Modeling and Control of Mechatronic Systems

Exercise 1 : Mathematical Modeling and Simulation

Dr Jayantha Katupitiya

Exercise 1

Using Matlab for Simulation

Aim

The aim of this exercise is to learn how to use Matlab scripting language to write programs to simulate a set of state equations. The state equations may represent any system including a set of kinematic equations or dynamic equations. To begin with we will use a system with just one state equation.

If you watched the videos describing the development of the Cruise Controller of a car, then you can easily identify the state equation that describes the system. It is;

$$\boldsymbol{F} = \boldsymbol{M}\dot{\boldsymbol{v}} + \boldsymbol{B}\boldsymbol{v} \tag{1.1}$$

If we rearrange this equation then it becomes;

$$\dot{\boldsymbol{v}} = -\frac{\boldsymbol{B}}{\boldsymbol{M}}\boldsymbol{v} + \frac{\mathbf{1}}{\boldsymbol{M}}\boldsymbol{F} \tag{1.2}$$

What we really want to see is how the value of $\boldsymbol{v}$ behaves as a function of time subjected to $\boldsymbol{F}$ we apply. This requires the integration of the differential equation given in (1.2). Therefore, this exercise is about understanding how to integrate one or more differential equations of this sort. Naturally, when a system is described fully using a mathematical model there going to be a set of these differential equations, one for each state variable, and that set of differential equations is called the state equations of the system. You can use this exact same method to integrate any number of state equations.

We have chosen to use two M-files (the files with extension `.m`), one containing the state equations, also called the mathematical model, and the other using the file containing the mathematical model to carry out the simulation. As a convention, we will use the file names as follows. The file containing the model will have names of the form `XXXX_model.m`. The file that carries out the simulation will have the name `XXXX_simulate.m`.

An example 'model' file (`mtrn3020_first_model.m`) is given below:

```
function dy = mtrn3020_first_model(t,y)
% Initialisation
M = 2000.0;% kg
B = 240.0; %N/(m/s)
```

```
% State variables - only one we have
% [x1]
% x1 = v in m/s.


% control inputs
% F -> Applied Force F(t)
F = 4000; % N

dy = zeros(1,1);

x1 = y(1);

dy(1)=-B/M*x1 + 1/M*F;
end
```

An example 'simulate' file (`mtrn3020_first_simulate.m`) is given below

```
% This file simulates the model described in mtrn9211_first_model.m

options = odeset('RelTol',1e-4,'AbsTol',1e-5);
[t,y] = ode45(@mtrn3020_first_model,[0 150],0, options);
figure(1);
plot(t,y(:,1)*60*60/1000.0,'b');%Speed in km/h
%axis equal;
title('Speed plot');
```

1. Simulate the open loop car model and obtain a plot of speed versus time for:
   (a) A step force of 4000 N input applied at time $t = 0$.
   (b) A step force of 4000 N input applied at time $t = 3$.
2. Implement a proportional controller and adjust the proportional gain to obtain fastest possible response without exceeding the applied force past 4000 N.
3. Implement a PI controller and adjust the gains to obtain best possible response with zero steady state error.
4. Implement a PID controller and tune it to obtain the best possible response.
5. If you now imagine that this car needs to be autonomously controlled without a driver to reach an exact positoin, change the state equations to enable the implementation of a position controller. Implement a proportional controller to control the position accurately.