Lab 1: Simulink, Matlab and the ECP Model 205 and 210 Systems

There are three basic goals to this lab:

1) Review (or learn for the first time) some basic Simulink and Matlab commands. Due to the fact the Simulink drivers for the ECP system are for Matlab 6.5.1, ***you must use Matlab 6.5.1 in the lab***. The Matlab/Simulink files you generate today will be the basis for much of the work we do in lab this quarter, and you will utilize them regularly in your homework.

2) Learn to compute transfer functions for open and closed loop systems and extract information.

3) Learn to connect and operate the ECP model 210 (rectilinear system) or model 205 (torsional system)

Part 1: Some background Matlab commands:

poly If we want to construct a polynomial with roots at -1 and -2, we type the command

```
y = poly([-2 -1])
```

Matlab will respond with the array [1 3 2], which corresponds to the polynomial $s^2 + 3s + 2 = (s+2)(s+1)$

tf To construct the transfer function $G_p(s) = \frac{s^2 + 3s + 2}{s^3}$ we type into Matlab:

```
Gp = tf([1 3 2],[1 0 0 0])
```

pole and ***zero*** To find the poles or zeros of a transfer function, we use the pole or zero command

```
pole(Gp)
```

```
zero(Gp)
```

minreal This command eliminates common poles and zeros from transfer functions. There is an optional argument (tol) that let's you specify how close the pole/zeros are to each other before you decide they are really equal. To utilize this command, you would type

```
G = minreal(G);
```

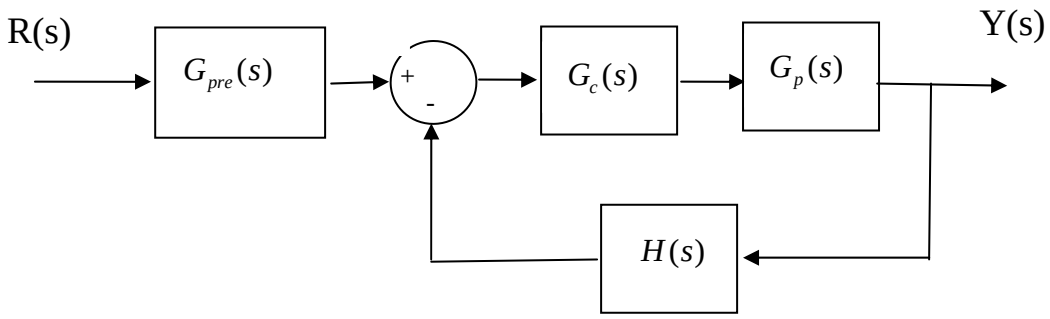
tfdata Sometimes we need to extract the numerator and denominator from the transfer function. This is particularly true when trying to implement a transfer function in Simulink, which expects a numerator and denominator. To do this we use the *tfdata* command, as follows:

```
[num_G,den_G] = tfdata(G,'v');
```

Sometimes we want to know the value of a transfer function $G(s)$ when $s \to 0$ .To do this in Matlab, and assign the value to a variable *final*, type

```
final  = num_G(end)/den_G(end);
```

feedback This is a useful command when we want Matlab to determine the closed loop transfer function for a system with the configuration shown below.



This is the type of configuration we will be utilizing extensively in this class. To find the transfer function from the input R(s) to the output Y(s), we would type

```
Go  = Gpre*feedback(Gc*Gp,H);
```

Note that this command may not clean up pole/zero cancellations, and we may need to utilize **minreal** after this command. If any of these elements is not present, set that value to a 1.

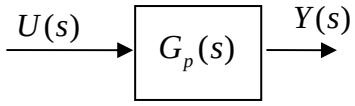
axis Sometimes we want the axes of one particular graph to be different than the other graphs on a page, or we want to examine something up close. To do this we utilize the axis command as follows:

axis([*min_x_value max_x_value min_y_value max_y_value*]);

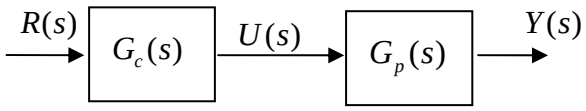
This limits the range of the x-axis from *min_x_value* to *max_x_value*, and limits the range of the y-axis from *min_y_value* to *max_y_value*.

Part 2a: Open loop transfer functions.

An **open loop** system is generally any system where the output is not compared to the input. A simple transfer function is an example of an open loop system:



Here $U(s)$ is the input, $Y(s)$ is the output, and $G_p(s)$ is a mathematical model of the *plant* or the thing we are trying to control. This is the type of open loop system model you used in ECE-200 and ES-205. We can also cascade transfer functions, as follows:



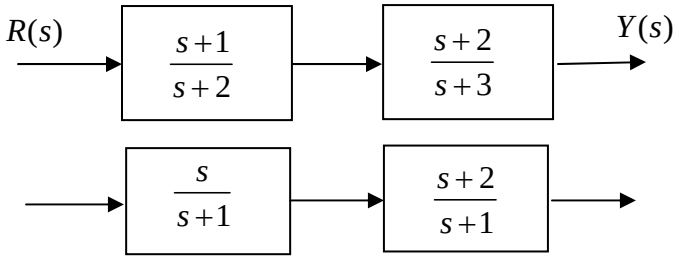
Here we have added a transfer function for the *controller,* $G_c(s)$. This is still an open loop system since we are not comparing the output to the input. The transfer function between the input $R(s)$ and the output $Y(s)$ is the product of $G_p(s)$ and $G_c(s)$. Hence

$$\frac{Y(s)}{R(s)} = G_c(s)G_p(s) = G_o(s)$$

In this course we typically write the transfer function from input to output as $G_o(s)$.

For you to do:

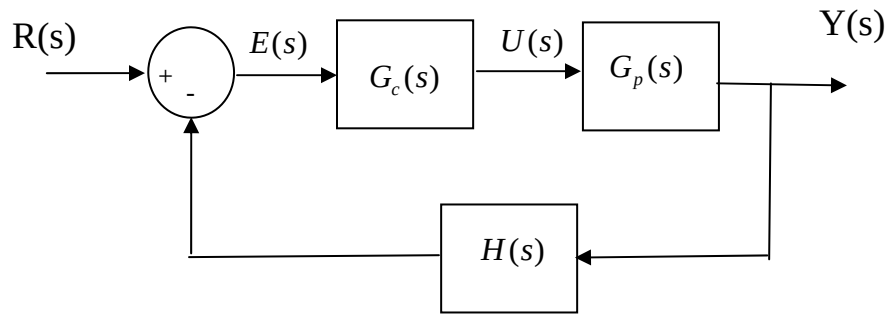
a) Compute the open loop transfer functions for the following two systems by hand
b) Compute the open loop transfer functions for the following two systems using Matlab. Be sure to use the *minreal* command to eliminate pole/zero cancellations.
c) Determine $G_o(0)$ for both of the following systems by hand and using Matlab.



Attach your work to the end of your memo.

Part 2b: Closed loop transfer functions.

A **closed loop** system is generally any system where the output is compared to the input. A typical closed loop control system is shown below. Generally the difference between the input and output is referred to as an error signal, $E(s)$, and this error signal is input to the controller, $G_c(s)$, which then generates the input $U(s)$ to the plant, $G_p(s)$. Sometimes, in order to compare the output to the input (which may be in different units), the output signal is modified by a conditioning transfer function, $H(s)$.

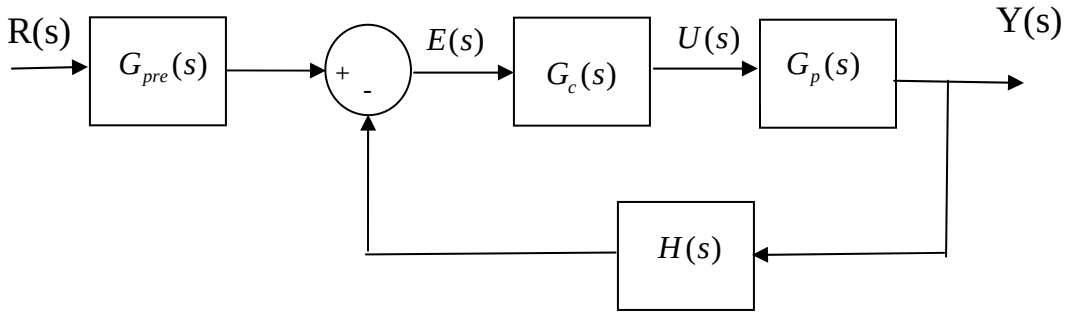


The (closed loop) transfer function between the input and output is given by

$$\frac{Y(s)}{R(s)} = G_o(s) = \frac{G_c(s)G_p(s)}{1 + H(s)G_c(s)G_p(s)}$$

We will use this formula a great deal in this class, so the sooner you learn it, the better off you'll be (we will derive it in a week or so).

Sometimes we include a prefilter, $G_{pre}(s)$, a transfer function after the input and outside of the feedback loop, to modify the transient and steady state behavior.



The transfer function for this system is

$$\frac{Y(s)}{R(s)} = G_o(s) = \frac{G_{pre}(s)G_c(s)G_p(s)}{1 + H(s)G_c(s)G_p(s)}$$

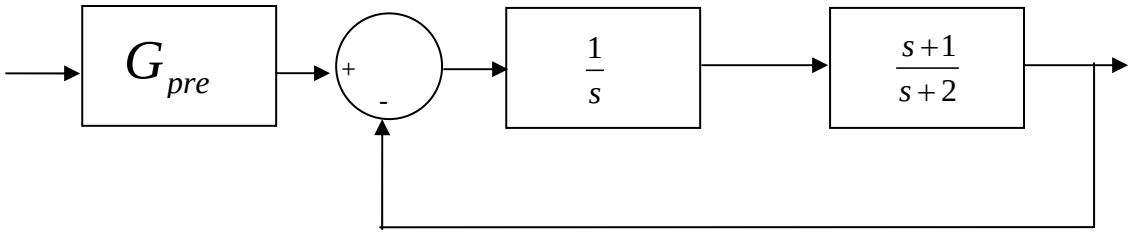
For you to do:

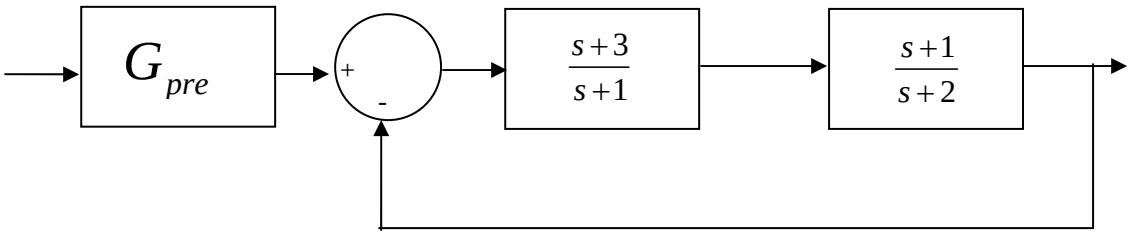
a) Assuming the value of the prefilter is 1, compute the closed loop transfer functions for the following two systems by hand

b) Assuming the value of the prefilter is 1, compute the closed loop transfer functions for the following two systems using Matlab. Be sure to use the *minreal* command to eliminate pole/zero cancellations.

c) Determine, by hand, the (constant) value of the prefilter so that $G_o(0) = 1$.

d) Determine, using Matlab, the (constant) value of the prefiler so that $G_o(0) = 1$.





Attach your work to the end of your memo.

Part 3a - Open Loop Response

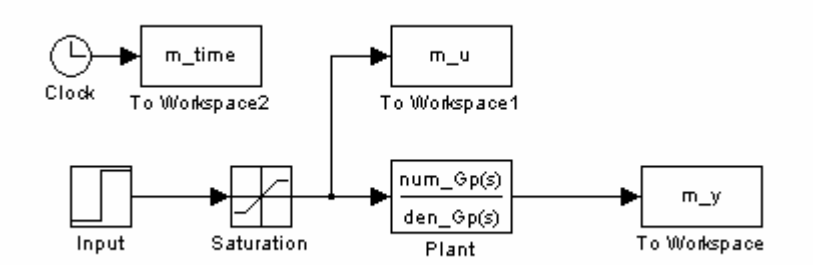
The first thing we will look at in this lab is the *open loop* response of a system. Here we basically assume we have the transfer function of some type of system. This system is usually referred to as the *plant*, and we are trying to improve on the response of the plant.

a) Create a folder for your work. I would suggest making a folder in the ECE-320 folder which identifies you and your lab partner, and then within this folder make a folder entitled something creative like Lab 1.

b) Open Matlab (6.5.1) and set the directory to this folder. Do not dump your stuff in the work (default) directory.

c) Go to the class website (~throne) and download **openloop_driver.m** (Matlab driver file) , **openloop.mdl** (Simulink model file), and **state_model_1dof.mat**.

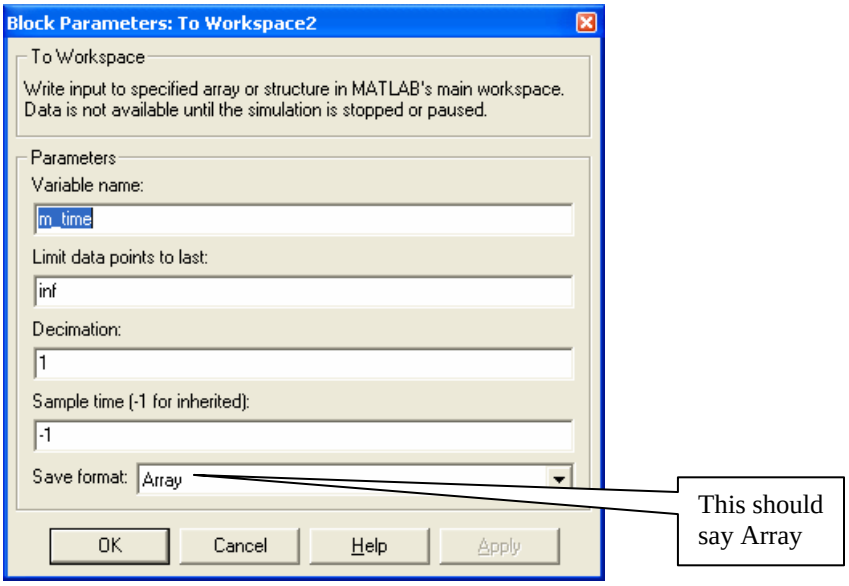
The Simulink file for the open-loop system should look like the following:



In this case, all of the things we are saving (to plot later) have the prefix *m*, such as *m_time*, *m_u*, and *m_y*. This is so we can (in future labs) compare the response of the ***model*** (the *m* things) with the response of the ***real*** system.

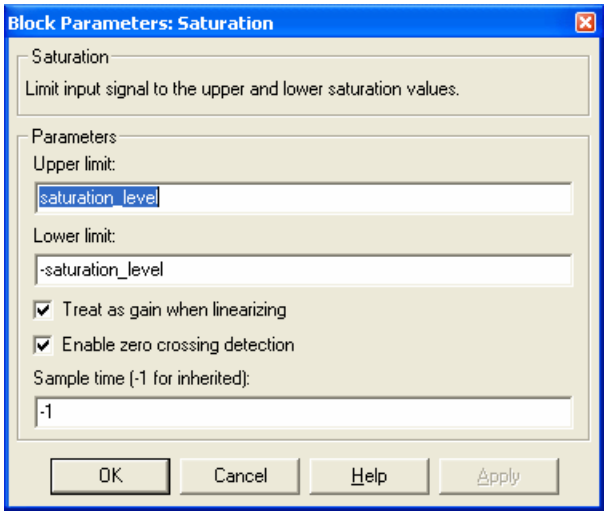
Before we go on, we should review a few of the important pieces of the Simulink file.

All of the items begin saved to the workspace that we want to be able to plot later, like m_time, look like the following:

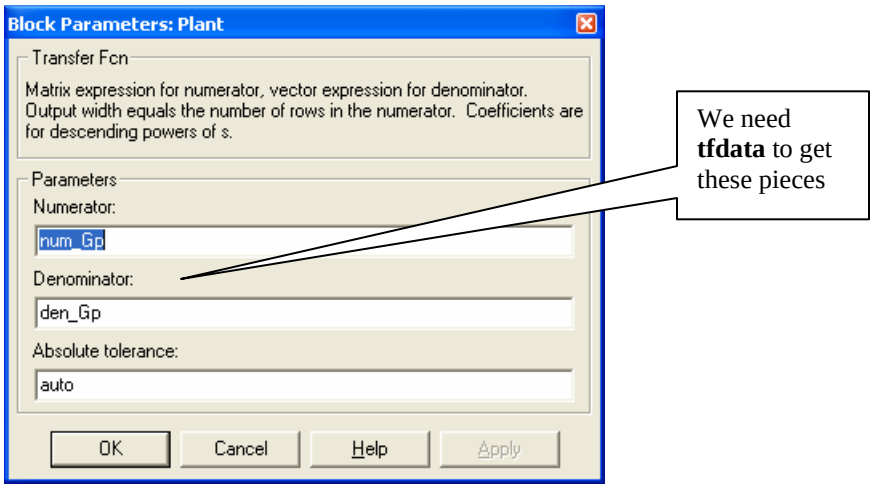


Note in particular that we are saving this data as an ***array***.

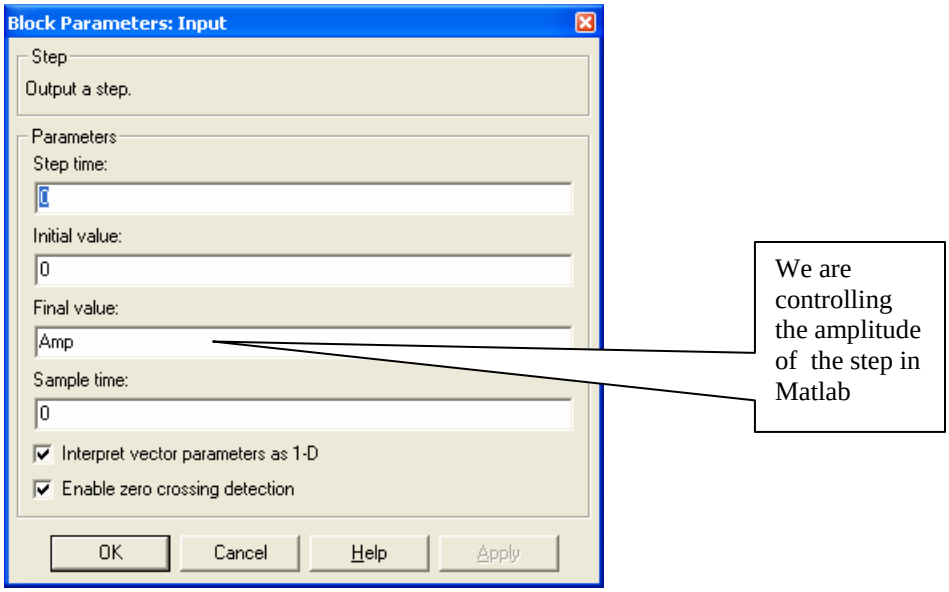
The saturation levels are determined by the ECP device, are coded into the Matlab driver, and are set using the following window (you shouldn't have to change this):



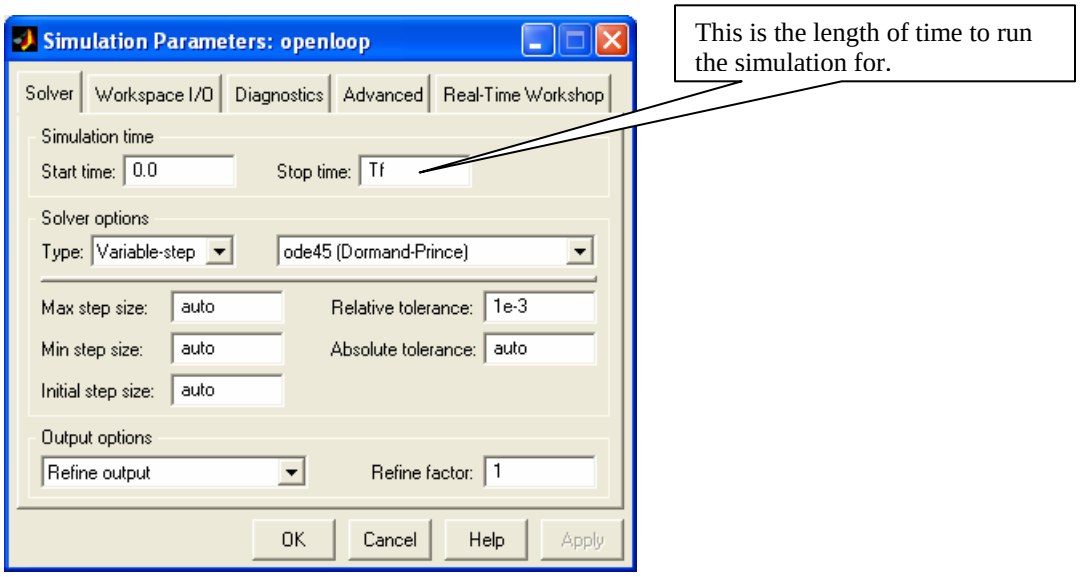
The (plant) transfer function is determined in Matlab, and is entered into the Simulink model as:



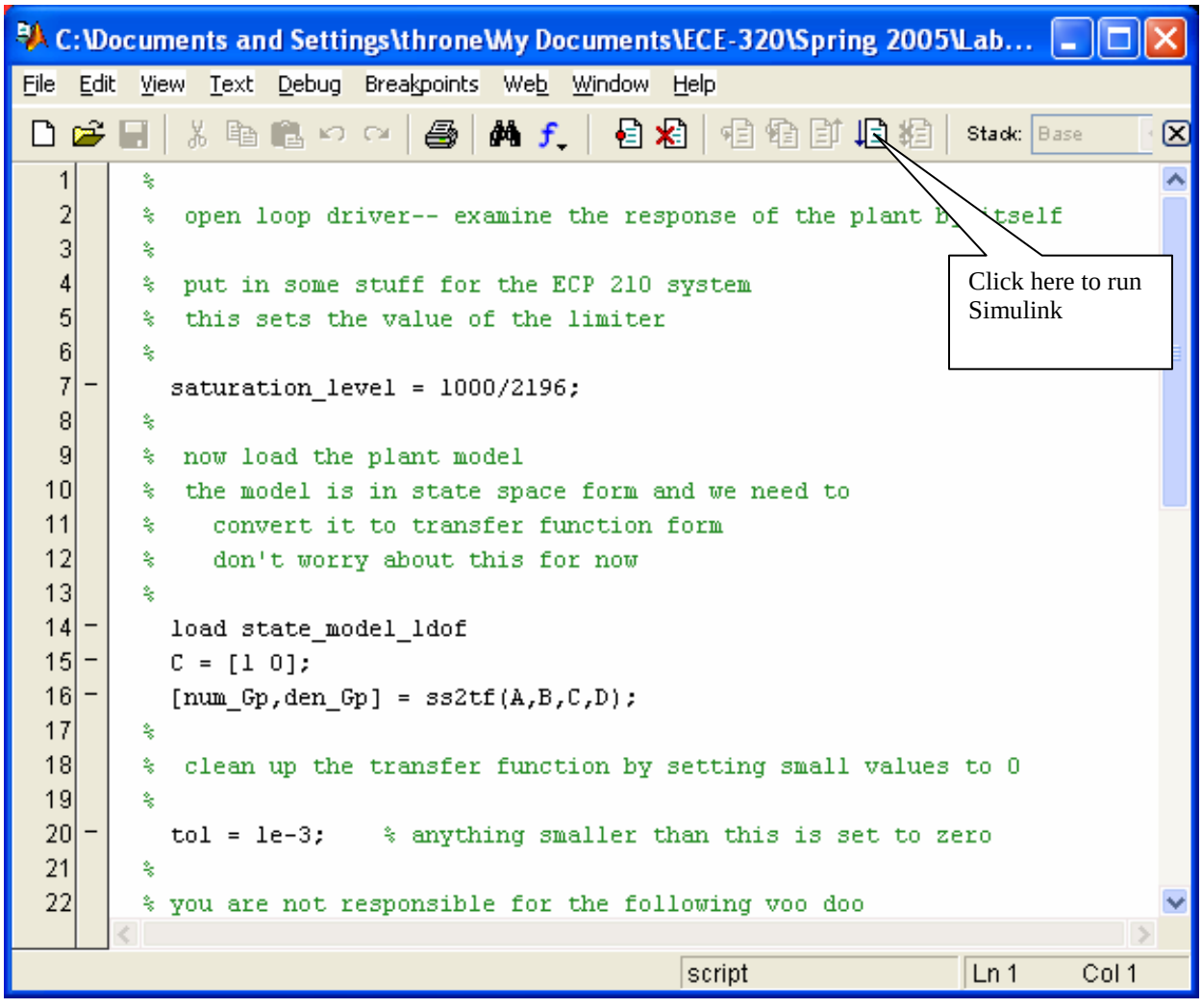
The system input is determined in Matlab, and is entered into the Simulink model by



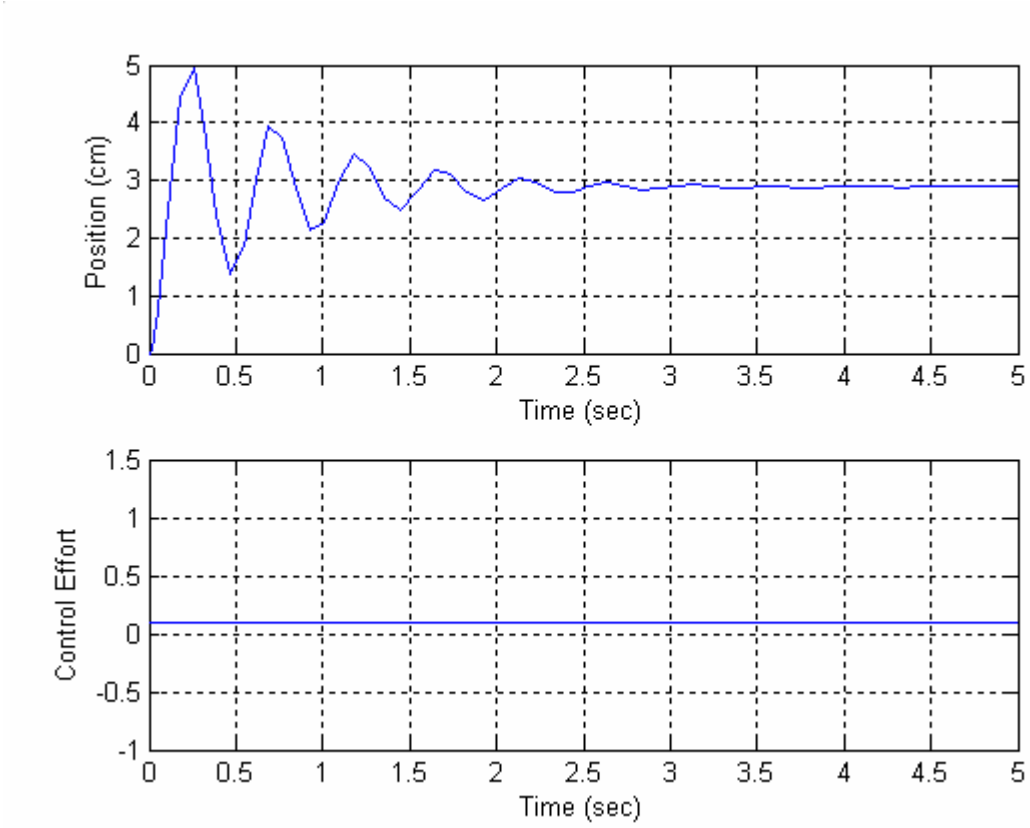
The length of time to run the simulation is also determined by Matlab, and is set in the Simulation Parameters section



Finally, to run the simulation, open **openloop_driver.m** and click on the down arrow:



If you run the m-file **openloop_driver.m**, you should get the following plot:



d) Adjust the input amplitude so that position in steady state is 1cm. Include this plot (and the corresponding input amplitudes) in your memo. Be sure to label each figure and provide a caption.

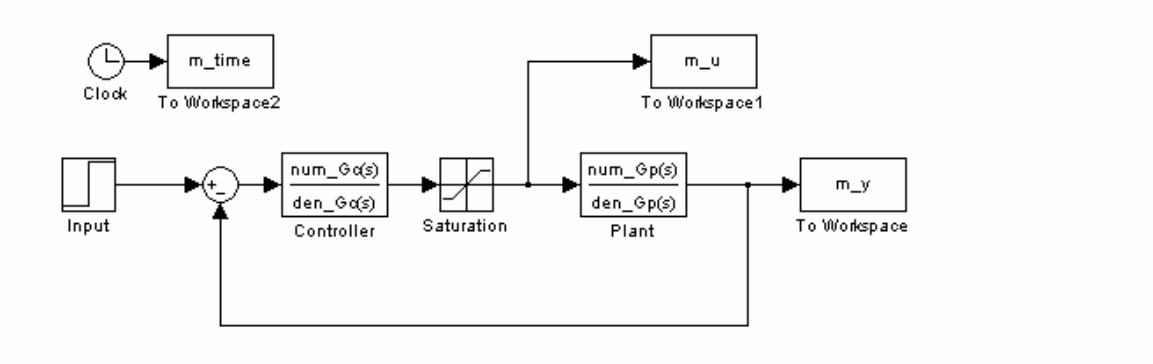
e) By adjusting the input amplitude, can we change the settling time of the system? Can we change the percent overshoot [(peak-steady state)/steady state x 100%]?

This an ***open loop*** system, in that the output signal is never compared to the input signal.

Part 3b - Closed Loop Control

a) Save your **openloop.mdl** file as **closedloop.mdl**. Save your **openloop_driver.m** as **closedloop_driver.m**. Be sure to modify **closedloop_driver.m** so it runs **closedloop.mdl**.

b) Modify **closedloop.mdl** so it looks like the following:



This is a ***closed loop*** system, in that we are feeding back the output and comparing this output to what we wanted the output to be. We have also added a new transfer function, the *controller* or *compensator*, usually denoted as Gc(s).

c) Now we need to try and figure out how to choose our controller. This is one of the main topics of this course, so at this point I'll just tell you.

Lets assume that we want the closed loop transfer function (the transfer function from input to output) to be of the form

$$G_0(s) = \frac{1}{s^2 + 1.8\omega_0 s + \omega_0^2}$$

for some ω_0. To enter this into Matlab, for $\omega_0 = 1$ we could type something like

```
wo = 1;
Go = tf(1,[1 1.8*wo wo^2]);
```

Type this to **closedloop_driver.m**, before the sim command. We will be varying wo so leave it a variable.

d) To determine the controller, we need to solve the equation (we will show where this comes from later in the class)

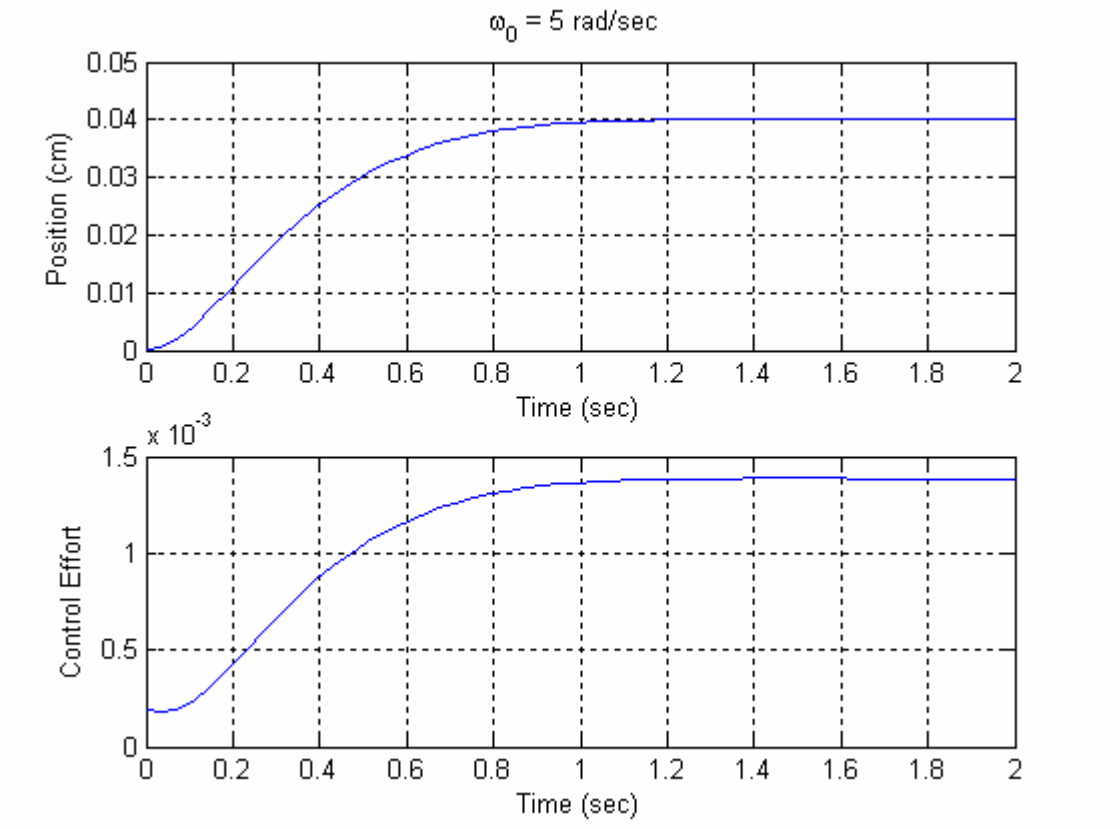
$$G_c(s) = \frac{G_0(s)}{G_p(s)\left[1 - G_0(s)\right]}$$

In Matlab we just have to type

Gc = Go/(Gp*(1-Go));

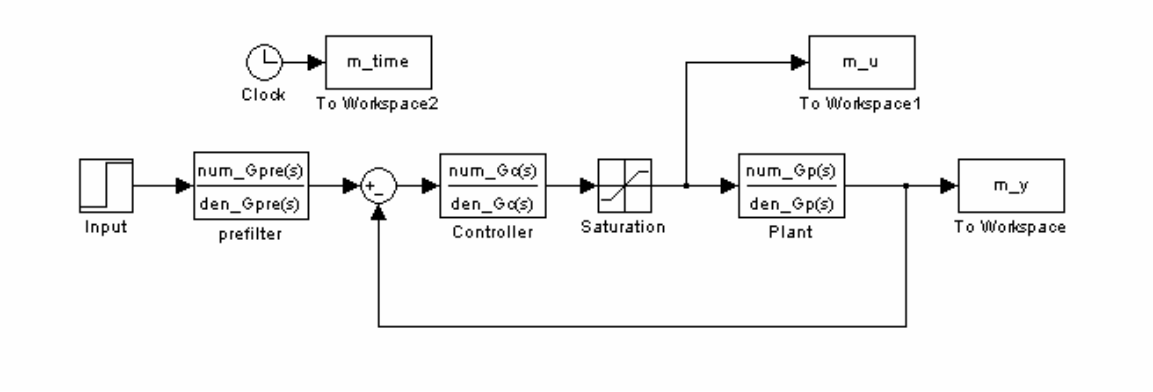
Be sure to eliminate pole/zero cancellations, and extract the numerator and denominator of the controller transfer function so Simulink can use the controller.

e) If you run closedloop_driver.m with wo = 5, Amp = 1, and Tf = 2 you should get the following plot.



f) Run closedloop_driver.m with wo = 10, 50, and 100. Be sure to vary the final time for each graph to include only that portion of the graph that is interesting. Include the plots in your memo. What happens to the settling time as wo is increased? What happens to the control effort? What happens to the steady state value?

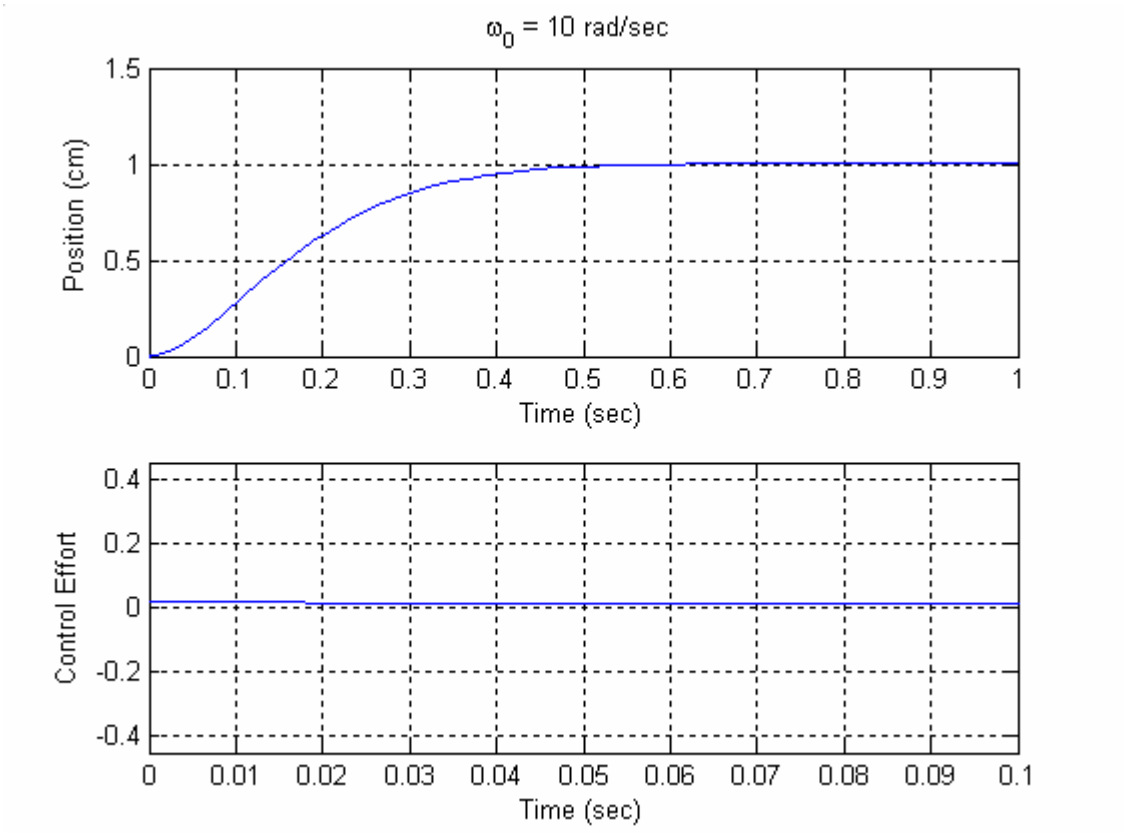
g) We now need to fix the final value, or the steady state error. There are many ways to do this (in this case we could have just modified Go), but we will use a prefilter, since this is what we will do with many of our controllers. Modify **closedloop.mdl** so it looks as follows:



h) For our system, we need to set the prefilter so the closed loop transfer function has a value of 1 as $s \to 0$. You should modify **closedloop_driver.m** to determine the steady state value of the closed loop transfer function (without the prefilter), and then set the prefilter gain to one over this. Although it is easy to determine analytically what this value should be in this case, you need to have Matlab compute it numerically. In this particular case the prefilter is a constant but we will still implement it as a transfer function, to keep it general. The denominator of this transfer function should be set to 1.

i) One of the things you will soon notice for a step input is that the control effort is usually largest just after the step. Since we are mostly concerned about the control effort at this point in time, modify **closedloop_driver.m** using the **axis** command to limit the time we look at the control effort to be from 0 to 0.1 seconds, no matter how long the Simulink simulation is run. Because of the limiter, the control signal will always be between **-saturation_level** and **saturation_level.**

j) If you have done steps (h) and (i) properly, the steady state value of the output should be the same as the steady state value of the input. For wo = 10, Amp =1, and Tf = 1.0, the output of **closedloop_driver.m** should look as follows:



k) Now run the simulation for wo = 20 and 40 rad/sec. What happens to the settling time? What happens to the control effort? Be sure to include these plots in your memo.

l) Run **closedloop_driver.m** again for wo = 50 rad/sec, then for wo = 60 rad/sec. You should notice that once we hit the limiter we start to get fairly weird behavior. This is because our system has suddenly become ***nonlinear***. Whenever we utilize the ECP system we want to be sure to stay within the linear range, if possible. Include both of these plots in your memo.

Your final memo should have a short cover page which answers the various questions that were asked. As attachments you should have each of the plots (these should be included in the document file), each with a figure number and a caption. You should also attach your final version of **closedloop_driver.m** and **closedloop.mdl**. The last page of your memo should have the following page signed by the instructor and attached, verifying you successfully connected to either the ECP model 210 or 205.

Part 4a - ECP Model 210

Be sure the connector box (black and gray box on the top of the shelf) is off before connecting the system. *Do not force the connectors. If they don't seem to fit, ask for help!* You need to read through the handout on the real-time windows and the ECP systems, and use **Model210_Openloop.mdl** with an input of $0.01\sin(4\pi t)$ cm. The instructor needs to sign below verifying your work.

Verified by _____

Part 4b - ECP Model 205

Be sure the connector box (black and gray box on the top of the shelf) is off before connecting the system. *Do not force the connectors. If they don't seem to fit, ask for help!* You need to read through the handout on the real-time windows and the ECP systems, and use **Model205_Openloop.mdl** with an input of $1\sin(4\pi t)$ degrees (Note: The ECP system works in radians, so you will have to convert this!). The instructor needs to sign below verifying your work.

Verified by _____