LAB 3
xPC TARGET, THE PC/104 STACK, AND CLOSED-LOOP CONTROL OF A MOTOR

Objectives	To learn to use Simulink to generate real-time control code for a target computer.
Preparation	The Simulink homework and reading the webpage describing the PC/104 hardware.
Tools	Prototyping breadboard, PC/104 stack as the target real-time computer, and host PC running Matlab with Simulink, Real-Time Workshop, and xPC Target.

Overview

In this lab, you will be learning how to use some very powerful add-ons to Matlab – Simulink, Real-Time Workshop, and xPC Target – to perform real-time computer control of a motor. Simulink is a visual programming environment for designing and simulating systems. Most programming in Simulink is accomplished through simple drag-and-drop procedures. Real-Time Workshop interfaces with a C compiler to compile Simulink programs into C code. xPC Target allows you to run this code in real-time on a "target" computer. In short, you create your control program on a "host" computer using Simulink, then you compile it and download it to the "target" computer which is running the xPC Target real-time operating system. This operating system executes your program in "real time," without all the overhead of operating systems like Windows which prevent you from running a program with guaranteed cycle times. For us, the host computer is a regular desktop PC, and the target computer is a PC/104 computer stack, with a CPU/motherboard card, a Sensoray 526 multifunction card, and a BoB (breakout board) for easily accessing all the inputs and outputs of the 526. These include 8 analog inputs, 4 analog outputs, 4 encoder inputs, and a number of digital inputs and outputs.

Before beginning this lab, you should follow the webpage instructions to create your first simple model in Simulink for download to your PC/104 stack. Once you have completed this, you are ready for the rest of the lab. This lab builds on your homework exercise of controlling a model DC motor.

Controlling a Real DC Motor

1. Build a linear amplifier as in Lab 2, to control the motor. This consists of a TIP31 and TIP32 transistor with bases and emitters attached, collector of the TIP31 to +12V, and collector of the TIP32 to -12V; 1/4 of an LM348 op amp with output attached to the bases of the transistors, input to the inverting input from the emitter of the transistors, and input to the noninverting input attached to analog output 1 of the BoB of your PC/104 stack. To attach to the analog output of the PC/104, you should use two long wires, twist

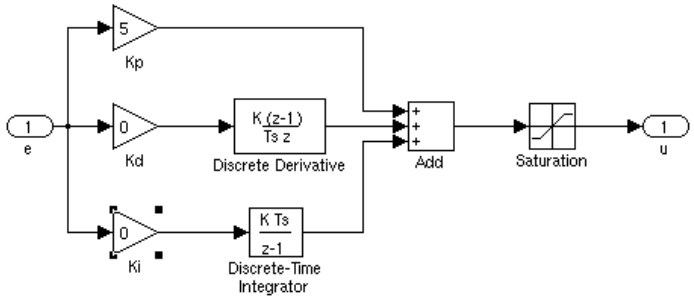
them together so you have a cable called a "twisted pair," attach one to analog ground of the BoB (either of the unmarked terminals at either end of the analog out terminal strip) and the other to analog output 1. Then plug the ground wire into ground of your protoboard and the analog output 1 wire into the noninverting input of your current amplifier op amp. Ideally we would test that your linear amplifier is working, but we'll assume you've hooked it up right! If it's not working later, you know you may have to debug it.

You can leave your PC/104 stack turned off for now. Attach the encoder cable to the ENC1 input on your BoB. Be careful! This cable can connect in either of two ways. Only one is right. Verify with the TA that yours is connected correctly.

2. Now you will essentially need to reconstruct the motor controller model you created in your homework. Create a new model in Simulink, and perform the following steps.

- Save your model as Lab3.mdl into the folder MATLAB71/work/TeamXY/Lab3, where XY is your team number. Make sure that the Current Directory in your Matlab window is this folder.
- Set the text box at the top right of your model window to "External" instead of "Normal," indicating that the model will run on an external computer.
- Open Simulation/Configuration Parameters and set the Solver Type to "Fixed-step," the Solver to "discrete (no continuous states)," and the Fixed-step size to "dt". (In the Matlab window, if you haven't already, set "**dt = 0.001**" indicating that the sample time will be 1 ms.) In Data Import/Export, check the "Limit data points to last" box and set it to 10000. This will put a limit on how much data our target PC/104 ships back to Matlab on the host PC at the end of the run. Specifically, with our step size of 1 ms, our PC/104 will send back no more than 10 seconds worth of data. Now, under Real-Time Workshop, set the System target file to be xpctarget.tlc.
- Drag in a Signal Generator and set the waveform to square, the frequency to 0.25 Hz, and the amplitude to 1. This will eventually mean that we want our motor angle to follow a square wave of amplitude 1 radian with a period of 4 seconds.
- Drag in a Sum block and change the "List of signs" to "|+-". Then attach the output of the Signal Generator to the + input of the summer.
- Drag in a Subsystem and rename it "PID Compensator." Connect its input to the output of the summer. Now go into the PID subsystem and rename its input e and output u. Drag into the PID compensator subsystem three gain blocks, and name them Kp, Kd, and Ki. Each of these takes e as input. Make the gains of the Kd and Ki blocks zero, and the Kp gain 5. Now drag in a Discrete Derivative block (in the "Discrete" category) and make its input the output of the Kd gain block, and drag in a Discrete-Time Integrator block and make its input the output of the Ki gain block. Make sure the Sample time for your Discrete-Time Integrator is dt. Drag in an Add block and change its list of signs to be "+++" so it has three inputs, and make these three inputs the outputs of the Kp Gain, the Discrete Derivative, and the Discrete-Time Integrator blocks. The output of the Add block should then go into a Saturation block (from "Discontinuities"). Set the lower and upper limits to 10 and -10, respectively. This will limit the output voltage we try

to send from the Sensoray card to be in the range +/-10V. Now connect the output of the Saturation to the output u of the PID subsystem. At the end of this step, your PID subsystem should look like this:



Note that the output voltage from your BoB is actually only in the range +/- 9.09V, not +/- 10V, as the BoB has protection circuits that multiply the Sensoray analog outputs by a gain of -0.909.

- Into your main model, drag in another Subsystem and give it the name "Motor." Connect its input to the output of the PID Compensator. In our homework, this subsystem (we used a transfer function in the homework) simulated the motor and provided the simulated angle of the motor as output. Now we will use it to send the control signal to the the Sensoray analog output and to read in the motor angle from the encoder using the Sensoray encoder input.

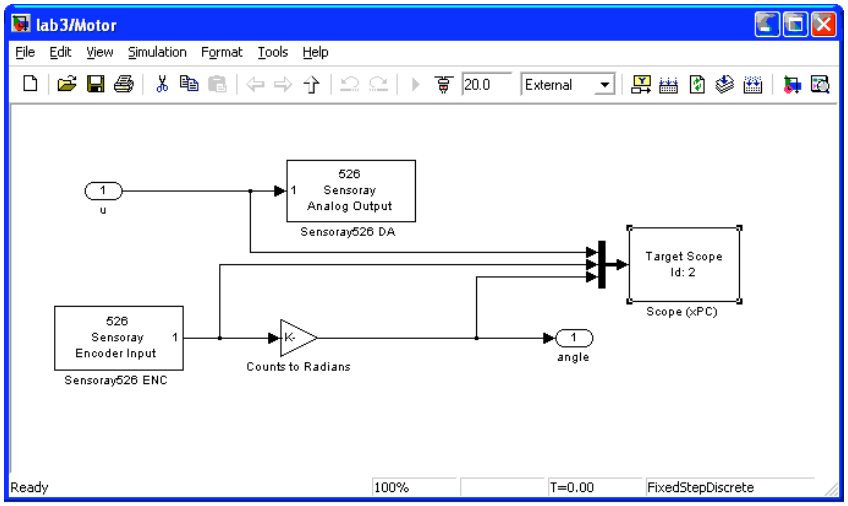
  Open up the subsystem and rename the input "u" and the output "angle." From the xPC Target category, click on D/A, then double-click on Sensoray, then drag a Sensoray526 DA block into your Motor subsystem. Open the block and make sure "Channels" is set to "[1]", meaning that you will output to channel 1. Set the reset vector to "[1]". This means that when your model finishes running, the output will be set to whatever value is in your "Initial value vector" (which should be [0], meaning 0 volts). Set "Sample time" to dt.

  From xPC Target/Misc., drag in a "Scope (xPC)". This is a special kind of scope that works in real-time on the target computer, not on the host like the regular scope. Change the the Scope mode to be Numerical, and choose the Numerical format to be **'Control: %8.4f ,Encoder: %8f ,Angle (rad): %8.4f'**. Note the single quotes must be included. This phrase is a formatting string that tells the target how to format the data it prints to the screen. This scope will eventually list the values of the control voltage, the number of encoder counts, and the angle of the motor in radians in real-time on the target's monitor. These numbers will be updated every **Number of samples** timesteps, which you can set to 250.

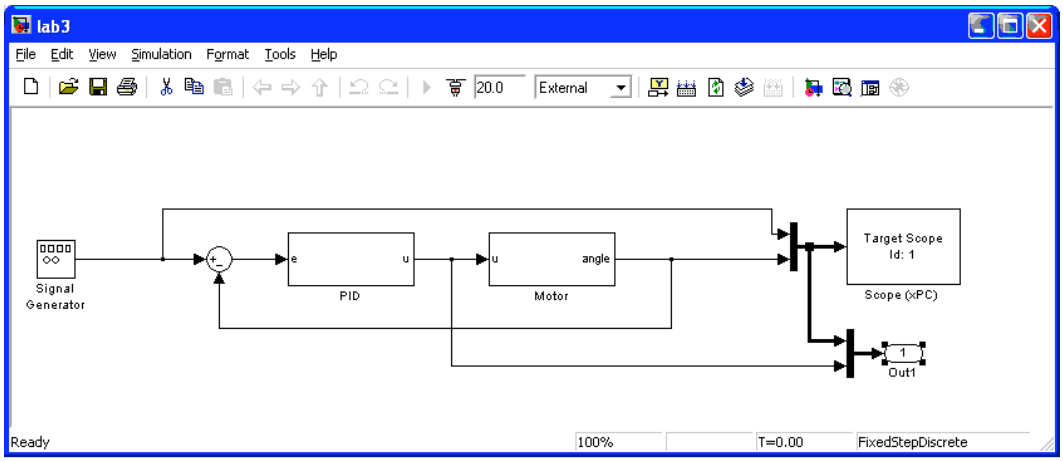
  Next, drag into your Motor subsystem a Sensoray526 Enc block, from xPC Target/Incremental Encoder. Open up the block and change the Count speed to 4x (meaning that 4 pulses will be counted for every "line" on the encoder disk). Change the Sample time to dt. Make sure the Channel is 1.

Now drag a gain block into your Motor subsystem and change its name to "Counts to Radians." This gain block changes the encoder counts to motor angle in radians. Our encoder wheel has 100 lines on it. Since we are using it in 4x mode (see the previous paragraph), we will count 400 counts every time the motor shaft spins one revolution. What we really want to control, however, is the angle of the output shaft of the gearbox on our motor, and we have a 6:1 gearbox on the motor. Every time the motor shaft spins six times, the output shaft of the gearbox spins once. This means we actually have 2400 counts for every time the output shaft spins once. Therefore, we should divide the number of encoder counts by 2400 to get the number of revolutions of the output shaft, and then multiply that by 2π to get the number of radians rotated. So, change the gain of your gain block to be "2*pi/2400."

Now, drag in a Mux block and give it 3 inputs. Finally, arrange your Motor subsystem so it looks something like below. You are done with your Motor subsystem.



- Into your main model, drag another xPC Target Scope. Make sure the Scope mode is "Graphical redraw," and set the Number of samples to 4000. This will allow you to see 4 seconds worth of data at a time on an oscilloscope-like screen. The target will collect 4 seconds worth of data, then dump it to the screen at the same time, while the target is running.
- Finally, complete your main model with two Muxes, a Sink "Out1" which will ship data back to Matlab, and by completing your connections so the final model looks something like below. Note that we have our feedback control system as we had in our previous homework, except that the Motor subsystem has blocks to connect to our multifunction card, and we are sending data to xPC Target scopes and back to Matlab after the run.



3. Now we will compile and run the model on your PC/104 stack. Turn on your stack, which should still be connected by ethernet to the router at your desk, as well as to the VGA monitor, your motor, and the linear amplifier you built (make sure you have power to your protoboard!) Your linear amplifier should be connected to your motor terminals by alligator clips.

To "build" your model, that is, to compile the C code and send it to your target computer, type Ctrl-B. (Alternatively, you can find this command under Tools/Real-Time Workshop.) After it has built successfully (you will know that it has by looking at your Matlab window, or by seeing the target computer's monitor display initialized), click on the little plug icon next to the execution time block at the top of your model window. This connects your host computer to your target computer, assuming you correctly have "External" instead of "Normal" in the nearby text box. (If you click on the plug icon again, you disconnect.) Finally, click on the play button next to the plug icon. Your model will run on the target PC/104! It will continue to run for the number of seconds specified in the text box.

If your motor moves, but does not simply move back and forth between two positions, then switch the clips on your motor terminals. If you still don't get this behavior, something went wrong, and it's time for you to do some debugging!

You can make some changes to your program even while the program is running. For example, if you set the run time to something long, like 100 seconds or more, you could change the gains in your PID compensator. As soon as one is changed, the target PC/104 will implement it. You don't need to rebuild the model. Other changes, however, require you to rebuild. For example, if you change the structure of the model (add or subtract blocks) or change the run-time, you will have to disconnect, rebuild, and reconnect.

4. Tasks for your lab writeup (only **one** required per group):
a. Demonstrate your working controller to the TA, and save your working model.
b. Tune your PID gains to get the best possible controller. A good controller has a step response which quickly achieves the desired value with little or no overshoot and little or no steady state error. Once you think you have achieved good performance, in Matlab you should plot the data. After running the model on your target, in your Matlab window type

TIME = tg.TimeLog;
DATA = tg.OutputLog;

This gives you a vector of timestamps and an n x 3 matrix of data values consisting of the reference angle, the actual angle, and the control voltage at each of the time stamps. This is the data that was sent out to the "Out1" port in your model, which the PC/104 stack sends back to the host. Now you can plot this data using

plot(TIME,DATA)

in Matlab. You might see that the output of your controller is always chattering back and forth between the saturation limits. If this is the case, then modify your plot to something like

plot(TIME,DATA(:,1:2))

to only plot the commanded angle and the actual angle. Print out this plot and write on it the Kp, Ki, and Kd gains you used. By looking closely at the plot using the zoom function, give the typical angle error just before the reference signal switches (this is approximately the steady-state error), the maximum overshoot in radians, and the amount of time it takes after the reference signal switches for the actual angle to get within 0.05 radians of the desired angle and stay there.

Even when you are using small gains, you may notice that your motor "hums" when you set Kd to be nonzero. You may also see in your plots that your control signal is jittery. Can you explain why? Sometimes this "humming" can be a bad thing. Can you suggest a way to suppress it while leaving Kd nonzero? Keep in mind that the "discrete derivative" block is calculating the motor velocity in a very simple way: essentially, it takes the current motor angle, subtracts the motor angle from the previous timestep, then divides the difference by the timestep.
c. Using the same gains you used in part b, change your reference signal to a sine wave of the same amplitude and frequency and plot the same information as in part b.
d. Try to force the motor to follow the sine wave input using only integral control (Ki), with Kp = Kd = 0. Can you do it?

You should turn in your plots for parts **b** and **c** and answer all questions in parts **b** and **d**.