Exercise 2: Dynamic System Simulation*

EENG350: Systems Exploration, Engineering, and Design Laboratory

Tyrone Vincent and Hisham Sager

Department of Electrical Engineering
Colorado School of Mines

Spring 2022

1 Introduction

Exercise Objectives:

- Create a Simulink block diagram model of a dynamic system and run a simulation
- Use the Matlab interface to run a simulation and display plots or animations of the results
- Create a simulation of a DC motor
- Use simulation to demonstrate the effect of PWM modulation, quantization and sampling
- Use the publish function of Matlab to produce a report
- Tune a PI controller to control the angular position of a DC motor

2 Simulink Introduction

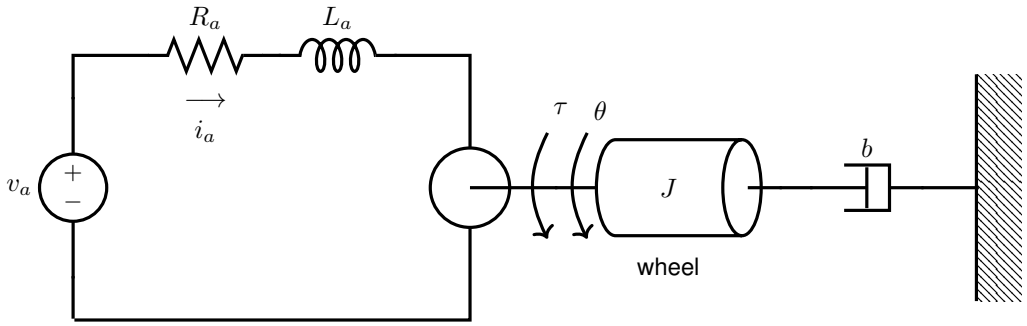
Simulink is a block-diagram oriented system simulation tool. Block diagrams are simply graphical representations of equations, and help us easily see the relationship between variables in a way that is not as clear when the equations are written out. There are several resources that you can use to become familiar with Simulink.

- For an introduction into the basics of Simulink (creating a model and simulating it) follow along to this tutorial: http://ctms.engin.umich.edu/CTMS/index.php?aux=Basics_Simulink. You will want have Matlab available to that you can re-create the model on your own computer.
- To see how to create Simulink block diagram that represents a set of differential equations, use this tutorial: http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=SimulinkModeling
- For more details and some advanced concepts, you can view Simulink video tutorials created by the Mathworks: https://www.mathworks.com/learn/tutorials/simulink-onramp.html.

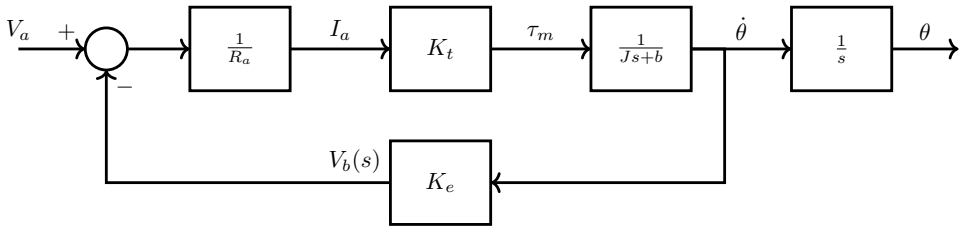
Note that you can get access to Matlab and Simulink at most any computer lab on campus, and in addition, you can get Matlab and Simulink on your personal computer. It is expected that you are already familiar with MATLAB and creating MATLAB functions. If not, you may also want to look at the Mathworks tutorials for MATLAB https://www.mathworks.com/support/learn-with-matlab-tutorials.html, especially the Matlab Onramp.

3 Problem 1: Simulation of a Motor

Review EENG307 Lecture 11, Motors and Hydraulic Actuators. A DC motor driving a wheel with bearing friction has the following ideal component representation.



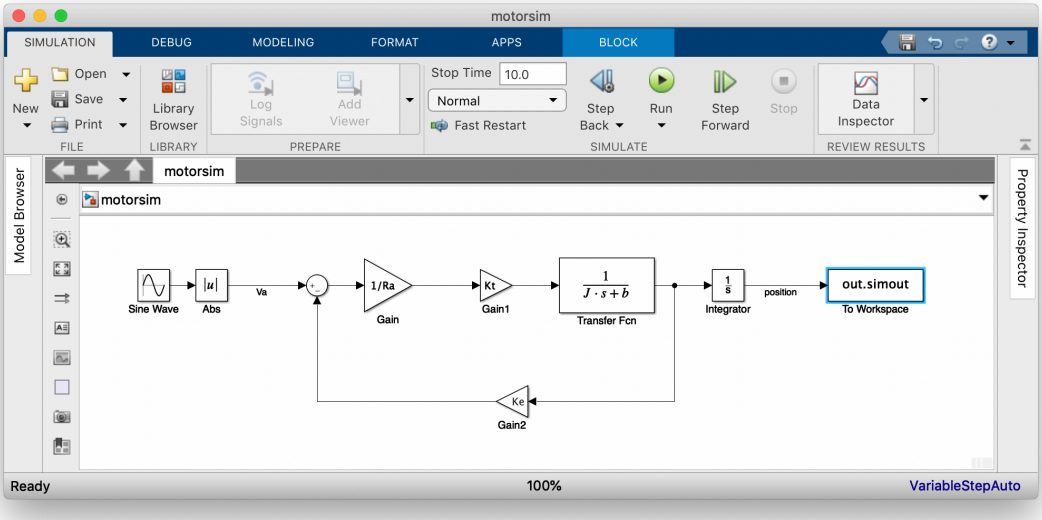
For this load, the motor block diagram is



Note that we have made the output of the load block $\left(\frac{1}{Js+b}\right)$ to be $\dot{\theta}$, with units of radians per second. This means that the back emf term is just K_e, rather than sK_e, and we integrate $\dot{\theta}$ with the integrator transfer function $\frac{1}{s}$ to get θ.

3.1 Simulink Motor Model

You can create a simulink block diagram for the motor model which looks pretty much like the one drawn above. We will add a rectified sinewave of amplitude 1 and frequency 1 rad/s as the voltage input (V_a). Create a script that will enter the values of the variables K_t, K_e, R_a, J and b, run the simulation, and plot the results. Use $R_a = 1$, $K_t = .5$, $Ke = .5$, $J = .05$, $b = .5$.

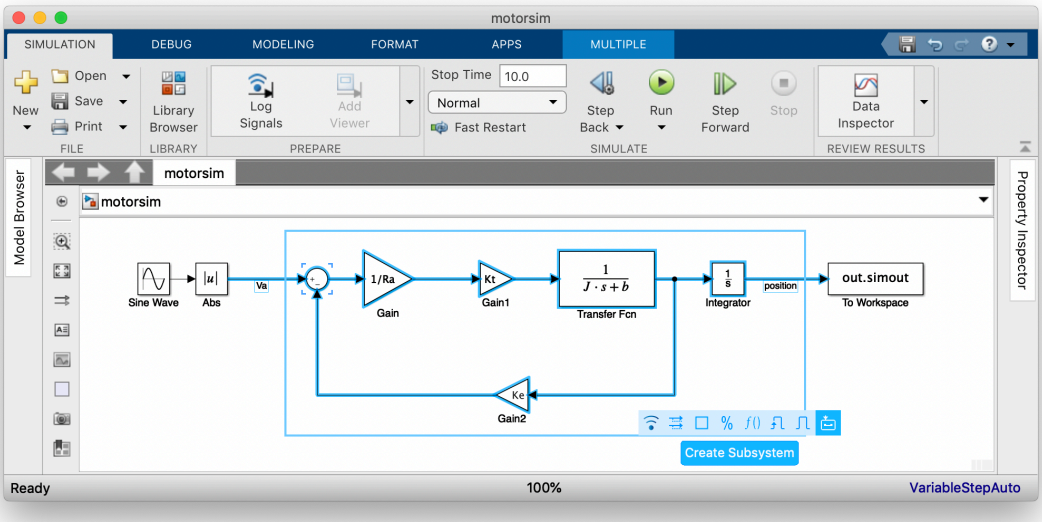


Note that to workspace blocks are used to save the results in the variable `simout` (for position). Note also that two of the signals have been named - Va and position (look for text below the lines/arrows). The variable Va is the voltage V_a and the variable position is the angular position θ. You can name a signal by clicking on the line/arrow associated with that signal and typing the name. The following code sets up the parameters, runs the simulation (assuming you have saved your Simulink block diagram as 'motorsim.slx', and plots the results.

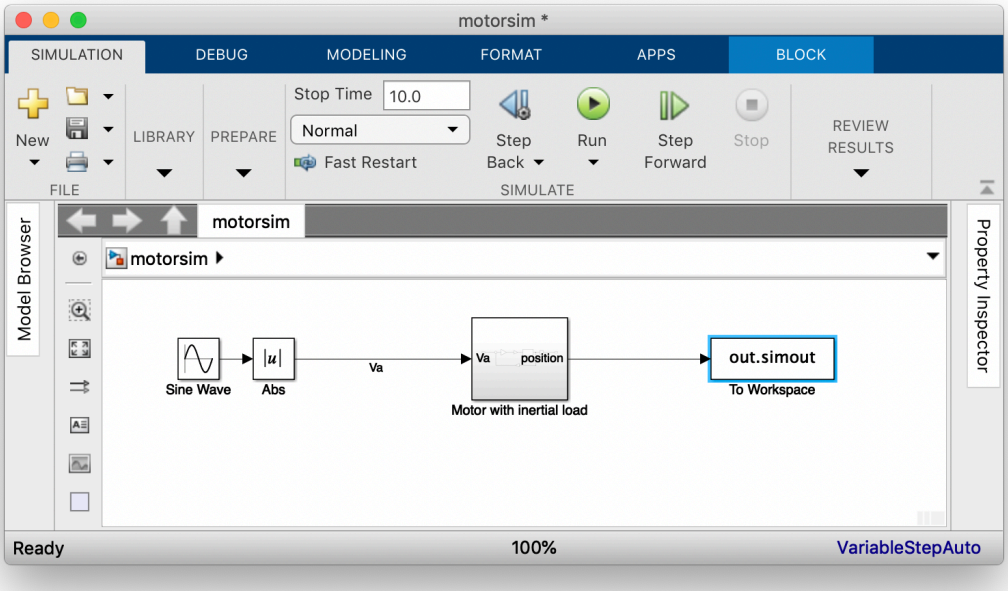
```
%% Runmotorsim.m
% This script runs a simulation of a motor with bearing friction and plots the
% results
%
% required file: motorsim.slx
%
%% Define motor parameters
Ra=1; % armaature resistance [Ohms]
Kt=.5; % motor torque constant [Nm/A]
Ke=.5; % back emf constant [Vs/rad]
J=.05; % Load inertia [Nm^2]
b=.5; % damping [Nm/s]

%% Run a Simulation
%
% This simulation applies a rectifed sinusoidal voltage to a DC motor model,
% with the output as the angular position in radians
%

%
% open the block diagram so it appears in the documentation when published.
% Make sure the block diagram is closed before running the publish function
%
open_system('motorsim')

%
% run the simulation
%
out=sim('motorsim');

%% A Plot of the results
%
% We see that the motor rotates in the positive direction, with some oscillations
% due to the varying input
%
figure
plot(out.simout)
```

Note that simulation results are saved in the structured variable `out`, and `out.simout` is accessed using the dot notation. The default for the to workspace variables are structures called Timeseries, that include both the time and signal magnitude. (Type `out.simout`) at the command line to see what is included in the structure.

In order to organize complicated block diagrams, Simulink allows you to group blocks into a subsystem. Select the blocks associated with the motor, and hover the cursor over the blue square that appears at the bottom. The first icon will create a subsystem.



The inputs and outputs of the subsystem block should be named Va and position, as they inherit the names of the signals. If not, double check on the block to name them. You can also click on the text below the block to name the block motor with interial load. Your block diagram should look like the following.

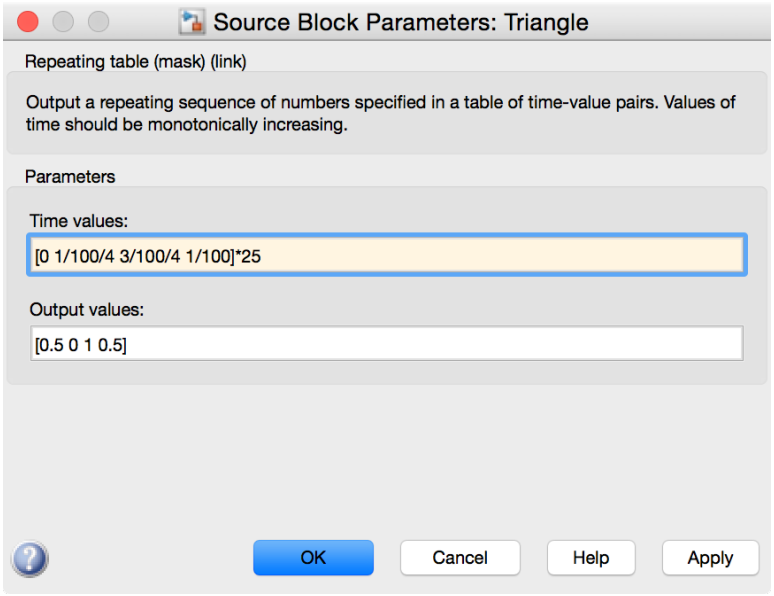


3.2 Pulse Width Modulation

Recall that the torque supplied by the motor is $K_t i_a$ where i_a is the armature current. In order to drive the wheel, the motor must be supplied with a voltage V_a that also has sufficient current to achieve the desired torque. Often, we have available a power supply/battery with constant DC output, but varying this voltage to intermediate values between 0 and max voltage requires additional circuitry and can be inefficient. A way around this problem is to use power transistors to switch the power supply on and off in a way that the average voltage is equal to the desired V_a.

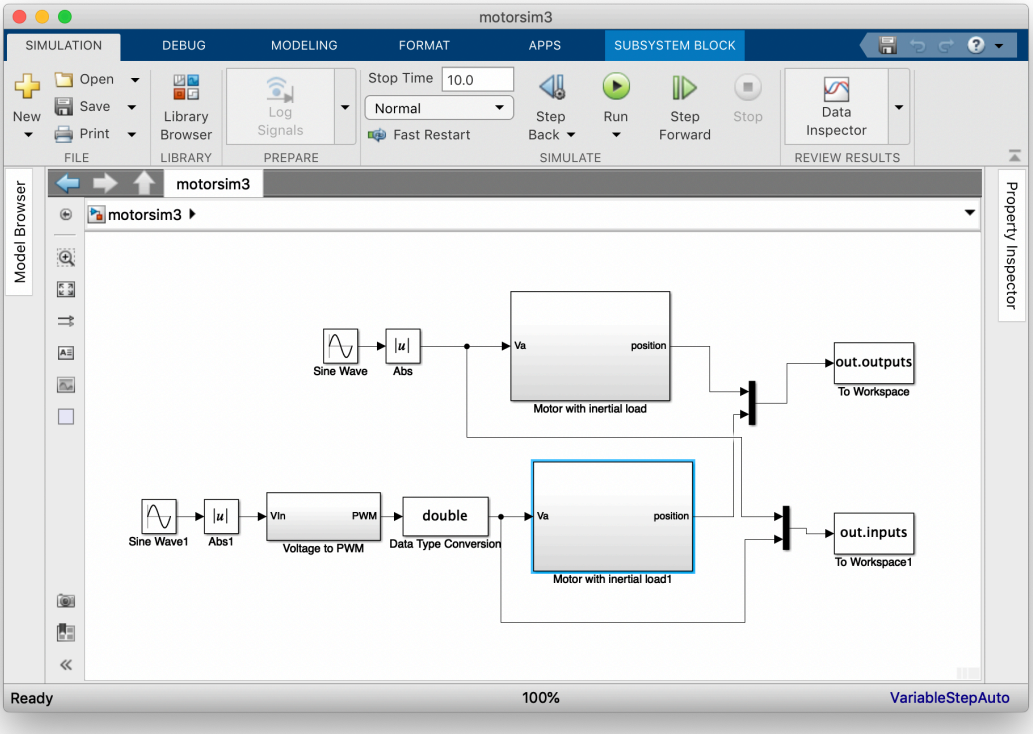
This is called pulse width modulation. With an h-bridge configuration, the polarity of the voltage can also be changed, allowing both positive and negative V_a. Read the PWM tutorial, and if you want, try using PWM to varying the intensity of a LED.

We can test the difference between a continuous V_a and PWM V_a using Simulink. Copy the motor subsystem block so that there are two copies. Type `scdpwm` at the command line [1]. This should open up a Simulink block diagram that contains a block called Voltage to PWM. Copy this block into your motor simulation. Double click on the Voltage to PWM block, and then double click on the triangle block. Multiply the time values: entry by 25, as shown below. This will make the period of the PWM signal to be 0.25 seconds.



The output of the Voltage to PWM block is a boolean, which can sometimes cause errors. Look in the Signal Attributes Folder in the Simulink Library Browser, and drag the Data Type Conversion block into your block diagram. Double click on this block, and select the Output data type to be double. Connect everything up so that is looks like the following block diagram.

[1] scdpwm.slx is a simulink block diagram comes with the simulink control design toolbox. If your installation of Matlab does not include this, you can get a copy of this file from the simulation and control assignment page on Canvas

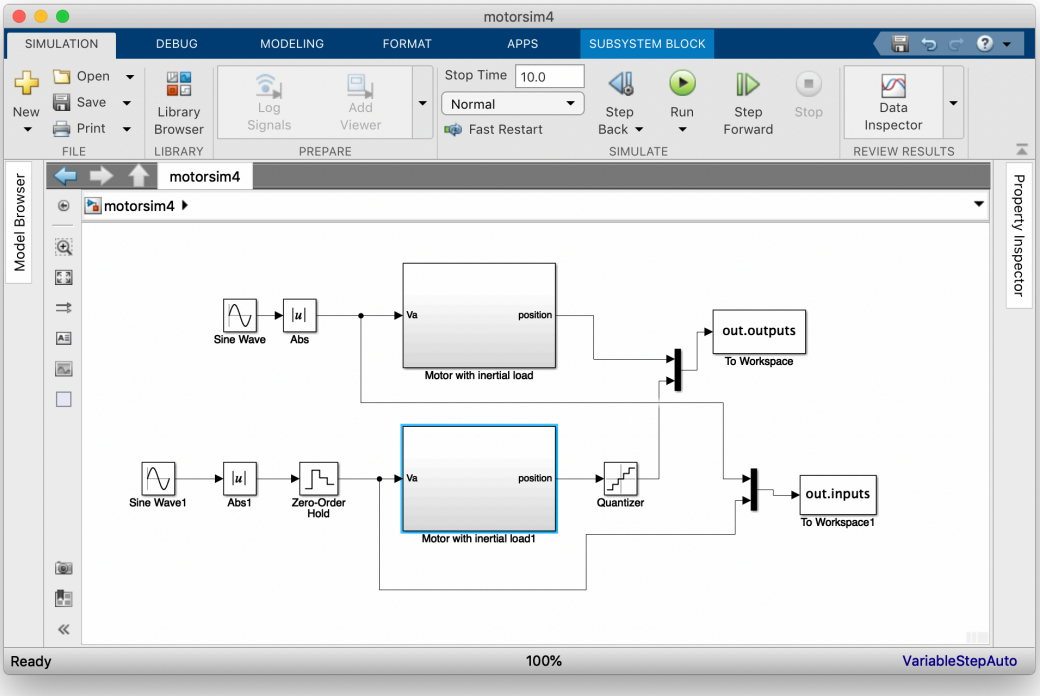


Create a script that will run the simulation, and plot the input and output signals. You should find that although the inputs look quite different, the outputs are almost identical. Be prepared to explain the results. Ironically, although PWM has advantages in the real world, the discontinuous PWM signal is difficult for Simulink to handle, so in the future, when we simulate the motor, we will use a continuous signal that represents the average of the actual PWM input.

3.3 Sampling and Quantization

A microcontroller is a digital system, so that measurements are discretized both in time and magnitude. Signals that are recorded at a periodic rate are called sampled, and signals that are recorded with finite precision (i.e. 10 bit representation) are called quantized. Sometimes a low sampling rate or high quantization error can cause issues, and we will want to be able to simulate their effect in Simulink. In addition, when the microcontroller sets an output signal, it can only vary this output signal at a set rate. This is called a zero-order hold, and results in a stair-case type of signal.

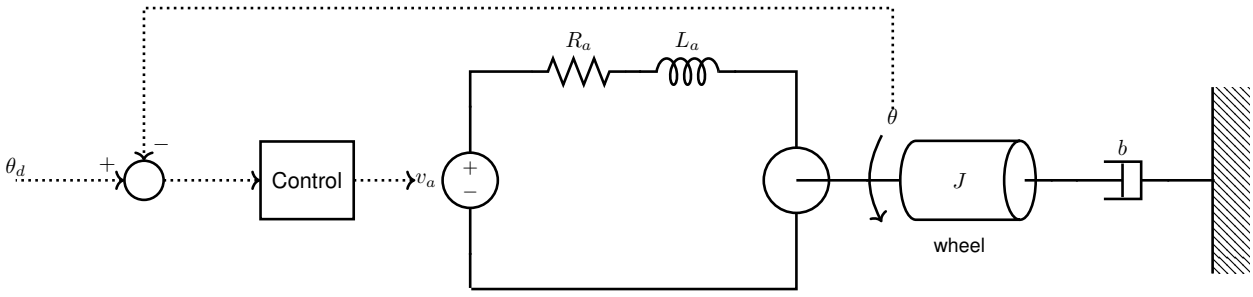
Make a copy of your simulink block diagram from the previous section. As discussed, the PWM signal is well approximated by a constant that is equal to the average value of the PWM, so we will remove that part, and insert a zero order hold (from the discrete library and quantizer block (from the discontinuities library). The default sample time for the zero order hold is 1 second, and the default quantization is .5, which are both fairly large.



Create a script that will simulate the block diagram and compare the inputs and outputs.

4 Problem 2: Feedback Control

For the second part of this assignment, you will simulate a feedback controller that can regulate the angular position of the motor shaft. A measurement of the position θ is compared to a desired position θ_d. The error between the two is fed to a controller, which then decides on the voltage to apply to the motor. A diagram of the setup looks like this



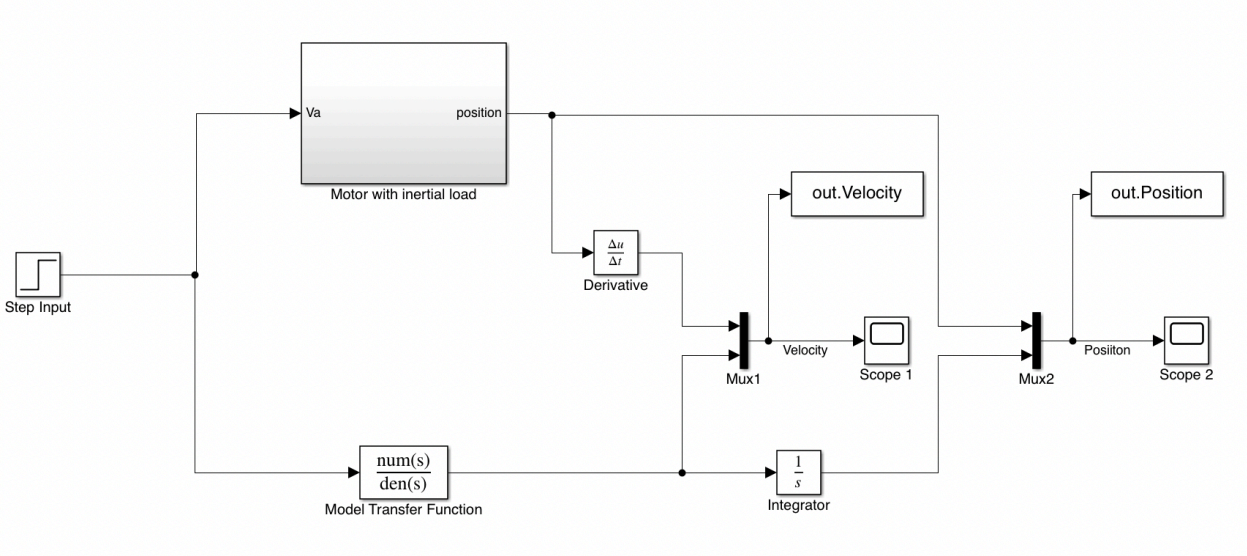
4.1 Finding the system transfer function

In order to design and test a controller, you need a model of the system to be controlled. In the first problem you created a simulation of the motor system using linear blocks, and it would be possible to use block diagram simplification to find the system transfer function. However, let's suppose you did not have access to all of the system parameters in the model (e.g. R_a, K_t, etc.) and you needed to find the system transfer function from experimental data.

A key experiment for dynamic modeling is the step response. In this experiment, the system input in increased by a constant amount, and the system output, or response, is recorded. The control designer then adjusts the parameters of a

transfer function until the step response of the transfer function matches the step response of the system. At this time, go through the EENG307 Notes: Lecture 15, System Identification, which is available on Canvas, to see examples of step response experiments, and matching those experiments to data.

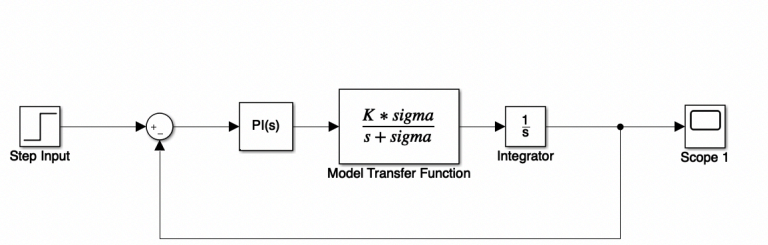
Since you are trying to match a simulation, rather than an experiment, you can set up your Simulink block diagram to contain both the system you are trying to match, and your parameterized transfer function, making the tuning process easy. A suggested setup is the following:



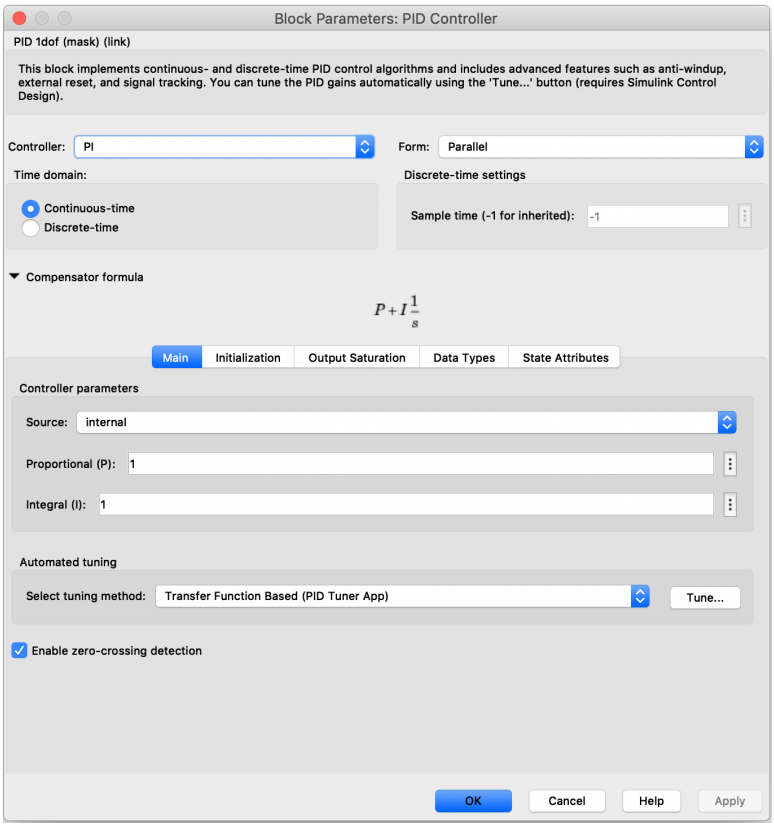
Notice that we are comparing both the velocity and position. The expected form of the transfer function of the motor with inertial load from voltage to position is $\frac{\Theta(s)}{V_a(s)} = \frac{K\sigma}{s(s+\sigma)}$. Since this transfer function has a pole at $s = 0$, the step response is unbounded. However, the transfer function from voltage to velocity is $\frac{\Theta(s)}{V_a(s)} = \frac{K\sigma}{(s+\sigma)}$, and stable. You can use the velocity response to try and tune the parameters K and σ, and then verify that the position response also matches.

4.2 Tuning the controller

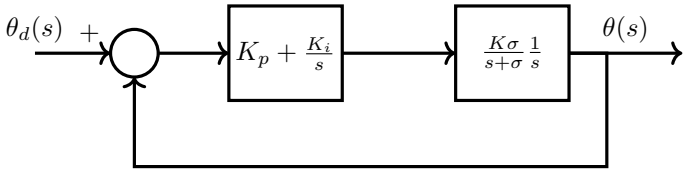
Once you have values for K and σ, create another simulink model that contains the motor transfer function (including the integrator) and a PI controller as shown below. The step input supplies the desired position (θ_d), while the actual position (θ) is read at the scope.



To get a PI controller, insert a PID controller block, double click on the block to open up the block parameters. In the menu next to "controller:", at the upper left, select PI.



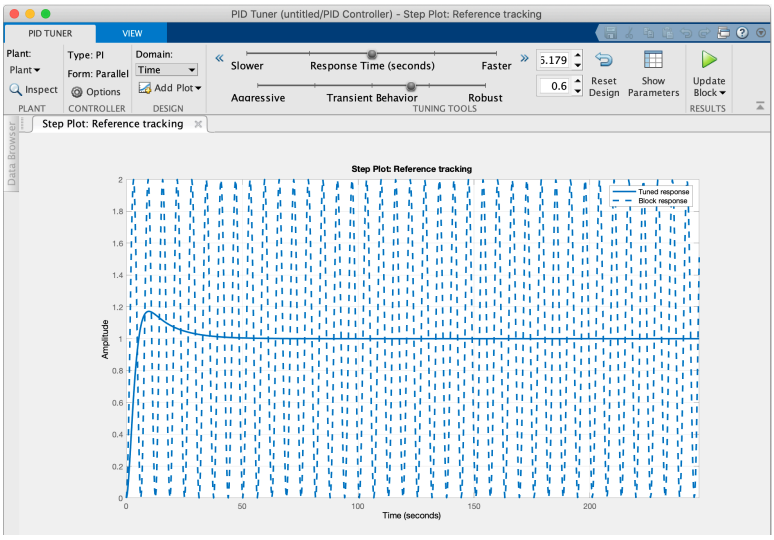
The simulink model implements the following block diagram.



You can verify that the closed loop transfer function is

$$\frac{\theta(s)}{\theta_d(s)} = \frac{K\sigma(K_p s + K_i)}{s^3 + \sigma s^2 + K\sigma K_p s + K\sigma K_i}$$

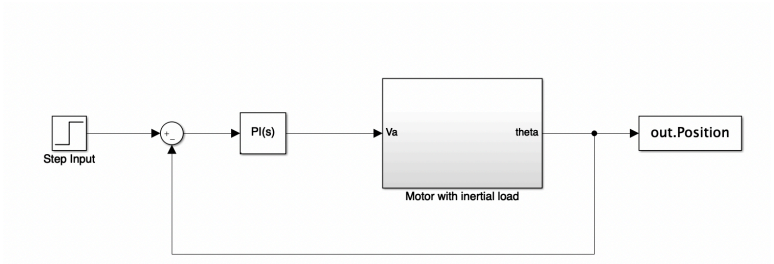
Because the closed loop system is third order, it is more difficult to choose K_p and K_i than if it were second order. Fortunately, Simulink provides tuning method. In the PI controller block parameters screen, click on "Tune..." in the lower right hand side. The following screen will appear.



The dotted line shows the closed loop step response if you use the current gains (may not be displayed if current gains given unstable closed loop) while the solid line is the response using tuned gains. You can adjust the tuning using the sliders at the top, to change the speed of response, and the transient behavior, which is mainly the overshoot. Adjust the sliders to achieve as fast a response as you can with less than 10% overshoot. You can right click on the plot and select "characteristics" to see the step response characteristics. Click update block to save you gains to the PI control block, and then you can close the tuning window. Click OK to accept the new gains, and you can run the Simulink simulation to verify that you get the desired response.

4.3 Implementing the controller on the full simulation

While the gains may work on the transfer function model that you identified, you will also want to verify that the controller works with the original detailed simulink model. Create a new Simulink block diagram, copying in both the detailed model and the PI controller, hooking them up in unity gain feedback as shown.



Name the signals that represent the desired position, voltage V_a, and actual position. Add To Workspace blocks that will save these three signals. Create a script that will simulate the system, and plot the results.

4.4 Implement a PD controller

Repeat the process in the section, but this time implement a PD controller. Be sure to save your files/results for the PI controller separately, as you will be turning in both.

5 Documentation

For each simulation, create a document that describes the form and function, with enough detail that someone else could recreate the results, and also discusses the significance of the results. This documentation should include the following:

- A simulink block diagram that simulates the DC motor driving a wheel with bearing friction, with a script to define constants and run the simulation.
- A simulink block diagram that compares PWM and continuous signals, with a script to define constants and run the simulation.
- A simulink block diagram that compares sampled and quantized signals, with a script to define constants and run the simulation.
- A simulink block diagram that compares the step response of a detailed DC motor model with the transfer function model that you have tuned, with a script to run the simulation and plot the results.
- A simulink block diagram that implements a PI controller on the detailed DC motor model, with a script to run the simulation and plot the results
- A simulink block diagram that implements a PD controller on the detailed DC motor model, with a script to run the simulation and plot the results

One easy way to create these documents is to have well documented code, and to use the publish function of the editor. You can watch a video demonstration of this. The publisher can create a variety of document formats, for submissions select a .pdf format. These documents should include the Simulink block digram, system and simulation parameters, plots of the results, **and an interpretation of the results (for example, using comments)**. Save your documentation for yourself, and upload a copy to the assignment link on Canvas.

6 Demonstration

Demonstrate each of the simulations listed in the Documentation section above. Be prepared to change parameters or adjust the simulation in response to questions from the TA or instructor.