

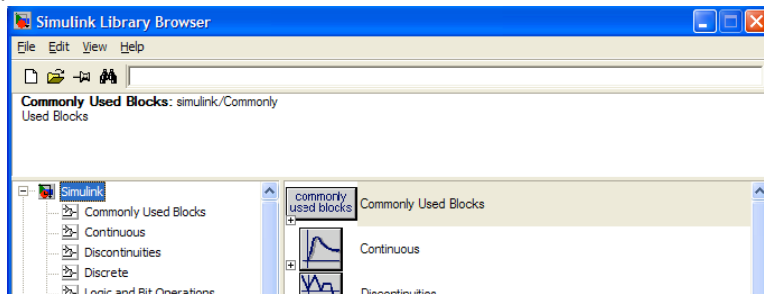
ME 333 Introduction to Mechatronics
Assignment 3
Due Tuesday Jan 24 at the beginning of class

Note: As with all homeworks, each student is required to do this homework on his/her own. You may consult with each other, but every student must complete the assignment alone. Turn in the printouts at the end of the assignment.

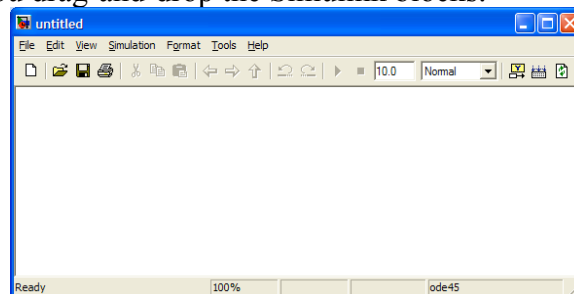
Simulink

Starting Simulink

- 1) Start Simulink by opening Matlab and typing “simulink” in the command line.
- 2) A window called the “Simulink Library Browser” will be displayed. The left frame of the window lists the various installed blocksets and the subcategories of included blocks. The right frame contains the Simulink blocks, which are used to program the simulation.

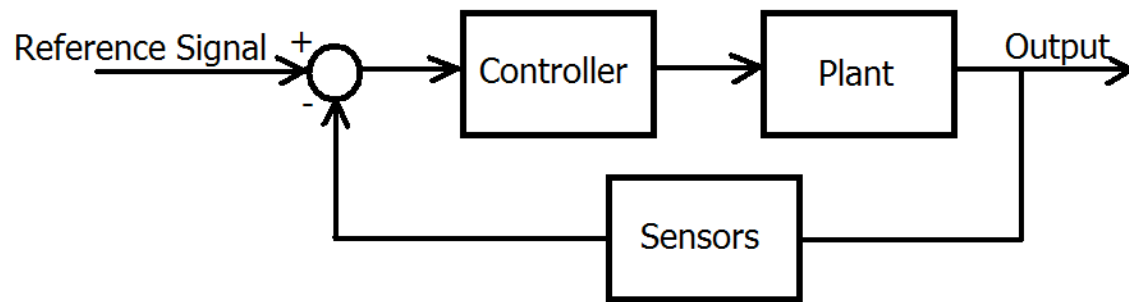


- 3) Click on File _ New _ Model to create a new model. This opens a new model window, where you drag-and-drop the Simulink blocks.



Simulating Control of a DC Motor

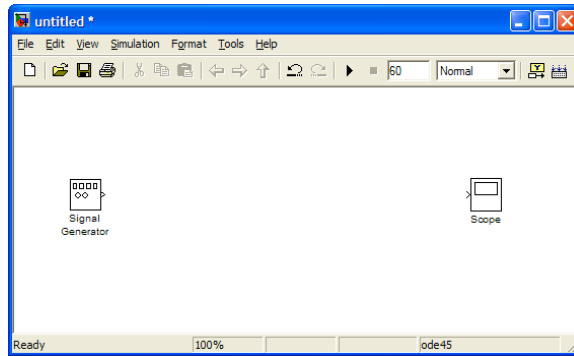
The objective of this exercise is to introduce you to many of the commonly-used blocks in Simulink while at the same time providing a real-world example. For now, you will be simulating a “virtual” DC motor. In lab you will use a similar program to control a real DC motor.



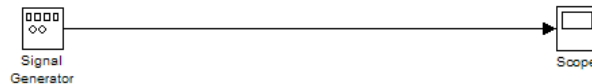
A basic feedback control system is shown above, drawn in the very common block-diagram form. The unique feature of Simulink is that simulation models appear in almost exactly the same format. If you program carefully, it will be very easy for an experienced engineer to quickly understand your model. The *Reference Signal* is what you want the *Output* to be. Your job is to design the controller so that the output signal matches the reference signal.

For our system, our *Plant* will be a DC motor. Because of inertia in the motor, we cannot instantaneously reach a desired velocity or position – the motor has to accelerate and decelerate the mass of the rotor. The motor is assumed to have a sensor attached to its output shaft to measure its position and velocity. A PID controller will be implemented in software to follow a square-wave motor position reference signal. It should be possible to control either the velocity of the motor (for example, if you are controlling the speed of a motor for an application like cruise control) or its position (for example, if you are controlling the position of a robot joint), though the controller for each is different.

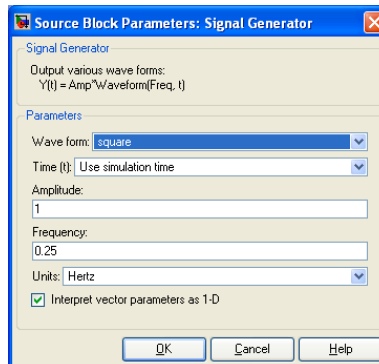
- 1) If you haven't already, open a blank model window.
- 2) Start by creating a reference signal using one of the built-in signal generators and display it to the screen.
 - a) To generate a signal, you will use a "Source" block, located in the Simulink Library Browser window. Find the "Signal Generator" block in the "Simulink\Sources" category and drag it into your blank model window.
 - b) To view or save signals, you will use a "Sink" block. In Simulink, there is a "Scope" block that behaves much like a digital oscilloscope you might find in the lab. Drag the "Scope" block into your model.



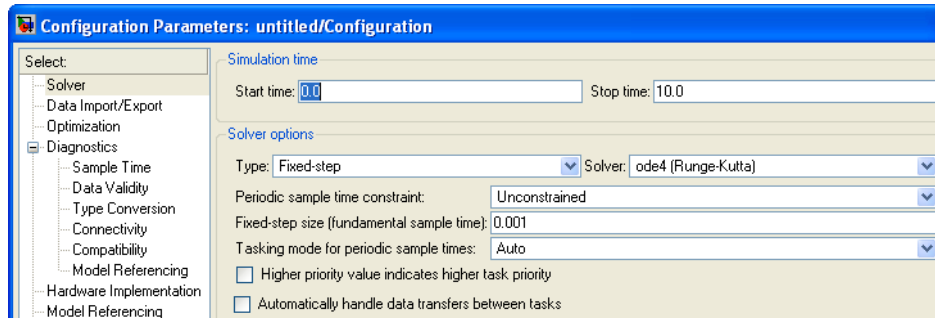
- c) To get the signal from the signal generator block to the scope, you will need to connect the two with a signal line. Notice that the Signal Generator block has a little hat pointing out (signifying a source) and the Scope has a little hat pointing in (signifying a sink). Click and drag the little hat from the Signal Generator to the scope. A dark black line with an arrow at one end should appear, meaning they are connected.



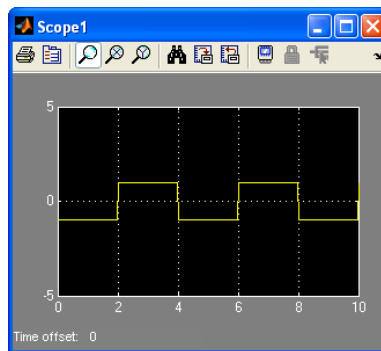
- d) To setup the Signal Generator to output a square wave, double-click on the block to open the “Source Block Parameters: Signal Generator” dialog. Change the “wave form” to “square” and the frequency to “0.25” Hz and click “OK.”



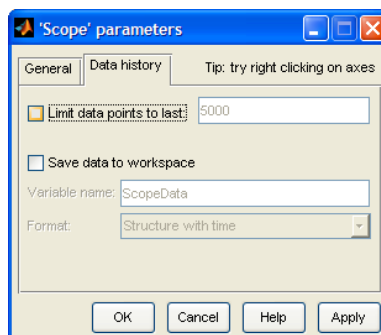
- e) Now we need to adjust the timestep for the simulation. To do this, click on Simulation _ Configuration Parameters. In the Solver menu, change the Solver type to “Fixed-step” and the Solver to “ode4 (Runge-Kutta)”. Then set the Fixed-step size to “0.001”. Then click “OK.”



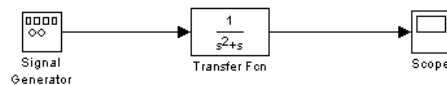
- f) To run the simulation, click on the little “play” button on the menu bar in the model window. The number next to this button is the simulation time in seconds. **NOTE:** in a virtual simulation, this is not the *real* time – the simulation will run as fast as the computer can go. This means setting this number to 10 seconds doesn’t take 10 seconds to run (usually less than one second).
- g) To view the signal in the scope, double-click on the scope block. You should see an output like what is shown below.



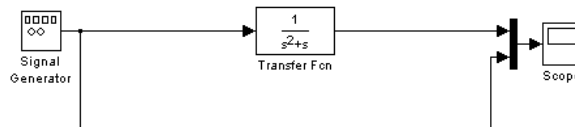
- h) In order to view the entire history of from the scope, you will have to open the scope parameters by clicking on the second icon (a little folder) in the scope window. Click on the “data history” tab and uncheck “limit data points...”. From now on when you run the



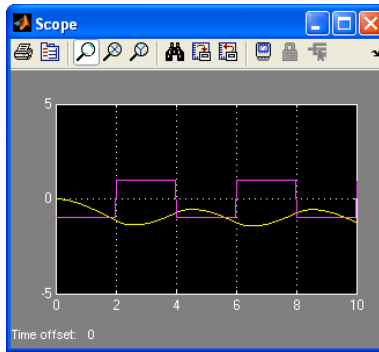
- 3) Now we will create our DC motor model and see what happens when we don't use a controller. This should be exactly the same as if we connected a square wave signal generator (plus current amplifier) up to a real motor and watched what happened. Controlling a motor like this is called "open-loop," because there is no feedback or compensation.
- We will create a very simple model of the DC motor using a transfer-function block. Transfer functions are used in control theory because they provide simple descriptions of linear dynamic systems and make it easy to solve linear differential equations (take ME391 or ECE 360!). Find the "Transfer Function" block from the Simulink\Continuous category and drag it into the model window.
 - Edit the Transfer Function block by double-clicking on it. Leave the numerator the same, [1], and change the denominator to [1 1 0]. Click "OK" to exit the dialog.
 - Now to apply the square wave, drag the Transfer Function block over the line connecting the Signal Generator and the Scope. If you lined it up correctly, the line will break and the Transfer Function block will be added into the series.



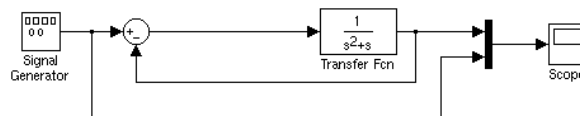
- Run the simulation and observe the output on the scope just as before. Now the scope is showing the position of our virtual motor. Not quite what we wanted, right? That's why we need a controller (or "compensator").
- Now it is also nice to be able to view both the input and output on the same scope graph. This is accomplished through multiplexing, or combining signals. For this you will use the "Mux" block found in the Signal Routing category. Drag this block into the model. You can see that the block has two inputs and one output. The number of inputs can be changed by double-clicking on it, but we only need two. Now, delete the signal line going into the scope and connect the output of the Mux instead. To connect the signal generator signal, you will need to drag the input hat from the Mux to somewhere on the signal connecting the Signal Generator to the Transfer Function. Then connect the output from the Transfer Function block to the other input. It will look something like this:



- Now run the simulation again and view the scope. It will show two different-colored lines.

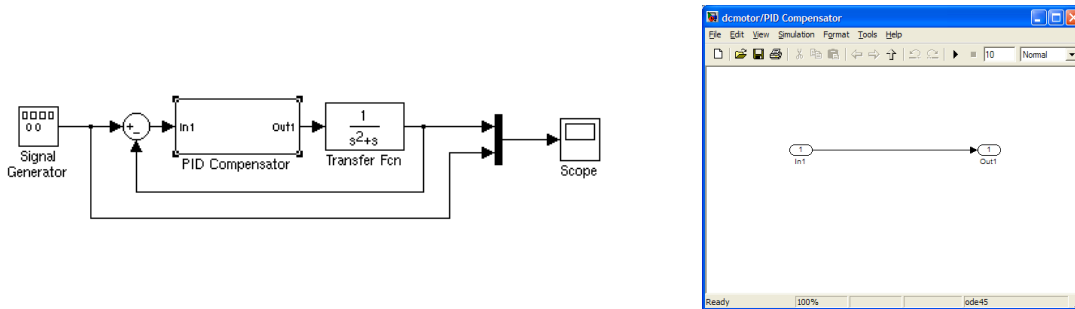


- 4) To try to match these two signals, we will create a control system using a feedback loop and a compensator. We could use a transfer function again like before, but instead we will create our PID compensator using a Subsystem. Subsystems make it easy to organize your model by grouping blocks into a common block.
- First we need to create our feedback loop. This loop goes from the output of our motor (the angle) and is subtracted from the reference signal to get a position error. To do the subtraction, you will need the “Sum” block or the “Add” block. Both are located in the “Math Operations” category. Drag one of the two into the model.
 - Edit the sum/add block so that it subtracts the motor angle from the reference angle. You may have to recreate some of your signal lines. And make sure you keep your reference signal going into the scope. Your model should look something like this:



- Now we will create our compensator subsystem. Find the Subsystem block in the “Ports & Subsystems” category and drag it into the model.
- If you double-click on the Subsystem block, you will open what appears to be a new model window. However, if you look at the name of the window you will see “[modelname]/Subsystem,” where [modelname] is the name of your model (what you saved it as). You do not have to save this model separately; it is included in your base model.
- You can also rename the block in the main model window to be anything you want. Change the name of the block to “PID Compensator” by clicking on the

text “Subsystem” below the model and typing over it. Now see the change reflected in the name of the subsystem model window.

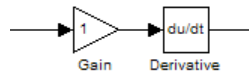


- f) Our PID Compensator subsystem is quite empty right now. As you can see, there are “Ports” defining the inputs and outputs of the subsystem. You can create more of these using the “In” and “Out” blocks also found in the “Ports & Subsystems” category. We don’t need any more. Also, you can rename these ports to make things easier to read. Rename “In1” to “e” and “Out1” to “u” (you’ll see why next).
- g) Now to create our PID control. Such a controller has the format:

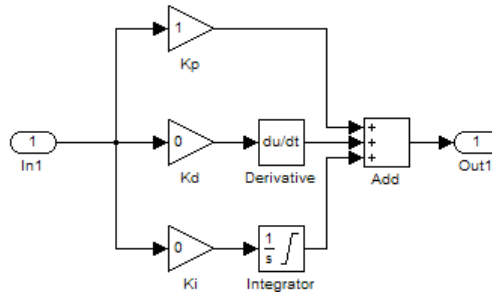
$$u = K_p e + K_d \frac{d}{dt} e + K_i \int e dt$$

where u is the controller output, e is the error you are trying to compensate, and the K ’s are gains. In our PID subsystem, the input is the error and the output is the control, so we need to add some mathematical blocks in between to get our PID controller. Create the proportional term by adding a “Gain” block (found in the “Math Operations” category).

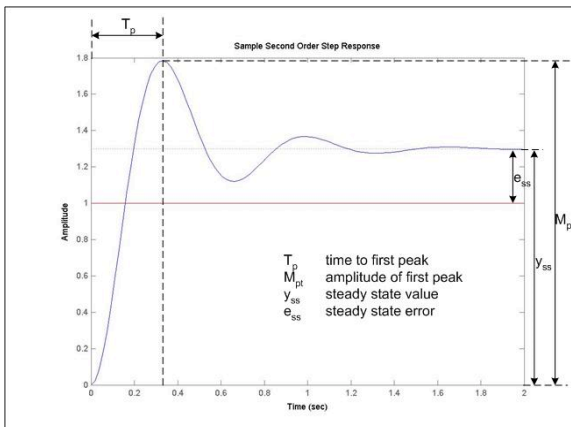
- h) Now to create the other terms, you will need to use the “Derivative” and “Integrator” blocks from the “Continuous” category. Also, these terms have gains associated with them, so you will need to add more “Gain” blocks. Remember, putting blocks in series just means that they are multiplied. For example, your derivative term should look like:



- i) Once you have all three terms built, you will need to add them together using the “Add” block. To do this, you will have to edit the “Add” block so that it will add three numbers. Double-click on the block and edit it so that it has three plus signs (“+++”). Then connect everything together. You should have gotten something like this:

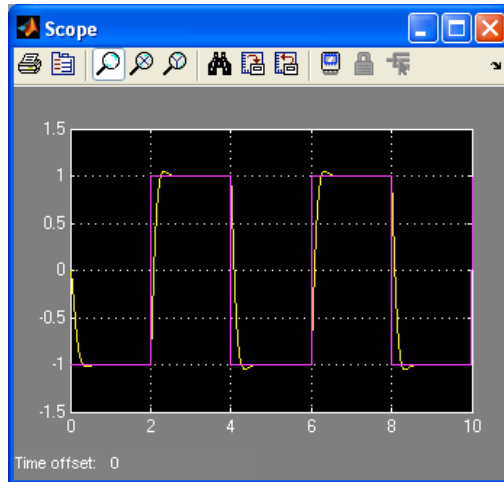


- j) To start with, set the “Kd” and “Ki” gains to zero. Also, you might need to edit the “Integrator” block to add limits. If you let the integral term add forever, it can cause large instabilities. Double-click and check the box, “Limit Output” and change the limits so that the value is [-1,+1] and click “OK.” In your model, you can see the Integrator block has changed appearance to show that you’ve added saturation limits.
- 5) Now run the simulation again. Again it won’t look good, but not as bad as the first time you ran it. Now it’s time to tweak the gains – called “PID tuning.” Below you will see a description of a step response and a table describing how the three gains affect it.



Gain	Rise Time	Overshoot	Settling Time	Steady-State Error
Kp	Decreases	Increases	-	Decreases
Kd	Increases	Decreases	Decreases	-
Ki	Decreases	Increases	Increases	Zero

- a) Adjust the gains until you can get a nice step response – short rise time, low overshoot, short settling time, low steady-state error. (Note: you may leave the integral gain Ki equal to zero, or at least it will likely be small.) A good controller might give performance that looks something like this:



To turn in: Print a screen shot of your model, your PID compensator subsystem, and the scope showing the performance of your controller. Along with these plots, give the gains K_p , K_i , and K_d that you used to achieve this performance.

- 6) Let's see what your actual control signal u is. Change the mux before your scope block to accept three inputs, and make u (the output of the PID compensator) this third input. Run the simulation again. You might find that u , which represents a control current or voltage, becomes very large. In practice, all real controllers have limited output ranges. In other words, the output saturates. Modify your PID compensator subsystem by including a saturation block (find it under "Discontinuities") between the "Add" block and the u output port. Modify the upper and lower limits to be +10 and -10. Now run your simulation again, and see if the behavior changes due to the practical output limits. If necessary, change your three controller gains until you get the best possible performance. You may not be able to get performance quite as good as before due to the saturation limits.

To turn in: Print a screen shot of your PID compensator subsystem with the saturation block and the scope showing the new performance of your controller. Give the gains K_p , K_i , and K_d that you used to achieve this performance.

In Lab 3 you will be constructing a control system very similar to this one to control a real DC motor. Successful completion of this homework will increase your chances to successfully complete Lab 3!