ESE 406 - SPRING 2012 - HOMEWORK #2

Governing Equations & Simulink Introduction

DUE 25-Jan-2012 (Late Pass 30-Jan-2012)

Textbook Problems Submit solutions to the following problems. Answers are provided so that you can check your work and request help as necessary.

Problem 2.1 Find the equations of motion only for the system in Figure 2.39(c).

Answer:
$$m_1\ddot{x}_1 = -k_1 x_1 - k_2(x_1 - x_2) - b_1(\dot{x}_1 - \dot{x}_2)$$
$$m_2\ddot{x}_2 = F - k_2(x_2 - x_1) - b_1(\dot{x}_2 - \dot{x}_1)$$

$$\ddot{\theta}_1 + \frac{g}{l}\theta_1 + \frac{9}{16}\frac{k}{m}(\theta_1 - \theta_2) = 0$$
$$\ddot{\theta}_2 + \frac{g}{l}\theta_2 + \frac{9}{16}\frac{k}{m}(\theta_2 - \theta_1) = 0$$

Problem 2.3 Answer:

Discussion Problem. We will use Matlab and Simulink to conduct simulations of the "Ping Pong Poise" in-class demonstration. We will start with "Case 2" which is the nominal PID controller we discussed in class. Take a moment to watch the video "Case2.wmv", which shows a capture sequence. The following graph shows the captured position measurements (in counts from the analog to digital converter) for this event (using the 2010 implementation, which was slightly different than what we saw in class this year):

KD=$A0 : KP = $30 : KI = $A0
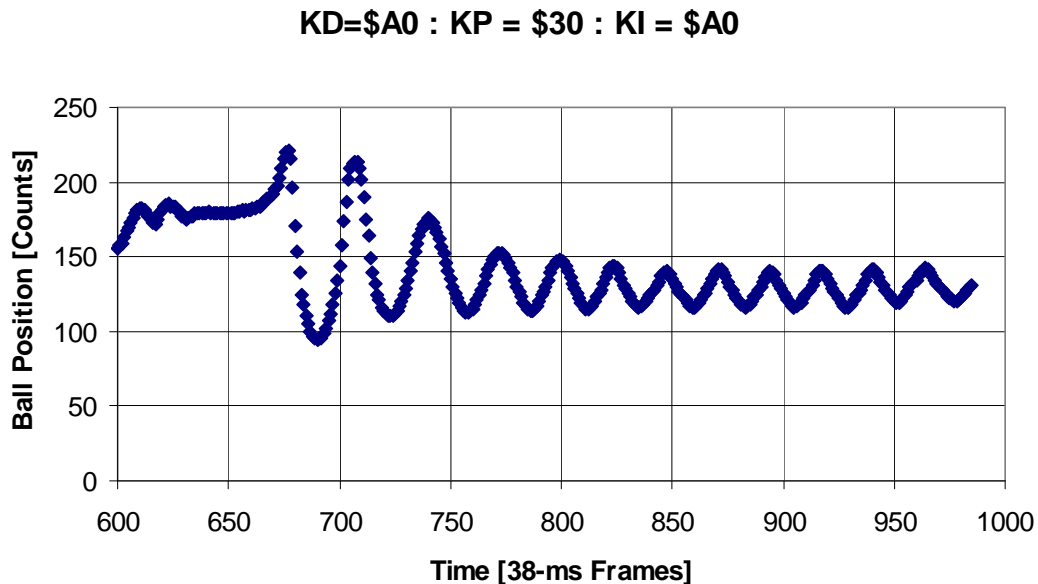
Note that initially, the ball is to the left (in the video) of the center, which is outside of the active sensor range. The ball comes into range between frames 650 and 700. The target position is 127.

Go through the steps below to get some experience working with Matlab and even more experience with Simulink. The last page tells you what you should submit to complete the homework. Please spend as much time as you can playing with Matlab and Simulink, because we will use them a lot this semester. Also, feel free to experiment with the feedback control. You will learn a lot doing these sorts of virtual experiments throughout the semester.

1. Launch Matlab (From a Penn PC, Start→Programs→Mathematics→MatlabR2011b). You should be able to use a more recent version, if available.

2. Using the "Current Directory" tab of the Matlab window, set the current directory to some convenient folder on the computer. I strongly recommend that you create a separate folder for each of homework assignment and save your work for the duration of the semester for reference. Place the `PingPongPoise.mdl` and `PingPongPoiseParameters.m` files that you download from Blackboard in the working directory you have chosen. We will use these files later.

3. In the Matlab Command Window, enter `simulink`. Hereafter, commands entered in the Matlab command Window will be specified with ">>", as

   ```
   >> simulink
   ```

   You should see the simulink library browser:
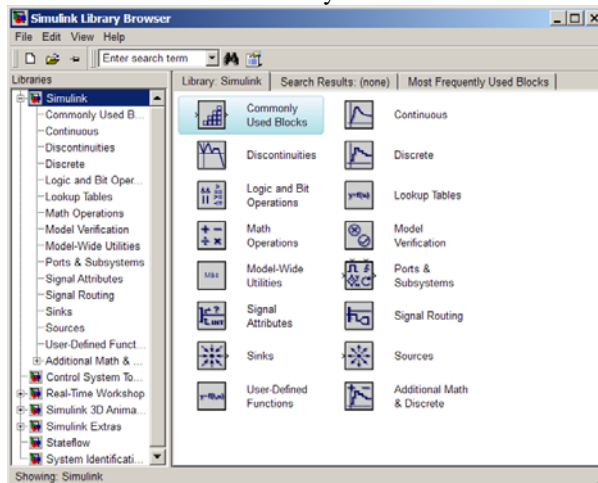
4. In Simulink, open a new model (Simulink Library Browser→File→New Model).

5. Open ⊞ Commonly Used Blocks group in the Simulink Library Browser. Most of the system elements that we need to build our model of the PPP system are here. Let's start by creating a very simple representation of the rocker arm, with the angle of the servo as an input and the position of the ball as an output. We said in class that we expect the acceleration of the ball to be proportional to the servo angle. Let's call the constant of proportionality "A" for now. We'll define a numerical value later. So, begin by dragging a ▷ Gain node into your model. Double click on the node and in the text field labeled "Gain" enter "A".

6. You should now save your model (Simulink→File→Save) in the working directory. You should save your model often while you are working, in case something crashes or goes wrong. This is a very good habit with Simulink; I always try to save my models before running the simulation. Call the model something like "HW01". Your model should now look like this:
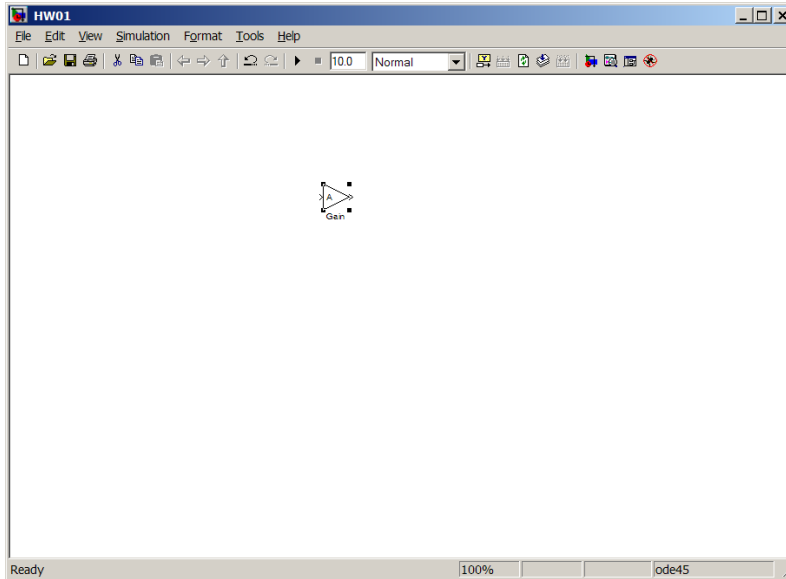
To save space from here on, I won't show the big white space in the model, just the nodes as you should have them connected.

7. Now, we need to integrate ball acceleration to get ball velocity and integrate ball velocity to get ball position. So, drag two [1/s Integrator] nodes into the model. We'll learn soon why an integrator is written as "1/s" in simulink. Now connect the integrators to the gain and your model should look like this:

[Gain A] → [Integrator 1/s] → [Integrator1 1/s]
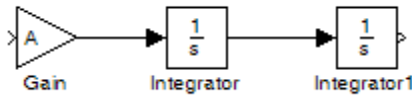
8. Now, we need to model the sensor, which was constructed of the photoresistors arranged as a voltage divider. In reality, the output of the sensor is a voltage between 0 and 5 volts and the voltage is a nonlinear function of ball position, as discussed in class. In our study of controls, we want to work with systems whose nominal value is zero, and we want to start by ignoring complicated nonlinearities. The best way of thinking about this is to say that the output of the sensor in our model is roughly equal to the output of the actual sensor minus about 2.5 volts. And our model will only be valid for small changes in ball position. This process is called "linearization" and we will do it more formally very soon. But for now, let's just say that the sensor generates a voltage that is a simple multiple of how far away from the center the ball is. We don't know what this gain is yet, so call it "B". Because the sensor is in the feedback loop, we want the input to come in on the right and go out on the left. So, after we insert the new gain, we go to (Simulink→Format→Flip Block). Our model look like this:

[Gain A] → [Integrator 1/s] → [Integrator1 1/s]
                                              ↓
   [Gain1 B] ←
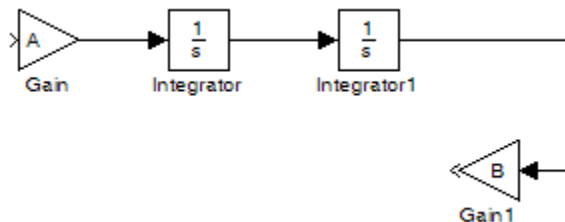
9. At this point, we might want to take a moment to rename our nodes. Do this by simply clicking on the node names and entering more descriptive names. You can also double-click on the connection lines to enter names for the signals that connect the nodes. These are good practices to help you understand your own model and to

make it easier for other engineers to work with the model:
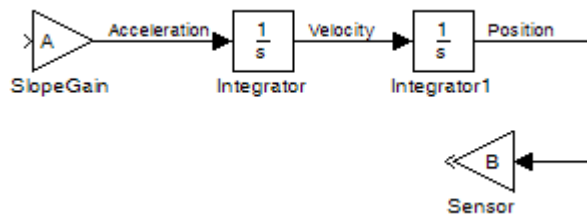
10. Now we need to model the digital-to-analog (D/A) converter. Last year, we used an 8-bit converter, so the output is a number between 0 and 255 (2^8-1). In reality, the output can only have integer values, but we will ignore this fact and use a gain of 51 [counts per volt] to represent the D/A conversion.
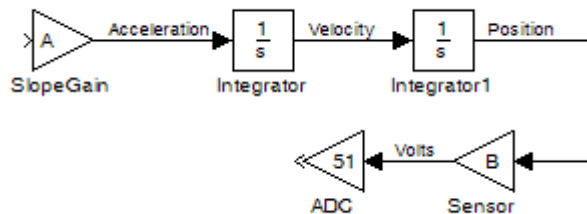

11. However, because the D/A conversion only happens when requested by the microcontroller, there is also what amounts to a pure time delay of half the time step size, as we discussed briefly in the first lecture. We need a new system element to model this. From the [Continuous] group, choose [Transport Delay]. Double click on the "Transport Delay" and in the text field marked "Time Delay" enter "0.5*Ts". We will define the sampling time, Ts, later.
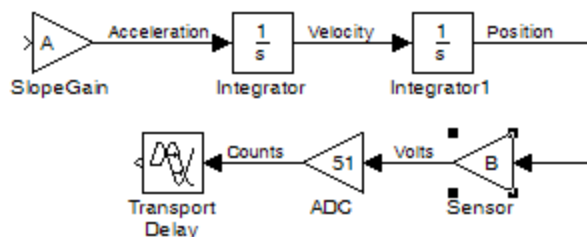

12. The feedback control was implemented with PID (Proportional, Integral, Derivative) control. Roughly speaking, proportional control is like a spring and derivative control is like a damper. Integral control doesn't have a simple counterpart in our mechanical system, but as you saw in class, integral feedback allows the system to achieve zero errors in steady state, which is often very desirable. We'll talk about all of these ideas a lot more before spring break.

You should now be comfortable adding elements to get your model to look like this:
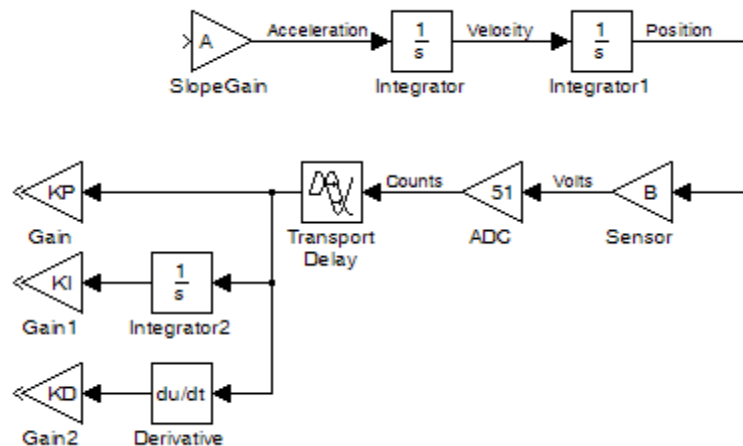

13. Next, we need to add these three signals together. Notice that the [Sum] node has only two inputs by default. But double-clicking on the node reveals instructions for changing the number and/or sign of the inputs. We want to use negative feedback of all three signals, so we enter "---" in the input field, and the model looks like this:
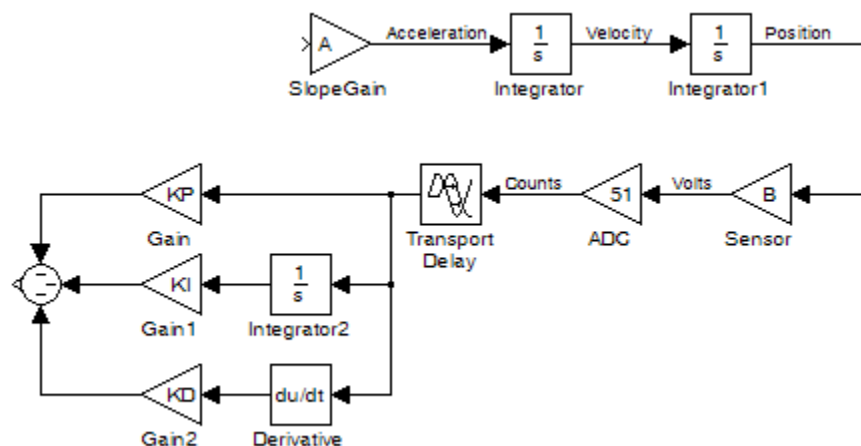
14. We are almost done. The feedback gains are computed in "counts" of servo pulse width. This signal goes to the servo controller, which converts the "counts" into the servo angle. This time, the discrete-time feedback is converted to a continuous-time angle (digital-to-analog), so we now have another half time step delay, plus a gain, call it "C", from counts into degrees. Actually, the servo doesn't instantaneously go to the commanded position. We are making an approximation here that is called "ignoring fast dynamics." In other words, as long as the servo is much faster than our ball motion, we can pretend that it does move instantaneously fast. We often make approximations like this. Sometimes, the effects of these fast dynamics aren't completely ignorable, but we don't want to take the trouble of understanding the details, so we do a very approximate model by adding a little bit more time delay. For now, we'll be happy to ignore the fast dynamics.

Our model now looks like this:
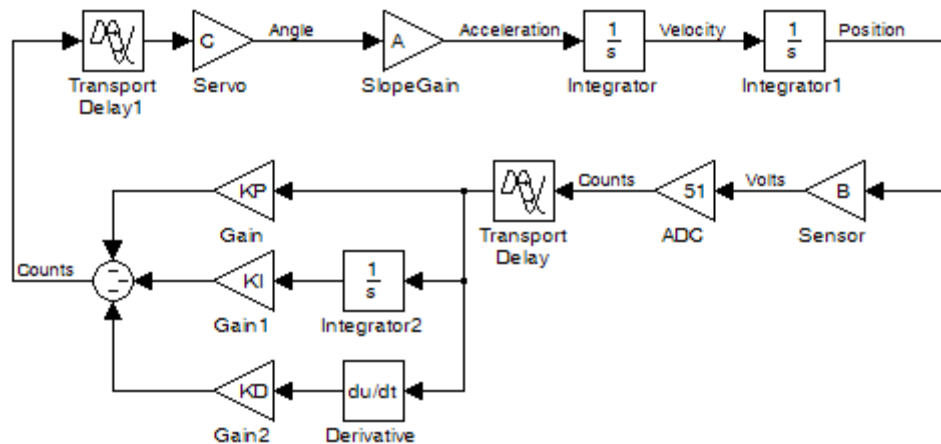
Don't forget to put "0.5*Ts" into the second transport delay.

15. Our model is now complete! We still have to do a few more things so that we can run the model and see the results. The first thing to notice is that because our device is always trying to keep the ball in the middle (it is a "regulator"), we don't have any inputs to get the system going. So, if we run this model, nothing will happen, because the ball and the servo will both just stay at zero. We can fix that by starting the ball off center. Double click on the "Integrator1" node and in the "Initial condition" field, enter "Xo". We will pick a value soon.

16. We also want to be able to view the results of our simulation easily, so let's connect a [Scope] element (in the "sinks" library) to the ball position:
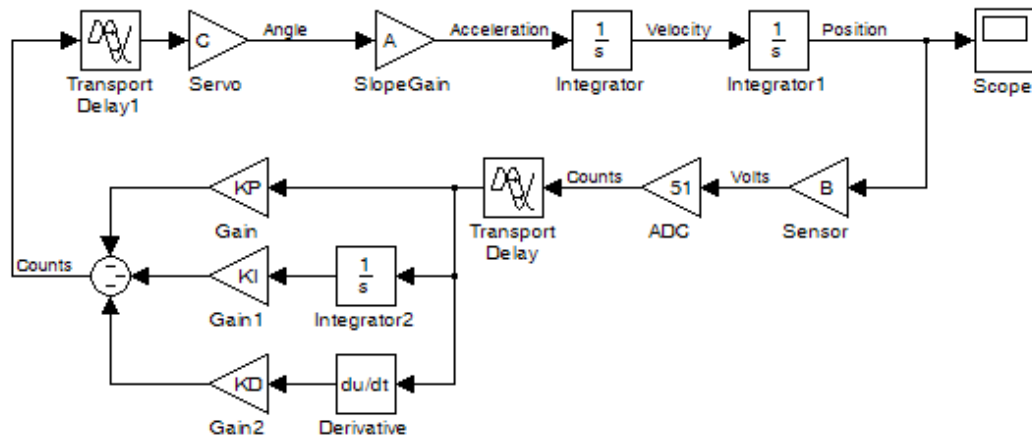
17. We will use the Scope to capture data so we can make pretty graphs later. To do this, double click on the Scope, then click the Parameters Icon [icon]. Go to the Data History tab and make it look like this:
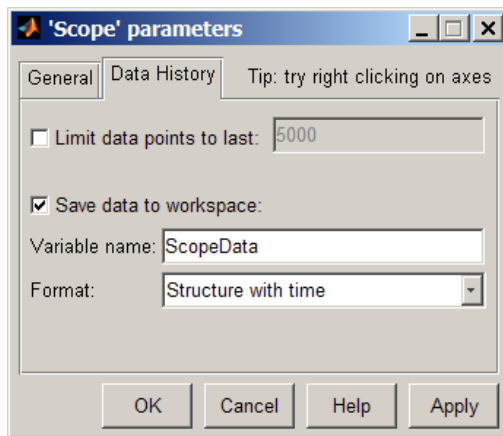
18. Next, we need to ensure that Simulink runs the model at a time step that is much smaller than the transport delays in our system. So, go to (Simulink→Simulation→Configuration Parameters) and under "Solver Options" and "Type", choose "Fixed-Step". In the "fixed-step size" field, enter "0.001". This tells simulink to advance the time in the simulation in one millisecond increments. (Graduate students should take some time to review numerical integration. Good google words are "Explicit Euler", "Runge Kutta" and "Adams Bashforth".)
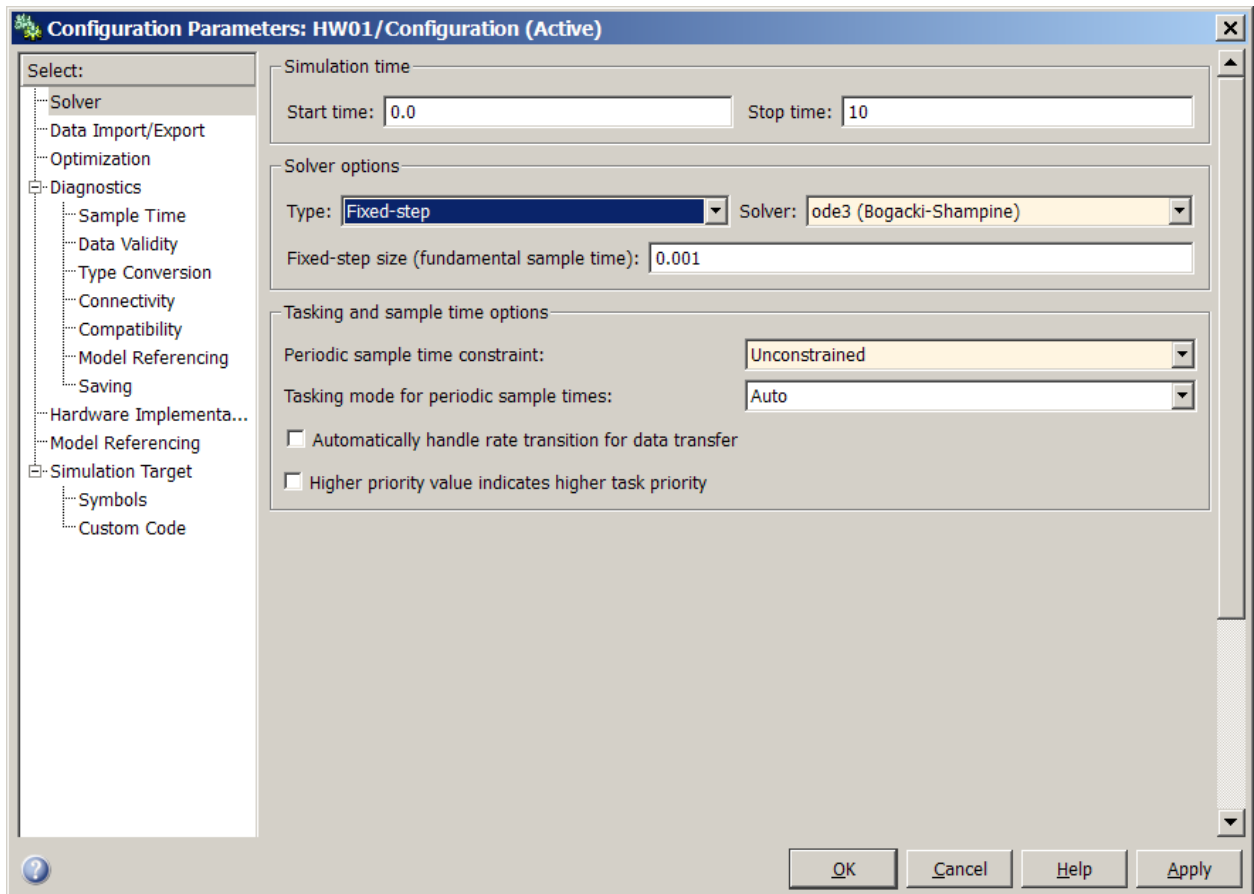
19. We need to pick some numbers. Let's measure distance in inches. It is always important in engineering to use units for which you have some intuition and also for which the values will be reasonable in magnitude. It is difficult to work with numbers like "0.0000065" or "720000000".

    a.   As we discussed in class (see also the notes from 18-Jan-2012), an object rolling down an incline will accelerate less quickly than a block sliding on a frictionless surface, because there is additional kinetic energy stored in the rotational motion. For a hollow sphere, we get only 60% of the acceleration of frictionless sliding. Therefore, the constant "A" should be 60 percent of the acceleration of gravity ("g" ~ 32*12 in/sec^2) per radian. But because we are measuring the servo angle in degrees, we need to divide by 57.3, so A ~ 4.0 [in/sec^2 per degree].

    b.   Our ball is about 1.6 inches in diameter, and the sensors are spaced about 2 inches apart. So, we estimate that B~2 volts/inch.

    c.   The servo scaling is 90 degrees for every 500 counts, so C ~ 0.18 degrees / count.

    d.   The sampling time of the microcontroller was approximately Ts=0.038 [seconds].

    e.   The nominal values of the gains used in the simulation were as follows:
```
KP=(3/16)
KD=(10/16)*Ts
KI=(10/4096)/Ts
```

    f.   Let's start with the ball about Xo=1.0 inch away from center.

20. What is the best way to enter these numbers? In the matlab window, go to (Matlab→File→New→Blank m File). Now just enter the parameters in the file (you can copy-and-paste from this document):
```
A=4.0;
B=2.5;
C=0.18;
Ts=0.038;
KP=3/16;
KD=(10/16)*Ts;
KI=(10/4096)/Ts;
Xo=1.0;
```

The ";" at the end of each line tells Matlab not to echo the values. Save the file as something like "HW01Parameters.m".

21. From the Matlab window, run the file:

```
>> HW01Parameters
```

This puts the variables you defined in the file into the "matlab workspace". Simulink can also read and write the variables in the workspace. Note that the workspace is also where the data saved by the Scope will go.

22. Let's run the model! In the toolbar at the top of the Simulink window, enter a run duration of 10 seconds and click the "play" icon:
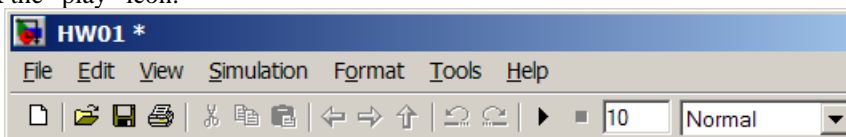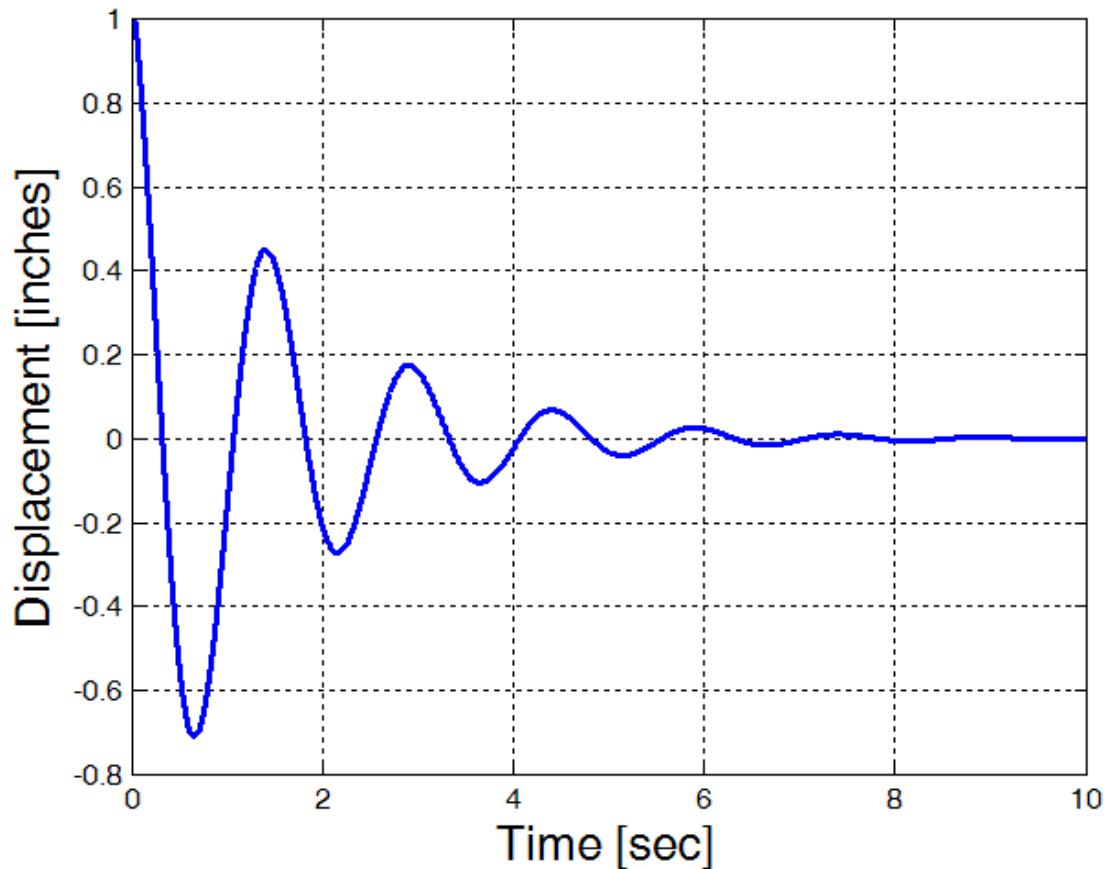

You should see a graph that looks like this:
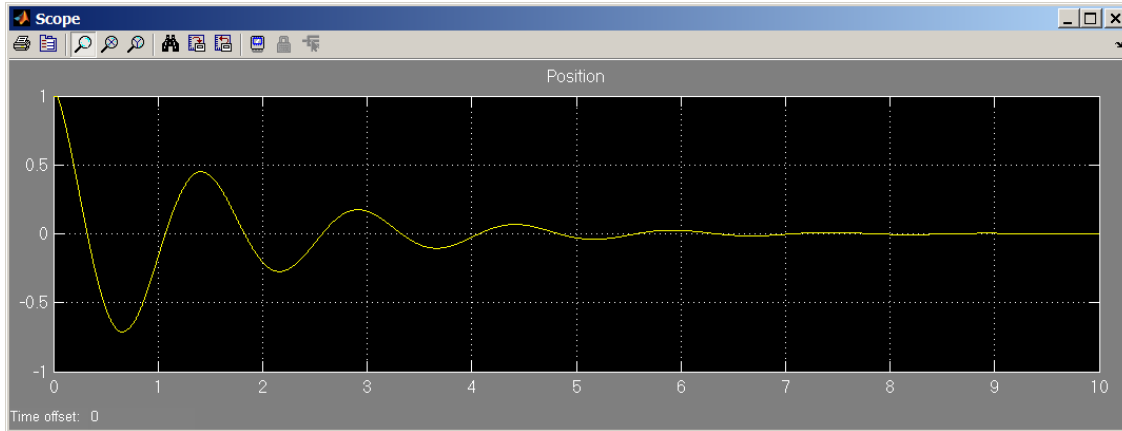
This looks a good bit like the behavior we saw in the video "Case2.wmv"!

23. The graphs that we get from the Scope are not very pretty. We wouldn't want to use a graph like that in a report or even a homework assignment. We can use matlab to make much prettier plots than that. For example, we can get a plot that looks like this:

with the following code (be sure you understand all of these commands):

```
plot(ScopeData.time,ScopeData.signals.values,'LineWidth',2)
grid on;
xlabel('Time [sec]', 'FontSize',16);
ylabel('Displacement [inches]','FontSize',16);
set(gcf,'Color','white')
```

24. You should play with Matlab's plotting capabilities (`>> help plot`) so that you can make pretty graphs.

25. Now let's try to compare our results more directly to the data collected and shown at the beginning of this homework, we need to look at counts. We can add another scope to our model (and remember that our sensor voltage was originally defined relative to center, so we need to add the center offset of 127 counts):
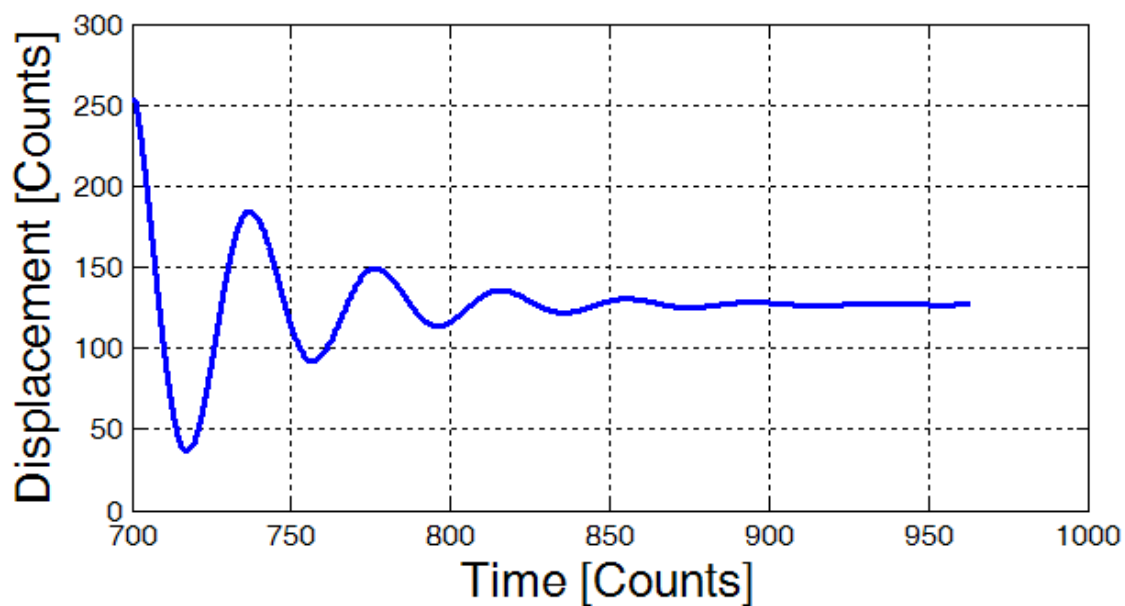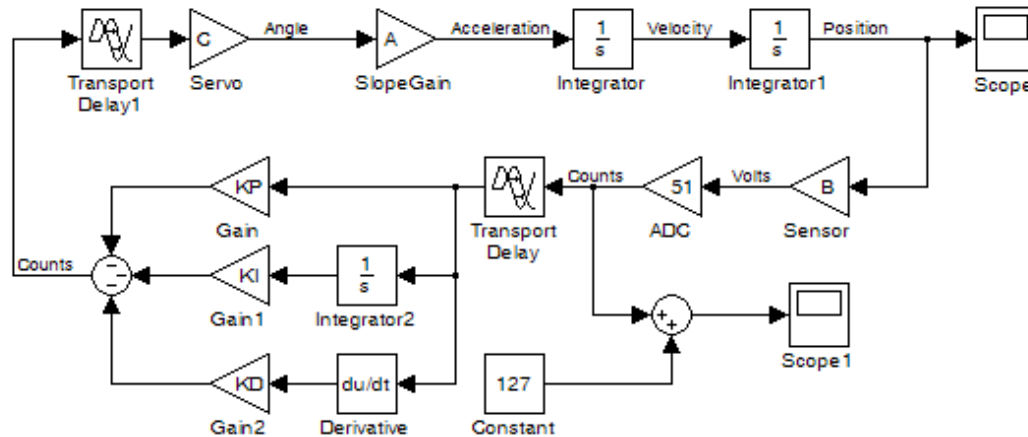
26. In light of the simplicity of our model and the approximations we made to the numerical values of the constants, this prediction is quite good. In particular, notice that the time between peaks is very accurately predicted! The rate of decay of the amplitude of the peaks is also well predicted at first, but then there appears to be some effect that we aren't modeling that causes the actual device to oscillate steadily instead of settling down completely, as our model predicts. If you are interested in exploring what these effects might be, please feel free to play with the model `PingPongPoise.mdl`. The required parameters for the model are in the file `PingPongPoiseParameters.mdl`.

27. Now that we have a reasonably accurate model, we can start to ask questions, like, "How small would the derivative gain have to be for the system to be unstable?" Start by watching the video "Case4.wmv". This shows mildly unstable oscillations caused by reducing the derivative gain to KD=(5.5/16)*Ts, as shown below:

KD=$58 : KP = $30 : KI = $A0
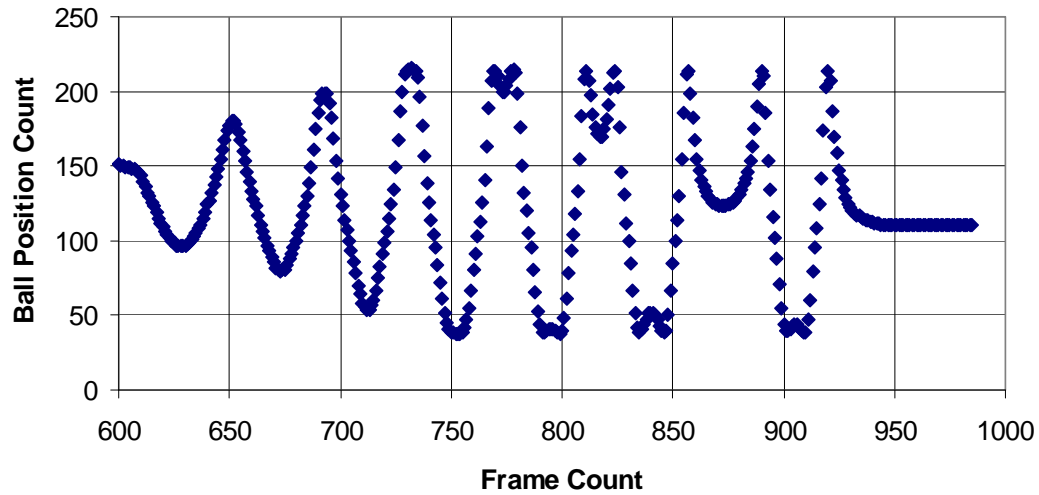
Notice that as the oscillation begins to take the ball outside of the active region of position sensing, the reported position folds back on itself with increasing amplitude.

To match this case, we need to start with the ball closer to the center. Try Xo~0.25 and then experiment with the value of KD to get a graph that looks something like this (or perhaps even more like the graph above):
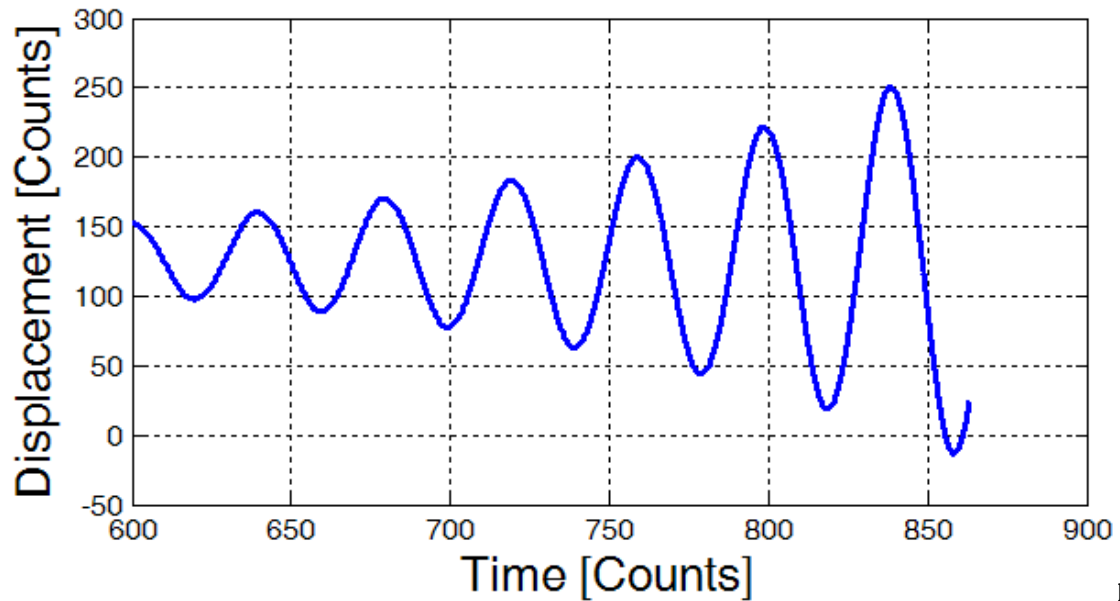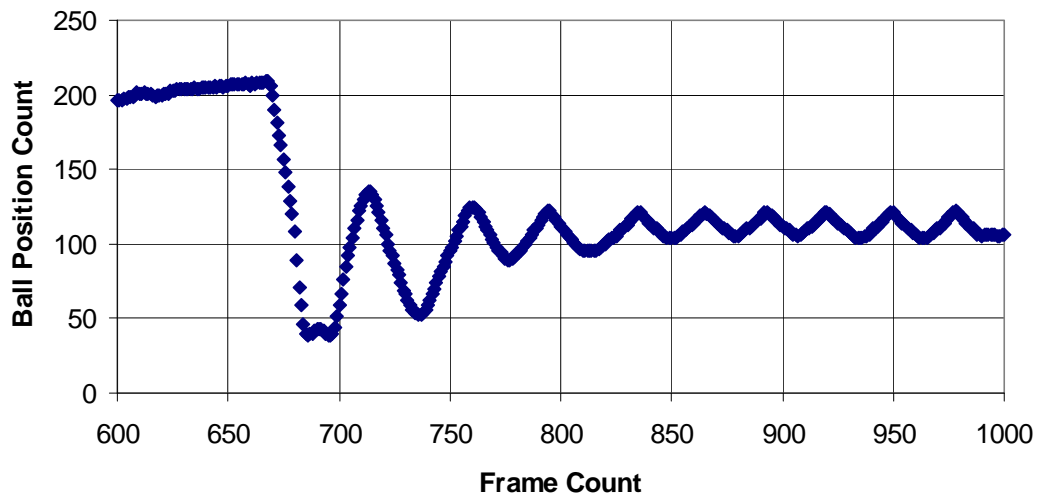

h

Don't worry if your value of KD is smaller than value given above. We ignored some time delay in the servo that causes the actual system to be a little bit less stable than the model.

28. If you wish, you can watch "Case1.wmv", which is a run with the integral feedback turned off. Here are the data:

KD=$A0 : KP = $30 : KI = $00

WHAT AM I SUPPOSED TO SUBMIT FOR THIS ASSIGNMENT?

Please submit the value of KD which resulted in unstable oscillations. You should also write a short description of something you found interesting in your investigations. You should include at least one pretty graph generated in Matlab. Submit at most one page for this problem. Spend your time experimenting and learning Matlab & Simulink, not writing long explanations!

If you are not already familiar with MATLAB, you should spend as much time as you can afford learning it now. There are many excellent online resources (google "matlab primer" or "matlab tutorial") to help you. We will use MATLAB virtually every week in the homework, so a small up-front investment (fixed cost) will save you a lot of time over the semester (recurring cost).