Lab 1: Dynamic Simulation using Simulink and MATLAB

Introduction

In this lab you will learn how to use Simulink to model and simulate dynamical systems. Simulink is integrated with MATLAB and uses a block diagram environment to represent dynamic systems. You can think of Simulink as a graphical programming tool. Instead of writing large amounts of code, you can simply drag-and-drop predefined blocks into the model window and connect the inputs and outputs to create a simulation.

Timing, numerical methods, and other settings can be setup in the system configuration before running the simulation in Simulink. Progress of the simulation can be monitored while running, and the final results can be exported to MATLAB for post-processing when complete.

Experiment

Required Steps to Simulate a System:

1. Determine the governing equation(s) of the system.
2. Draw out the block diagram representing the system dynamics on paper.
3. Create the corresponding block diagram in Simulink.
4. Add additional source and sink blocks to manage the inputs and outputs.
5. Setup the simulation parameters and run the simulation.
6. Display and analyze the results.

Steps 1 and 2 should be completed in the Prelab assignment. The remaining steps are explained in the following sections.

Structure of a Block Diagram in Simulink:

An overall block diagram in Simulink typically has the form shown in Figure 1. This figure only represents the general high-level form of a block diagram. It does not include many more detailed blocks within the system and different options for source and sink blocks.

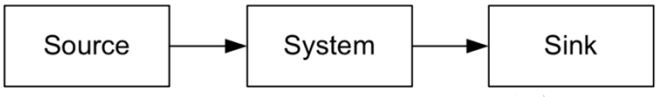

Figure 1: Typical Block Diagram Model in Simulink

The system block defines all the system dynamics you want to simulate. Depending on the complexity of the system, this block may consist of one or more predefined blocks from the Simulink library.

If the system dynamics has a forced input, you will need to include a source block to supply a source which creates the desired input. The Simulink library contains several source blocks to generate a forced input such as a step input, ramp input, sine wave, or square wave.

The output signals of the system can be displayed or stored using a sink block. Again, the Simulink library provides a variety of sink blocks to choose from. For example, the *To Workspace* block stores the signal history of the specific output in the MATLAB workspace.

Exercise 1

This exercise will ask you to model and simulate the spring-mass-damper cart from the Prelab. With the block diagram in your answer to Question 2 of the Prelab assignment, a corresponding block diagram can be created in Simulink. Figure 2 shows what the block diagram will look like in Simulink.

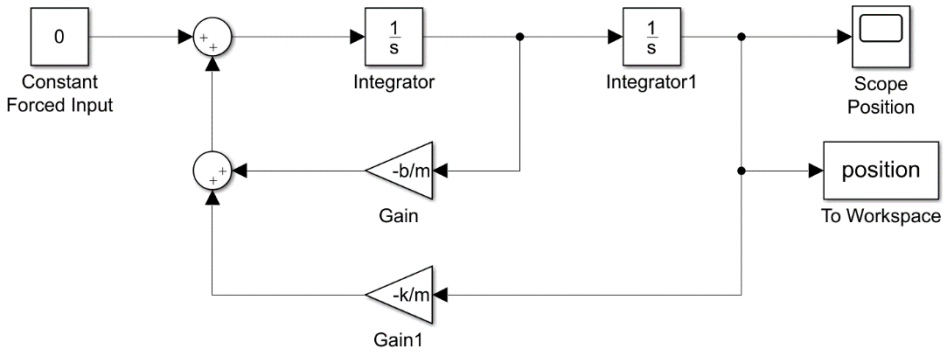

Figure 2: Simulink Block Diagram of Spring-Mass-Damper

Create New Simulink Model:

First run MATLAB, then start Simulink by typing *simulink* in the MATLAB command window.

```
>> simulink
```

This can also be done in MATLAB's home toolbar by clicking *new*, then *Simulink model*. The Simulink start page will open where you can select from a variety of model templates. To begin from scratch, simply click the blank model.

Add Blocks from Libraries:

There are two ways to add blocks from the built-in libraries. 1: Open the library browser  , find the block in the corresponding library, and drag-and-drop the block into your model window; 2: Click in the white space of the model window to invoke a search bar, then search for the block and hit enter. The second method is much faster and more convenient when the block name is known.

Add two sum blocks, two integrator blocks, and two gain blocks to your model. These are found in the *Simulink/Math Operations* and *Continuous* libraries. Arrange the blocks as shown. Click on a block and use CTRL+R and CTRL+SHIFT+R to rotate clockwise and counter-clockwise.

Add scope, to workspace, and constant blocks to your model. These are found in the *Simulink/Sources* and *Sink* libraries. The to workspace block takes the connect signal's history and stores it in the MATLAB workspace when the simulation finishes. The scope block displays the connected signal while running.

Connecting Blocks:

Connections are made at each block's inward and outward arrows. Most connections can be made by clicking and dragging arrows along the desired route. To branch off, right-click and drag from the existing connection. Complete the connections shown in Figure 2.

Set Block Parameters:

Set the $-b/m$ gain block value to -2 by double-clicking on the block. Set the $-k/m$ gain block to -40. The first integrator outputs velocity. Its initial condition should be zero. The second integrator outputs position which should have an initial condition of 100mm. Set its initial condition value to 0.1. Set the constant input to zero $(f(t) = 0)$ to simulate an un-forced system. Change the variable name of the *To Workspace* block to position.

Setup Model Configuration Parameters:

Click the gear icon  to pull up the model configuration parameters. In the Solver settings, change the Solver to be *ode45* instead of *auto*. Leave the *Type* as *Variable-step*. Click the arrow next to *Solver details* to view more settings. Change the *Max step size* to 0.005 and the *Relative tolerance* to $1e$-5. Click OK to apply the changes.

Run Simulation and Plot Results:

Run the simulation by clicking the play button. You can view the signal output while running by clicking the scope block and opening the scope's plot window. The signal output is also available in the MATLAB workspace via the to workspace block. Go to the MATLAB command prompt and type:

```
>> plot(position.time, position.data)
```

Save the Block Diagram:

Save your model as a .slx file in your bench's directory, C:\ME460_SPxx\ABx\Lab1.

Before proceeding, you should become familiar with other blocks available in the libraries. Most blocks used in this lab can be found in *Simulink/Sources*, *Sinks*, *Commonly Used Blocks*, and *Math Operations*. Block functionalities are mostly obvious by name, but to understand its use, double-click the block and then click *help* for a detailed explanation.

Inverse Laplace Transform:

Compare the Simulink simulation with MATLAB's inverse Laplace Transform. Take the Inverse Laplace Transform of $X(s)$ using the impulse() function, where $X(s)$ is the Laplace Transform of position of the mass-spring-damper system. The impulse() function computes the impulse response in the time domain of the given transfer function. Since the Laplace Transform of $\delta(t)$ is 1, this function simply computes the Inverse Laplace Transform $x(t)$ of the given transfer function $X(s)$.

Write the following code in a MATLAB script file and run it multiple times with varying damping constant.

```
m = 1;                    % mass
```

```
k = 40;                          % spring constant
b = 2;                           % damping constant
x0 = 0.1;                        % initial position
v0 = 0;                          % initial velocity
T = linspace(0,10,200);          % time vector
num = [0,m*x0,b*x0+m*v0];        % TF numerator
den = [m, b, k];                 % TF denominator
x = impulse(num,den,T);          % inverse laplace
plot(T,x)                        % plot results
```

Compare the Simulink plots to the plots generate with MATLAB's impulse() function.

Exercise 2

With the block diagram from Question 4 of the Prelab, build a corresponding Simulink model. Note the equations of motion have a nonlinearity. Use the *Trigonometric Function* block from the *Math Operations* library.

This block diagram is very similar to exercise 1 (the previous gains $-k/m$ and $-b/m$ are now $-g/l$ and $-b/ml^2$ as well as a $\sin\theta$ block inserted just before the $-g/l$ gain). This is because both systems are second order (two integrators). Make the necessary modifications to exercise 1's .slx file and save as a new Simulink model. Run the simulation and plot your results.

Now consider a linear approximation of θ for $\sin\theta$. You can comment through a block by clicking on the block and using CTRL+SHIFT+Y. Modify you Simulink model accordingly.

Run the linearized simulation with different initial conditions and compare it to the original nonlinear simulation. You can select all blocks, copy, and paste them to create a second simulation loop in the same Simulink model. Try plotting both linearized and nonlinear results in the same plot.

Exercise 3

Now you will develop nonlinear and linearized Simulink models of a hydraulic motor system similar to what is on the hydraulic bench. This system is shown in Figure 3 and the equations of motion will be described below.

From the setup in Figure 3, there are two states in the system: 1 – the motor speed ω, 2 – the pressure downstream of the valve P_d. The two inputs are the valve command u_v, and the pressure upstream of the valve u_{P_u}. The relationship between torque applied T to the hydraulic motor and the motor's speed ω can be captured well with a linear first order model; see Eq. (1).

$$J\dot{\omega} = T - b\omega \tag{1}$$

J is the motor inertia and b is the friction coefficient. The speed of the hydraulic motor ω and the flow rate through the motor q_m are related by the motor displacement D; see Eq. (2).

$$q_m = D\omega \tag{2}$$

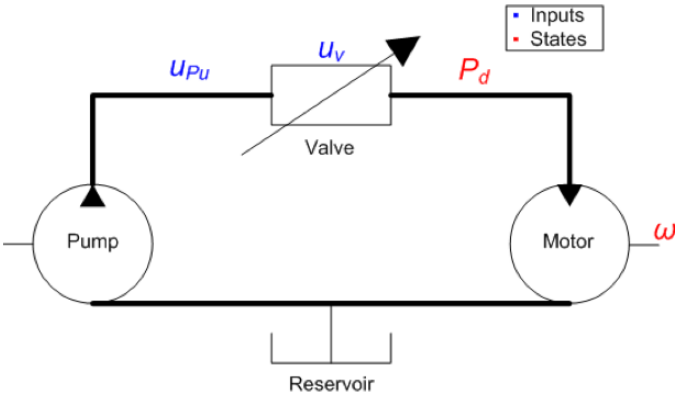

Figure 3: Hydraulic Motor System with Valve Control

Several assumptions have been made such as ignoring the leakage and static friction in the hydraulic motor. To consider leakage, a more complex relationship between flow rate and motor speed is required. These effects play a minor role in the physical system's behavior and are not necessary for deriving an adequate model of the hydraulic motor system. Though it is important to acknowledge what assumptions you are making when modeling a physical system.

The torque applied to the motor T is a function of the pressure drop across the motor (downstream pressure P_d). The downstream pressure P_d has a dynamic relationship with the value flow rate q_v and the motor flow rate q_m. See Eq.s (3) & (4).

$$\dot{P}_d = \frac{\beta}{V}(q_v - q_m) \tag{3}$$

$$T = DP_d \tag{4}$$

V is the volume of oil inside the hose connection the valve to the motor and β is the bulk modulus of the oil (a measure of oil's resistance to compression). When the flow through the valve and the more remain equal, the downstream pressure remains constant.

The last component of the system is the valve. The valve is actuator with a control signal u_v to vary the downstream pressure (and motor speed) for a given upstream pressure u_{P_u}. Eq. (5) relates these values to the flow rate through the valve q_v.

$$q_v = ku_v\sqrt{u_{P_u} - P_d} \tag{5}$$

k is the coefficient relating the valve control signal to valve opening. This equation is known as the orifice equation and can be derived from Bernoulli's equation[1].

[1] White, F.M., Fluid Mechanics, 5th ed. McGraw-Hill, Boston: 2013.

Combine Eq.s (1)-(5) into a system of two nonlinear dynamic equations. Linearized the system of equations about the operating point: u_{v0}, $u_{P_u}0$, P_{d0}, ω_0. Using the parameters from Table 1, build a Simulink model of both nonlinear and linearized versions of the hydraulic motor system. Use the same configuration parameters as the previous exercises. Simulate both versions with an upstream pressure of 2.75 kPa and a constant valve control signal of 2.5 V.

Table 1: Hydraulic Motor System Parameters

Bulk Modulus	β	240 MPa
Hose Volume	V	30 cm^3
Displacement	D	4 cm^3/rad
Friction Coefficient	b	1 Nms
Motor Inertia	J	0.5 kgm^2
Valve Coefficient	k	2
Control Set-Point	u_{v0}	2.5 V
Upstream Pressure	u_{P_u}	2.75 MPa
Initial Downstream Pressure	P_{d0}	0.5 MPa
Initial Speed	ω_0	0 rad/s

Report Questions

1. Compare the plots from Simulink's simulation and from MATLAB's impulse function in Exercise 1. What are the differences? Should they be similar or not?
2. In Exercise 2, at what initial condition does the linearized model poorly approximate the original nonlinear model?
3. How are the observations you made in Question 2 (above) relevant to controller design?
4. In Exercise 3, how do the nonlinear and linearized models compare? How do they compare when changing the upstream pressure?
5. Save your plots and simulation files to your bench's directory C:\ME460_SPxx\ABx\Lab1.