steady-State Error

It is the difference between the desired final output and the actual one. Practically this difference can never be reduced to zero. It is calculated as$abs(dcgain(1 - system))$.

LAB TASKS:

1. Determine the characteristics of first order system when k=1 and t=0.1.
2. Determine the characteristics of second order system when k=1 and w_n=10.

CONCLUSION AND COMMENTS:

LAB SESSION 09:

REDUCTION OF MULTIPLE SUB-SYSTEMS

OBJECTIVE:

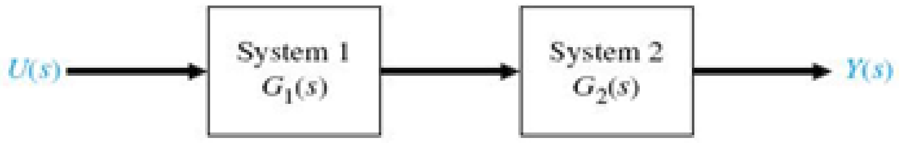
To learn commands in MATLAB that would be used to reduce linear systems block diagram using series, parallel and feedback configuration.

THEORY:

Multiple subsystems are represented in two ways: as block diagrams and as signal-flow graphs. Although neither representation is limited to a particular analysis and design technique, block diagrams are usually used for frequency-domain analysis and design, and signal-flow graphs for state-space analysis.

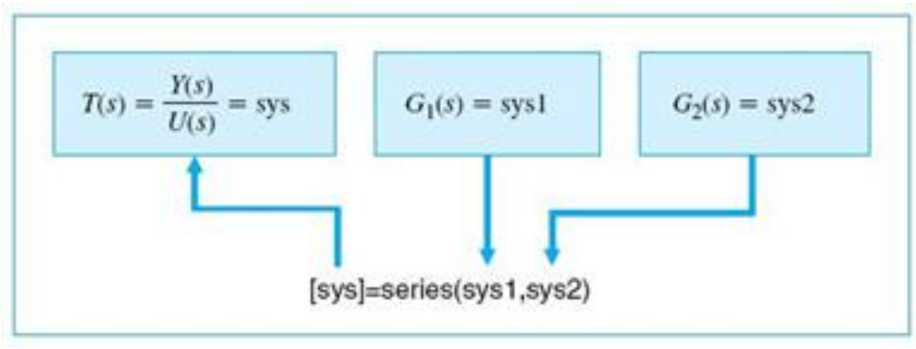
a. SERIES CONFIGURATION:

If the two blocks are connected as shown below then the blocks are said to be in series. It would like multiplying two transfer functions. The MATLAB command for such configuration is "series".



Series configuration

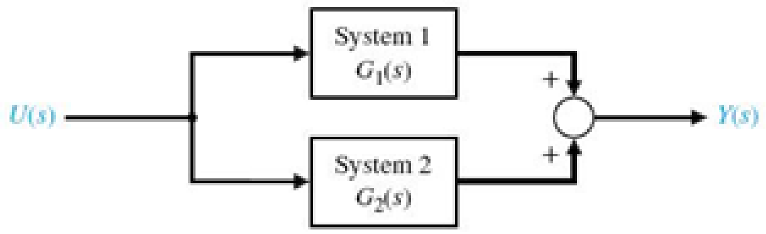
The series command is implemented as shown below:



Implementation of series command

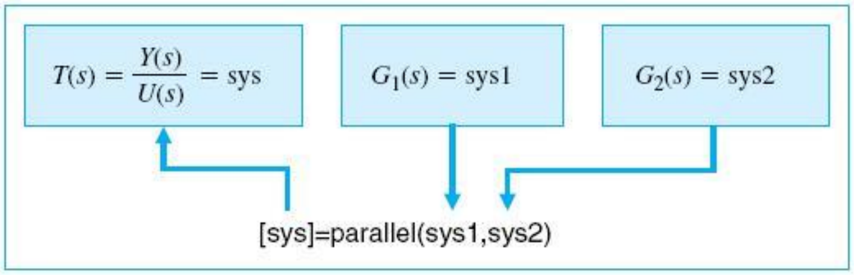
b. PARALLEL CONFIGURATION:

If the two blocks are connected as shown below then the blocks are said to be in parallel. It would like adding two transfer functions.



Parallel configuration

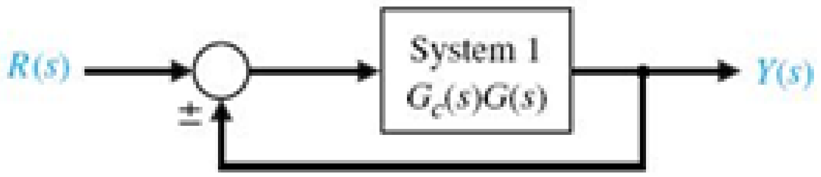
The MATLAB command for implementing a parallel configuration is “parallel” as shown below:


$T(s) = \frac{Y(s)}{U(s)} = \text{sys}$
$G_1(s) = \text{sys1}$
$G_2(s) = \text{sys2}$
[sys]=parallel(sys1,sys2)

Implementation of parallel command

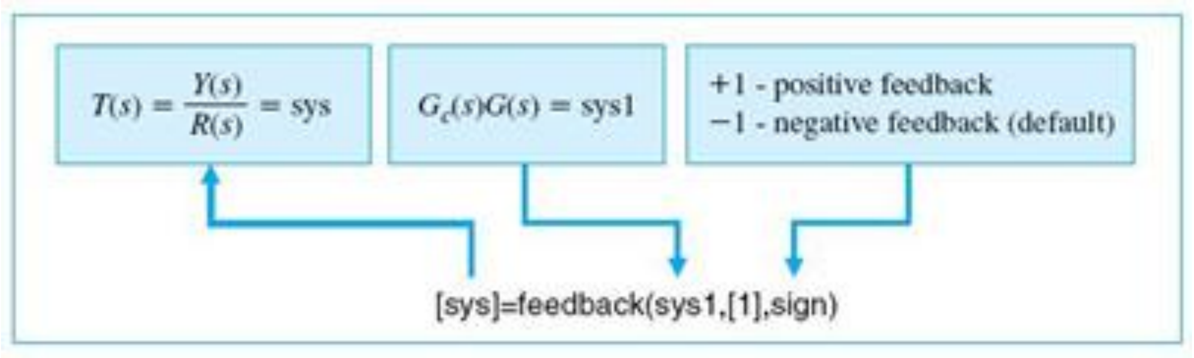
c. FEEDBACK CONFIGURATION:

If the blocks are connected as shown below then the blocks are said to be in feedback. Notice that in the feedback there is no transfer function H(s) defined. When not specified, H(s) is unity. Such a system is said to be a unity feedback system.



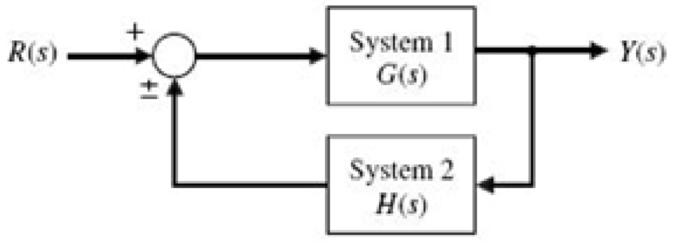
Unity feedback system

The MATLAB command for implementing a feedback system is “feedback” as shown below:


$T(s) = \frac{Y(s)}{R(s)} = \text{sys}$
$G_c(s)G(s) = \text{sys1}$
+1 - positive feedback
−1 - negative feedback (default)
[sys]=feedback(sys1,[1],sign)

Implementation of Unity feedback system

When H(s) is non-unity or specified, such a system is said to be a non-unity feedback system as shown below:



Non-unity feedback system

A non-unity feedback system is implemented in MATLAB using the same “feedback” command as shown:

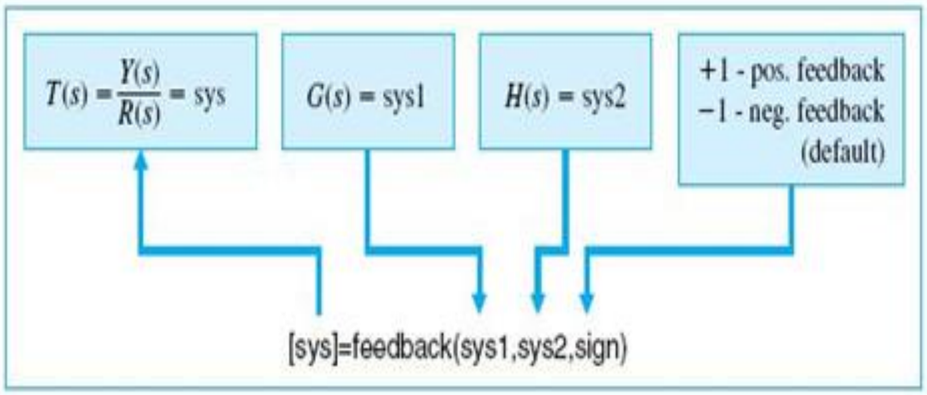

$T(s) = \frac{Y(s)}{R(s)} = \text{sys}$
$G(s) = \text{sys1}$
$H(s) = \text{sys2}$
+1 - pos. feedback
−1 - neg. feedback (default)
[sys]=feedback(sys1,sys2,sign)

Figure: 4.8 Implementation of Non-unity feedback system.

LAB TASKS:

1. Solve the following transfer functions in series, parallel and feedback (positive, negative) configurations.
   $$\mathrm{G1} = \frac{s+7}{(s+700)(s+4)}$$
   $$\mathrm{G2} = \frac{s+3}{(s+6)(s+90)}$$
2. Model the series, parallel and feedback (positive, negative) configurations and their resultants in Simulink.
3. Compare the outputs when input is impulse and step. Comment.
4. Solve the given multiple sub-system in Matlab to find the equivalent transfer function.
5. Model it in Simulink and compare the results with the equivalent transfer function with inputs impulse and step.

CONCLUSION AND COMMENTS:

LAB SESSION 10:

STEADY STATE ERROR

OBJECTIVE:

To study and verify the steady state error of Linear Time Invariant System using SIMULINK.

THEORY:

Steady State Error is the difference between input and output at t=∞ or at large value of time, when our system waveform reaches steady state. It is represented by **e (∞).**For finding steady state error of a particular system, three inputs are used i.e.

1. Step input
2. Ramp input
3. Parabola input

Steady state error can be calculated of only Stable systems. Unstable system's steady state error does not exist as there is no steady state in systems.

Steady state error can be improved by multiplying the system with integration $(1/s\hat{}n)$ where n=1, 2, 3…

PROCEDURE:

- Open Simulink in MATLAB and create a new model
- Open **Simulink Library Browser** where you find all the blocks you may use in Simulink
- To verify the effect of input waveform and system type upon steady-state error.

LAB TASKS:

1. For the negative feedback system of Fig. below:

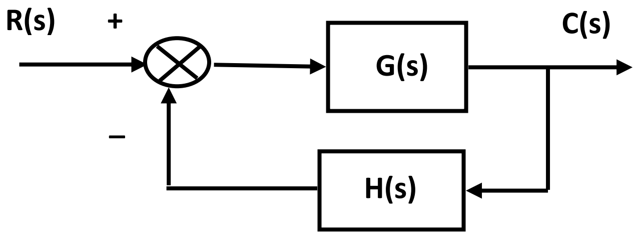


Figure: 6.1 Negative feedback system.

   Where $G(s) = \frac{K\,(s+6)}{(s+4)(s+7)(s+9)(s+12)}$ and H(s) = 1, calculate the steady-state error In terms of K for the following inputs: 5u(t), 5tu(t), and $5t^2u(t)$.

2. Repeat lab task 1 for $G(s) = \frac{K\,(s+6)(s+8)}{s(s+4)(s+7)(s+9)(s+12)}$ and H(s) = 1.
3. Repeat lab task 1 for $G(s) = \frac{K\,(s+1)(s+6)(s+8)}{s^2(s+4)(s+7)(s+9)(s+12)}$ and H(s) = 1.
4. Using Simulink set up the negative feedback system of task 1. Plot on one graph the error signal of the system for an input of 5u(t) and K = 50, 500, 1000 and 5000. Repeat for inputs of 5tu(t) and $5t^2u(t)$.
5. Using Simulink set up the negative feedback system of task 2. Plot on one graph the error signal of the system for an input of 5u(t) and K = 50, 500, 1000 and 5000. Repeat for inputs of 5tu(t) and $5t^2u(t)$.
6. Using Simulink set up the negative feedback system of task 3. Plot on one graph the error signal of the system for an input of 5u(t) and K = 200, 400, 800and 1000. Repeat for inputs of 5tu(t) and $5t^2u(t)$.

CONCLUSION AND COMMENTS:

LAB SESSION 11:

ROOT LOCUS

OBJECTIVE:

To generate the root locus (path of roots) of the system when value of gain (k) changes

THEORY:

In control theory and stability theory, root locus analysis is a graphical method for examining how the roots of a system change with variation of a certain system parameter, commonly gain within a feedback system. The root locus plots the poles of the closed loop transfer function in the complex s plane as a function of a gain parameter. The path that closed loop poles attain when the value of k changes is called Root Locus. The path of root locus can be of any shape. It starts with the open loop pole and ends at the open loop zero. Root locus effects on stability, steady state error. Here k changes from 0 to ∞.At k=o the position of poles of open loop system and closed loop system is same but by increasing k, the position of closed loop pole will change only.

In the previous sessions, we discussed absolute stability of a closed-loop feedback control system using Routh-Hurwitz method. We also studied how the performance of a feedback system can be described in terms of the location of the roots of the characteristic equation in the s-plane. We know that the response of a closed-loop feedback control system can be adjusted to achieve the desired performance by judicious selection of one or more system parameters. It is, therefore, very useful to determine how the roots of the characteristic equation move around the s-plane as we change one parameter. The Root Locus method, as the name suggests, looks at the loci of the roots as some parameter of interest, within the system, is varied. Since we already know that the position of the roots of the characteristic equation strongly influence the step response of the system, we can find values of the parameter which will position the roots appropriately, using the method of Root Locus. This method involves root locus diagrams which the students are expected to have learnt in the theory class. In this lab we will use MATLAB's powerful computing capability to find and trace root locus of a given system.

EXAMPLE

Let we have a unity feedback system as shown. In plotting root loci with MATLAB we need the characteristic equation of this system defined as.

$$1 + K\frac{(s+3)}{s(s+1)(s^2+4s+16)} = 0$$

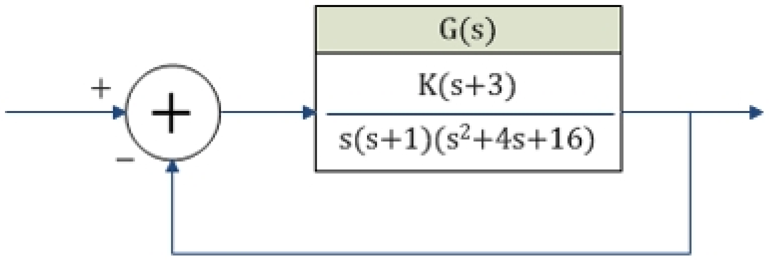


Figure: A unity Feedback System.

MATLAB provides us with $rlocus$ command to compute and display root locus of any system. The syntax of this command is as follows

>>$rlocus(num, den)$

Here num is array of open-loop numerator coefficients and den is array of open-loop denominator coefficients. Using this command, the root locus is plotted on the Figure window as shown in Fig. 7.2. The gain vector K is automatically determined and contains all the gain values for which the closed-loop poles are to be computed. However, we can define vector K as per our own will and provide it to $rlocus$ command as

>>$rlocus\ (num, den, K)$

If this command is invoked with left-hand arguments, like

$[r, k]\ =\ rlocus(num, den)$

$[r, k]\ =\ rlocus(num, den, k)$

$[r, k]\ =\ rlocus(sys)$

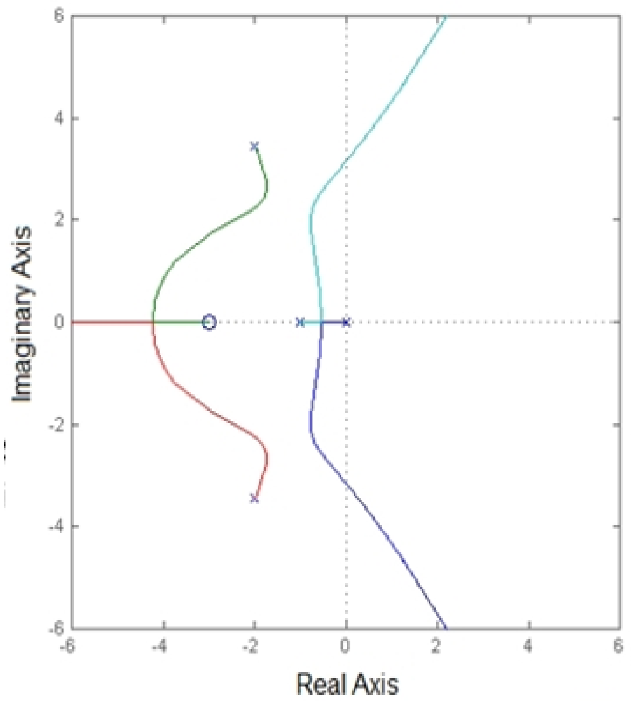


Figure: Root Locus of the system shown in Figure: 7.1.

LAB TASKS:

1. Find the loop gain and poles position by plotting the root locus of the following system with negative feedback.

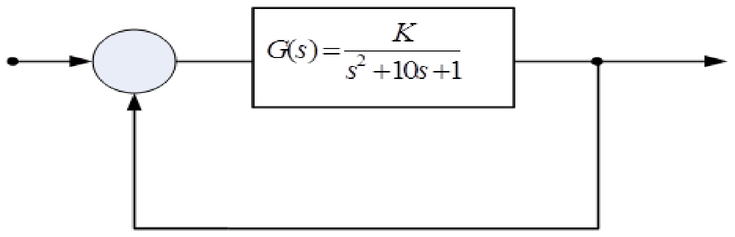


Figure 7.3: Block Diagram of the closed loop system.

2. If the open-loop system is G(s) $=\frac{K(s+1.5)}{s(s+0.5)(s+10)}$, estimate the percent overshoot at the following values of gain, K: 20, 50, 85, 200, and 700. Sketch the root locus for the system.

CONCLUSION AND COMMENTS:

LAB SESSION 12:

FREQUENCY RESPONSE METHOD

OBJECTIVE:

6. To plot the frequency response of any given system precisely using MATLAB.
7. Analyzing Bode plots and Nyquist plots.
8. Determining the phase margins and gain margins.

THEORY:

FREQUENCY RESPONSE OF THE SYSTEM

Frequency response is the quantitative measure of the output spectrum of a system or device in response to a stimulus, and is used to characterize the dynamics of the system. It is a measure of magnitude and phase of the output as a function of frequency, in comparison to the input the effectiveness with which a circuit, device, or system processes and transmits signals fed into it, as a function of the signal frequency. Also called frequency-response curve .a graph of frequency response, with signal amplitude or gain plotted against frequency.

GAIN MARGIN

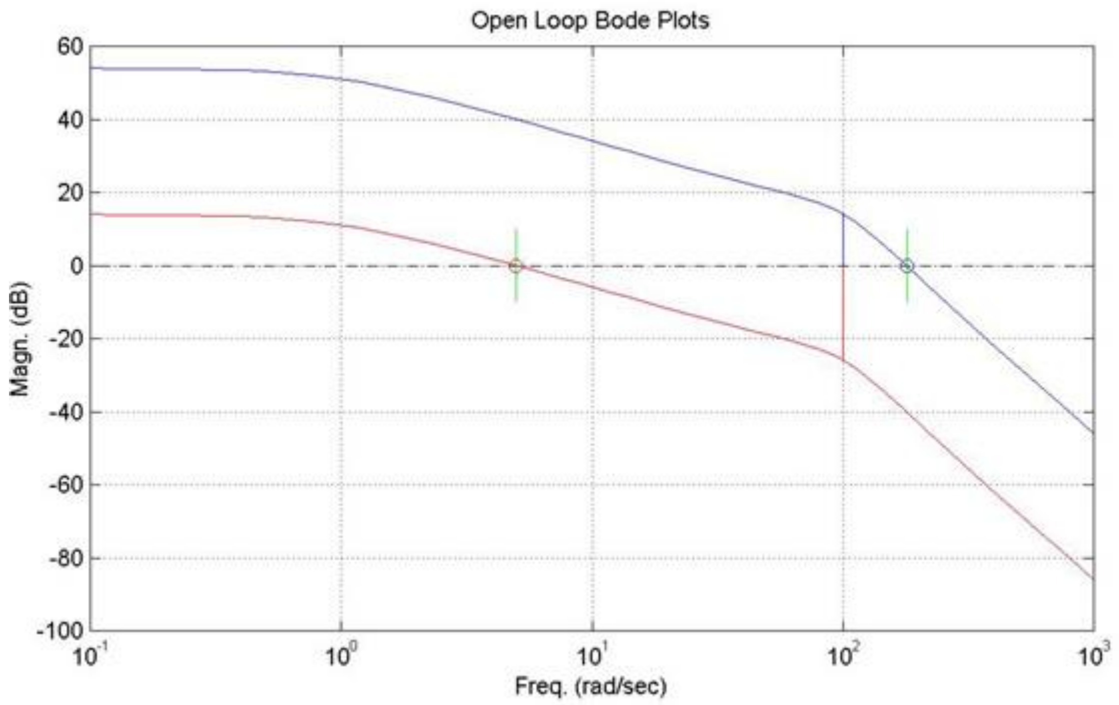
Find the frequency where the PHASE becomes -180 degrees. On our picture, this is at 100 (rad/sec) (marked with a green 'o'on the lower plot). Find the GAIN, **G (in dB)**, at this SAME FREQUENCY (from the upper plot). Then, we define the GAIN MARGIN as:

Gain Margin = **0 - G dB**

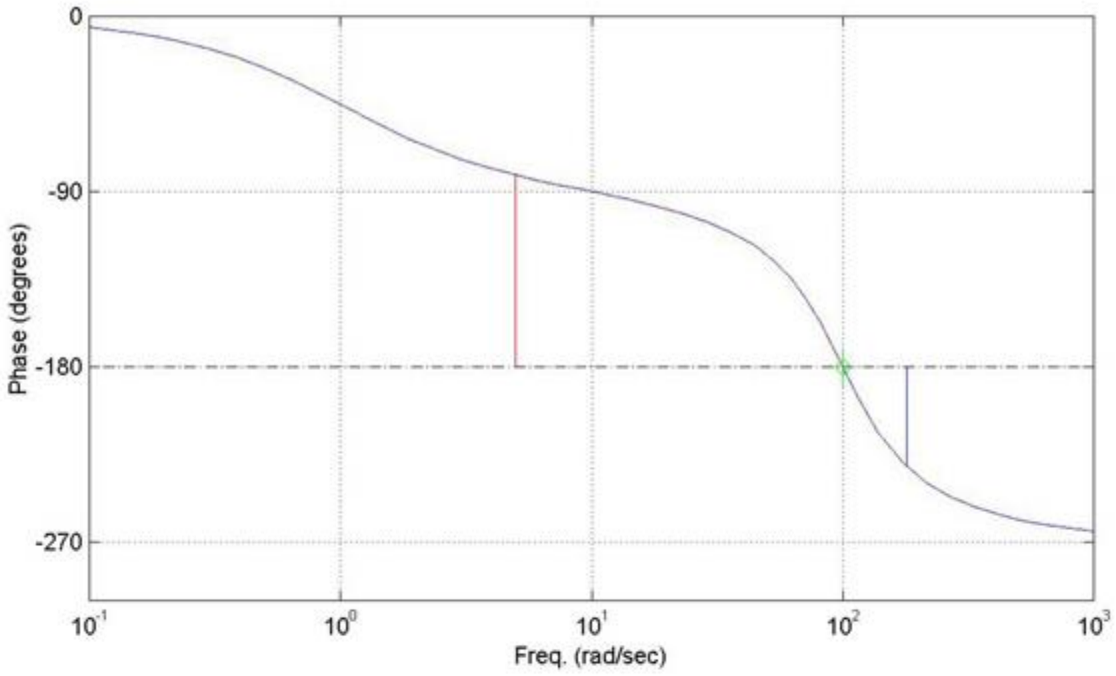
(Note that G is in dB here... But you may want to convert between dB and magnitude as a ratio. To covert magnitude, M, to gain in decibels (dB), G, you use G=20*log10(M). To convert G to M, M=10^(G/20))Gain Margin = **1/M**if you are measuring Magnitude (M) as a ratio (*not* is dB).

PHASE MARGIN

Find the frequency where the GAIN is 0 dB. (This means the output and input amplitudes (magnitudes) are identical at this particular frequency; on the Bode plot, it's where the transfer function crosses 0 dB on the upper [magnitude] plot.)
For the red Bode plot, this happens at about 5 (rad/sec) [marked with a red 'o' in the upper plot]. For the blue Bode plot, the 0 dB crossover occurs at a frequency of around 181 (rad/sec) and is shown with a blue 'o'. Find the PHASE, P (in degrees), at this SAME FREQUENCY (by now looking at the lower plot). (This particular phase is marked in the lower plot at right for both the red and blue transfer functions with lines of corresponding color...) Then, we define the PHASE MARGIN as:

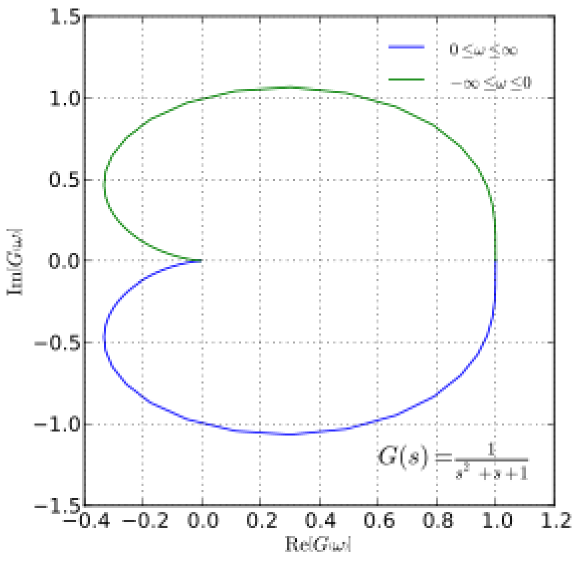


Phase Margin = **+P + 180 degrees**



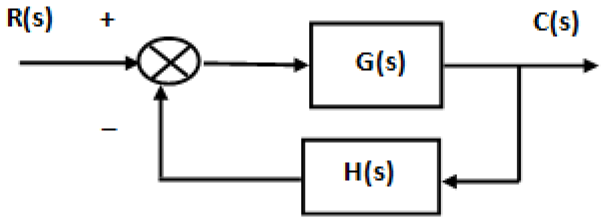
NYQUIST PLOT:

A Nyquist plot is a parametric plot of a frequency response used in automatic control and signal processing. The most common use of Nyquist plots is for assessing the stability of a system with feedback. In Cartesian coordinates, the real part of the transfer function is plotted on the X axis.



LAB TASKS:

1. Consider the following negative feedback system with $G(s) = \frac{K}{s^3+9s^2+30s+40}$ and H(s) = 1.



9. Draw the bode plot of the given system using K= 50, 100, 500.
10. Determine gain margin and phase margin for each value of K.
11. Comment on the differences.
12. Plot the Nyquist response of the given system using K=50, 100, 500.

2. Consider the following unity feedback system having a forward path transfer function

$$G(s) = \frac{2s^2+5s+1}{s^2+2s+3}.$$

13. Draw the bode plot of the given system.
14. Plot the Nyquist response of the given system.

3. Consider the following unity feedback system having a forward path transfer function

$$G(s) = \frac{1}{s(s+1)(s+2)}.$$

15. Draw the bode plot of the given system.
16. Plot the Nyquist response of the given system.

CONCLUSION AND COMMENTS:

LAB SESSION 13:

PID CONTROLLER DESIGN FOR DC MOTOR SPEED

OBJECTIVE:

- Understanding PID controllers
- Using PID controllers to meet design specifications.

THEORY:

PROPORTIONAL CONTROLLERS:

We cannot use these **types of controllers** at anywhere and with each type controllers, there are certain conditions that must be fulfilled. With **proportional controllers** there are two conditions and these are written below:

1. Deviation should not be large; it means there should be less deviation between the input and output.
2. Deviation should not be sudden.

Now we are in a condition to discuss proportional controllers, as the name suggests in a proportional controller the output (also called the actuating signal) is directly proportional to the error signal. Now let us analyze proportional controller mathematically. As we know in proportional controller output is directly proportional to error signal, writing this mathematically we have,

$$A(t) \propto e(t)$$

Removing the sign of proportionality we have,

$$A(t) = K_p \times e(t)$$

Where, K_p is proportional constant also known as controller gain. It is recommended that K_p should be kept greater than unity. If the value of K_p is greater than unity (>1), then it will amplify the error signal and thus the amplified error signal can be detected easily.

INTEGRAL CONTROLLERS:

As the name suggests in **integral controllers** the output (also called the actuating signal) is directly proportional to the integral of the error signal. Now let us analyze integral controller mathematically. As we know in an integral controller output is directly proportional to the integration of the error signal, writing this mathematically
we have,

$$A(t) \propto \int_0^t e(t)dt$$

Removing the sign of proportionality we have,

$$A(t) = K_i \times \int_0^t e(t)dt$$

Where, K_i is integral constant also known as controller gain. Integral controller is also known as reset controller.

DERIVATIVE CONTROLLERS:

We never use **derivative controllers** alone. It should be used in combinations with other modes of controllers because of its few disadvantages which are written below:

1. It never improves the steady state error.
2. It produces saturation effects and also amplifies the noise signals produced in the system.

Now, as the name suggests in a derivative controller the output (also called the actuating signal) is directly proportional to the derivative of the error signal. Now let us analyze derivative controller mathematically. As we know in a derivative controller output is directly proportional to the derivative of the error signal, writing this
mathematically we have,

$$A(t) \propto \frac{de(t)}{dt}$$

Removing the sign of proportionality we have,

$$A(t) = K_d \times \frac{de(t)}{dt}$$

Where, K_d is proportional constant also known as controller gain. Derivative controller is also known as rate controller.

EFFECTS OF USING THESE CONTROLLERS ON THE SYSTEM:

The general effects of each controller parameter (K_p, K_d, K_i) on a closed-loop system are summarized in the table below. Note, these guidelines hold in many cases, but not all. If you

truly want to know the effect of tuning the individual gains, you will have to do more analysis, or will have to perform testing on the actual system.

CL RESPONSE	**RISE TIME**	**OVERSHOOT**	**SETTLING TIME**	**S-S ERROR**
Kp	Decrease	Increase	Small Change	Decrease
Ki	Decrease	Increase	Increase	Decrease
Kd	Small Change	Decrease	Decrease	No Change

EQUATION FOR THE DC MOTOR SPEED TO BE USED:

$$P_{motor} = \frac{K}{(Js+b)(Ls+R)+K^2}$$

DESIGN SPECIFICATIONS:

- Settling time < 2 seconds
- Overshoot < 5%
- Steady state error < 1%
- Discuss briefly what will be the suitable combination of the controllers used.

LAB TASKS:

1. Use Proportional controller K_p=100 and command ‘C=pid(K_p)’ , determine the step response of closed loop system ‘sys_cl=feedback(C*P_motor,1)’. Comment on the output.

2. Use proportional controller K_p=75, integral controller K_i=1 and derivative controller K_d=1 using command ‘pid(K_p,K_i,K_d)’, determine the step response of the closed loop system. Comment on the output.

3. Use proportional controller K_p=100, integral controller K_i=200 and derivative controller K_d=1 using command ‘pid(K_p,K_i,K_d)’, determine the step response of the closed loop system. Comment on the output.

4. Use proportional controller K_p=100, integral controller K_i=200 and derivative controller K_d=10 using command ‘pid(K_p,K_i,K_d)’, determine the step response of the closed loop system. Comment on the output.

5. Comment if the design specifications are met or not.

CONCLUSION AND COMMENTS:

LAB SESSION 14:

PID CONTROLLER DESIGN FOR MAGNETIC LEVITATION SYSTEM

OBJECTIVE:

- Understanding PID controllers
- Using PID controllers to meet design specifications.

THEORY:

PROPORTIONAL CONTROLLERS:

We cannot use these **types of controllers** at anywhere and with each type controllers, there are certain conditions that must be fulfilled. With **proportional controllers** there are two conditions and these are written below:

3. Deviation should not be large; it means there should be less deviation between the input and output.
4. Deviation should not be sudden.

Now we are in a condition to discuss proportional controllers, as the name suggests in a proportional controller the output (also called the actuating signal) is directly proportional to the error signal. Now let us analyze proportional controller mathematically. As we know in proportional controller output is directly proportional to error signal, writing this mathematically we have,

$$A(t) \propto e(t)$$

Removing the sign of proportionality we have,

$$A(t) = K_p \times e(t)$$

Where, K_p is proportional constant also known as controller gain. It is recommended that K_p should be kept greater than unity. If the value of K_p is greater than unity (>1), then it will amplify the error signal and thus the amplified error signal can be detected easily.

INTEGRAL CONTROLLERS:

As the name suggests in **integral controllers** the output (also called the actuating signal) is directly proportional to the integral of the error signal. Now let us analyze integral controller mathematically. As we know in an integral controller output is directly proportional to the integration of the error signal, writing this mathematically
we have,

$$A(t) \propto \int_0^t e(t)dt$$

Removing the sign of proportionality we have,

$$A(t) = K_i \times \int_0^t e(t)dt$$

Where, K_i is integral constant also known as controller gain. Integral controller is also known as reset controller.

DERIVATIVE CONTROLLERS:

We never use **derivative controllers** alone. It should be used in combinations with other modes of controllers because of its few disadvantages which are written below:

3. It never improves the steady state error.
4. It produces saturation effects and also amplifies the noise signals produced in the system.

Now, as the name suggests in a derivative controller the output (also called the actuating signal) is directly proportional to the derivative of the error signal. Now let us analyze derivative controller mathematically. As we know in a derivative controller output is directly proportional to the derivative of the error signal, writing this
mathematically we have,

$$A(t) \propto \frac{de(t)}{dt}$$

Removing the sign of proportionality we have,

$$A(t) = K_d \times \frac{de(t)}{dt}$$

Where, K_d is proportional constant also known as controller gain. Derivative controller is also known as rate controller.

EFFECTS OF USING THESE CONTROLLERS ON THE SYSTEM:

The general effects of each controller parameter (K_p, K_d, K_i) on a closed-loop system are summarized in the table below. Note, these guidelines hold in many cases, but not all. If you

truly want to know the effect of tuning the individual gains, you will have to do more analysis, or will have to perform testing on the actual system.

CL RESPONSE	**RISE TIME**	**OVERSHOOT**	**SETTLING TIME**	**S-S ERROR**
Kp	Decrease	Increase	Small Change	Decrease
Ki	Decrease	Increase	Increase	Decrease
Kd	Small Change	Decrease	Decrease	No Change

EQUATIONS FOR THE MAGNETIC LEVITATION ARE TO BE USED:

$$\frac{77.8421}{0.0311s^2 - 30.5250}$$

DESIGN SPECIFICATIONS:

- Settling time minimum
- Overshoot minimum
- Steady state error minimum
- Discuss briefly what will be the suitable combination of the controllers used.

LAB TASKS:

1. Use PID block in Simulink to set parameters as K_p=0.5, K_i=0, K_d=0, determine the step response of closed loop system. Comment on the output.

2. Use PID block in Simulink to set parameters as K_p=0.8, K_i=0, K_d=0.05, determine the step response of closed loop system. Comment on the output.

3. Use PID block in Simulink to set parameters as K_p=0.8, K_i=0.1, K_d=0.05, determine the step response of closed loop system. Comment on the output.

4. Use PID block in Simulink to set parameters as K_p=1.5, K_i=0.03, K_d=15, determine the step response of closed loop system. Comment on the output.
5. Comment if the design specifications are met or not.

CONCLUSION AND COMMENTS:

LAB SESSION 15:

PID CONTROLLER DESIGN FOR BALL AND BEAM SYSTEM

OBJECTIVE:

- Understanding PID controllers
- Using PID controllers to meet design specifications.

THEORY:

PROPORTIONAL CONTROLLERS:

We cannot use these **types of controllers** at anywhere and with each type controllers, there are certain conditions that must be fulfilled. With **proportional controllers** there are two conditions and these are written below:

5. Deviation should not be large; it means there should be less deviation between the input and output.
6. Deviation should not be sudden.

Now we are in a condition to discuss proportional controllers, as the name suggests in a proportional controller the output (also called the actuating signal) is directly proportional to the error signal. Now let us analyze proportional controller mathematically. As we know in proportional controller output is directly proportional to error signal, writing this mathematically we have,

$$A(t) \propto e(t)$$

Removing the sign of proportionality we have,

$$A(t) = K_p \times e(t)$$

Where, K_p is proportional constant also known as controller gain. It is recommended that K_p should be kept greater than unity. If the value of K_p is greater than unity (>1), then it will amplify the error signal and thus the amplified error signal can be detected easily.

INTEGRAL CONTROLLERS:

As the name suggests in **integral controllers** the output (also called the actuating signal) is directly proportional to the integral of the error signal. Now let us analyze integral controller mathematically. As we know in an integral controller output is directly proportional to the integration of the error signal, writing this mathematically
we have,

$$A(t) \propto \int_0^t e(t)dt$$

Removing the sign of proportionality we have,

$$A(t) = K_i \times \int_0^t e(t)dt$$

Where, K_i is integral constant also known as controller gain. Integral controller is also known as reset controller.

DERIVATIVE CONTROLLERS:

We never use **derivative controllers** alone. It should be used in combinations with other modes of controllers because of its few disadvantages which are written below:

5. It never improves the steady state error.
6. It produces saturation effects and also amplifies the noise signals produced in the system.

Now, as the name suggests in a derivative controller the output (also called the actuating signal) is directly proportional to the derivative of the error signal. Now let us analyze derivative controller mathematically. As we know in a derivative controller output is directly proportional to the derivative of the error signal, writing this
mathematically we have,

$$A(t) \propto \frac{de(t)}{dt}$$

Removing the sign of proportionality we have,

$$A(t) = K_d \times \frac{de(t)}{dt}$$

Where, K_d is proportional constant also known as controller gain. Derivative controller is also known as rate controller.

EFFECTS OF USING THESE CONTROLLERS ON THE SYSTEM:

The general effects of each controller parameter (K_p, K_d, K_i) on a closed-loop system are summarized in the table below. Note, these guidelines hold in many cases, but not all. If you

truly want to know the effect of tuning the individual gains, you will have to do more analysis, or will have to perform testing on the actual system.

CL RESPONSE	**RISE TIME**	**OVERSHOOT**	**SETTLING TIME**	**S-S ERROR**
Kp	Decrease	Increase	Small Change	Decrease
Ki	Decrease	Increase	Increase	Decrease
Kd	Small Change	Decrease	Decrease	No Change

DESIGN SPECIFICATIONS:

- Settling time < 3 seconds
- Overshoot < 5%
- Discuss briefly what will be the suitable combination of the controllers used.

<u>LAB TASKS:</u>

TASK 1:

0. Use Proportional controller K_p=1 and command 'C=pid(K_p)' , determine the step response of closed loop system 'sys_cl=feedback(C*P_ball,1)'. Comment on the output.
1. Use K_p=5, 10, 100 and determine the step response. Comment on the variations in the response of the system.

TASK 2:

2. Use proportional controller K_p=10 and derivative controller K_d=10 using command 'pid(K_p,0,K_d)', determine the step response of the closed loop system. Comment on the output.
3. Use K_d=20,40 and determine the step response of the system.

TASK 3:

4. Comment if the design specifications are met or not.

CONCLUSION AND COMMENTS: