Lab 1 , SFWR ENG 3DX4

Modeling a DC Motor and Introduction to Simulink and Quarc Library

First lab week of: Jan. 22, 2023
Prelab Due by the end of day: Jan. 29, 2023
Demo Due by End of Lab Period, Week of: Jan. 29, 2023

1 Announcements

In this lab, you will be evaluated on prelab questions and demonstration of working laboratory exercises. Evaluation of lab exercises will be performed on a checkpoint system, with checkpoints indicated in your laboratory instructions by the phrase "show your work/output to your TA." In order to receive full marks for a lab, you must show a TA your work/output at each checkpoint, and your work must be correct and complete. In addition, lab work must be completed in the allotted laboratory time. Prelab questions are to be completed *individually*, Lab exercises are to be performed in groups of two.

2 Laboratory Goals

- Introduction to the Simulink graphical environment, and Quanser Quarc Library.
- Simulating a plant using transfer functions.
- Use a PID controller to control the position of a simulated dc motor.
- Observe the step response of a plant with a feedback controller, and learn to take step response metrics (settling time, overshoot and steady state error).
- Observe the effects of adjusting proportional, integral and differential gain (K_p, K_i and K_d) on step response metrics.
- Design a PID controller to meet given design criteria.

3 Introduction

As discussed in lecture, open loop systems only produce output by transforming input, and do not incorporate previous output values or sensor feedback. As a result, open loop systems are sensitive to disturbances, and cannot be precisely controlled.

By measuring the output response of a system and comparing it to a reference, an open loop becomes a closed loop system. Feedback allows not only for robustness in the face of system disturbances, but also allows much finer control over the system's response characteristics.

In this lab you will design a PID controller to control the position of the shaft of a DC electric motor. The objective of this controller is to produce a step response with the following specifications:

- Settling time must be less than 0.04 seconds.
- Overshoot must be less than 16%.
- The system must be free of steady-state error.

PID controllers will be discussed in lecture, and can also be found in section 9.4 of Nise. Settling time and percent overshoot are defined in section 4.6. Steady state error is defined in chapter 7. With respect to steady state error, we will be dealing exclusively with step input response for the purposes of this lab.

Once built, a PID controller must be "tuned" to produce a desired step response:

1. First observe the open loop response and note what needs to be compensated.
2. Adjust proportional gain K_p to improve rise time.
3. Adjust differential gain K_d to improve percent overshoot.
4. Adjust integral gain K_i to eliminate the steady-state error.
5. Finally, if the step response still does not meet the specified criteria, perform minor adjustments of K_p, K_i and K_d until it does.

4 Prelab Questions

The general open loop transfer function which models the angular velocity $\omega(t)$ of a motor is:

$$G_\omega(s) = \frac{\omega(s)}{U(s)} = \frac{A}{\tau s + 1}$$

where A and τ are positive, real valued constants.

1. What is the transfer function for the angular position of a motor $\theta(t)$?

2. What, if any, is the steady state value of $\omega(t)$ in open loop response to a step input:

$$u(t) = \begin{cases} U_o, & t \geq 0 \\ 0, & t < 0 \end{cases}$$

5 Laboratory Exercise

5.1 Modelling a DC Motor

Analog, or continuous, control systems may be tested either physically or in simulation. To simulate a control system, we require a software system in which to build a model (in this case Simulink), and a mathematical model of the device we wish to simulate. In this lab, we will use a transfer function in the Laplace domain to simulate a DC motor. Specifically:

$$\frac{\theta}{V} = \frac{K}{s((Js+b)(Ls+R)+K^2)}$$

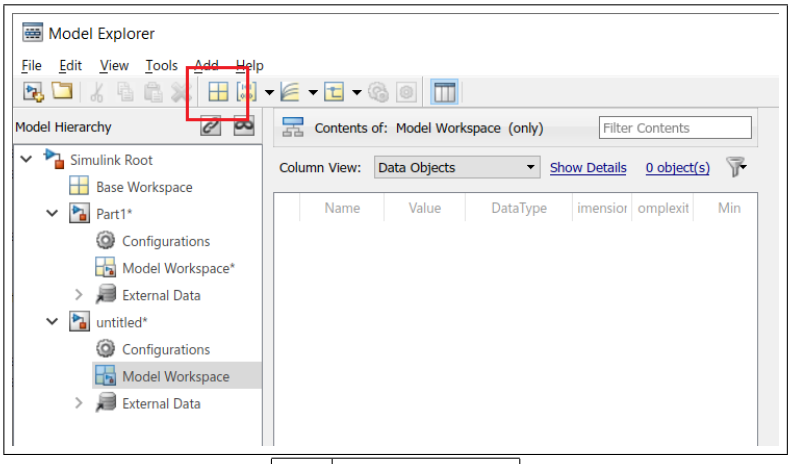
In order to create a model in Simulink that represents the above transfer function, proceed as follows:

- Launch MatLab and Simulink.
- From the Simulink launch screen, create a new blank model.
- Rearrange the above equation such that it is of the form:

$$G(s) = \frac{n_0}{d_0 s^3 + d_1 s^2 + d_2 s + d_3}$$

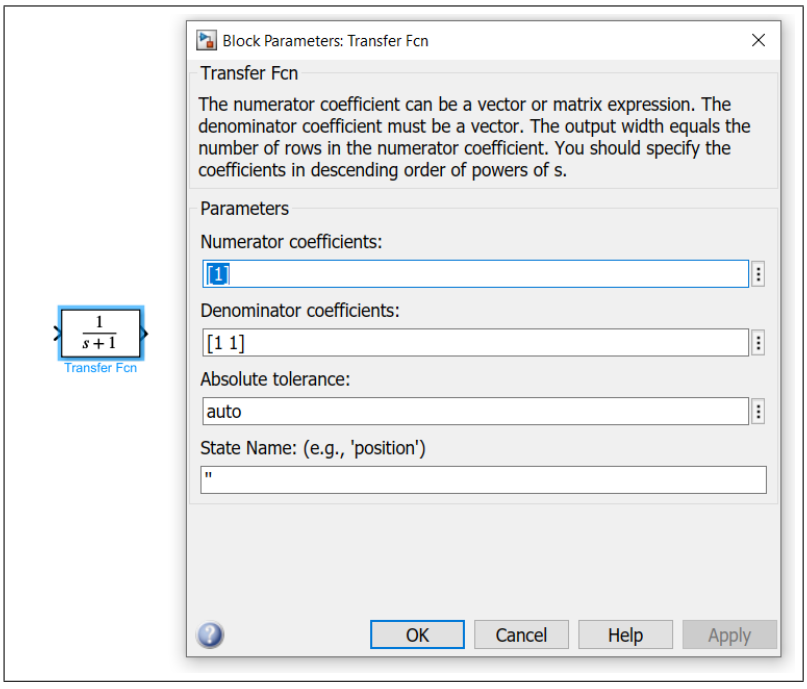
  Where the numerator and denominator coefficients $d_0 ... d_3$ are expressions containing only the constant values K, J, b, L and R. Make a note of these expressions, as we will use them to construct our transfer function shortly.

- Let's set up our constants inside the Simulink workplace. From the toolbar, select **Modelling >> Model Explorer**.
- In the model hierarchy, select the model you are working on (probably called "untitled"). Then, select "model workspace" within the "Model Hierarchy" pane.
- Add the following variables, with the corresponding values. Variables may be added using the "Add MATLAB Variable" button in the toolbar:



J	$3.2284e-6$
b	$3.5077e-6$
K	0.0274
R	4
L	$2.75e-6$

- Now we may construct our transfer function. In your Simulink model, create a **Transfer Fcn** (Simulink / Continuous). Double clicking on it will reveal it's configuration window:



- The coefficients of the numerator and denominator expressions are given as vectors. We are not restricted to constants or variables in Simulink, we may use full expressions here! Enter the expressions you derived above in the correct order to create the transfer function. Note: You can check your work by expanding the size of the transfer function block in the model view (expand

to a size that is large enough!). When you do, you will be able to tell if the right coefficient ended up with the right power of s.

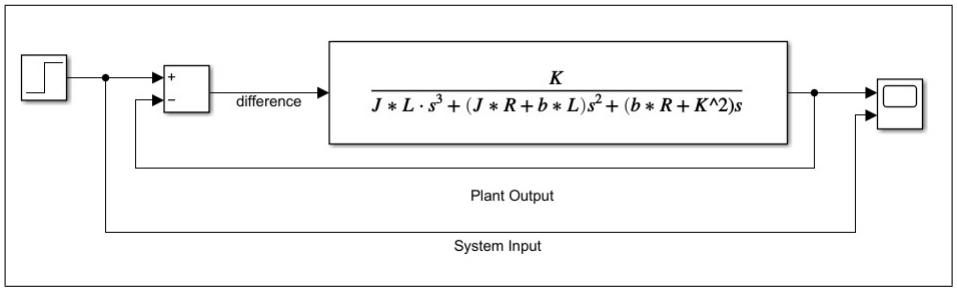
- Now let's hook our transfer function up to an input and take a look at the output.
- "Step Response" is the system's response to a "Step Input", which is a piecewise function which starts at zero, and "steps" to some specified value. This simulates a user requesting a certain motor position, for example. Create a **Step** (Simulink / Sources) block as our input. Configure it with a step time of 0.02s, and leave the rest of the parameters unchanged.
- Connect the step block's output signal to the transfer function's input port.
- Next, create a **Scope** (Simulink / Sinks)
- Hook the transfer function output up as the first scope input, and the input signal as the second.
- Set the overall simulation time to 0.2s and run the simulation.
- The TA will ask you the following questions, please prepare your answers and then check in with the TA to complete *CHECKPOINT 1*
  - What does the graph represent? What does the first derivative of this graph represent, and what would it look like? (please describe)
  - What is represented by the non-linear section of this graph?
  - By observing the graph, what is the steady state error of this response, assuming we wish to control the angular position of the motor?
  - What is the percent overshoot of this response?
  - what is the settling time of this response?
  - Is this response stable with respect to angular position?

5.2 Creating a Closed Loop System

In Part 1, we implemented *open-loop control*, meaning that we gave the system an input, and merely observed the output. In this part, we will design a system that has some degree of control (this being a class in control systems).
One very commonly used mechanism to achieve system control is, rather than feeding the input directly into the plant, read the current state of the plant, as well as the input, and feed the *difference between the two* into the plant. This way, the larger the difference between where you are and where you want to be, the harder the plant works to correct it.
Let's set up our plant from Part 1 with **feedback.**

- Create a subtract block.

- Disconnect the step input from the transfer function and feed it into the "+" terminal of the subtract block.
- Branch the plant's output line, and feed it into the "-" terminal of the subtract block.
- Feed the output of the subtract block into the plant's input terminal. Your system should look something like the following:



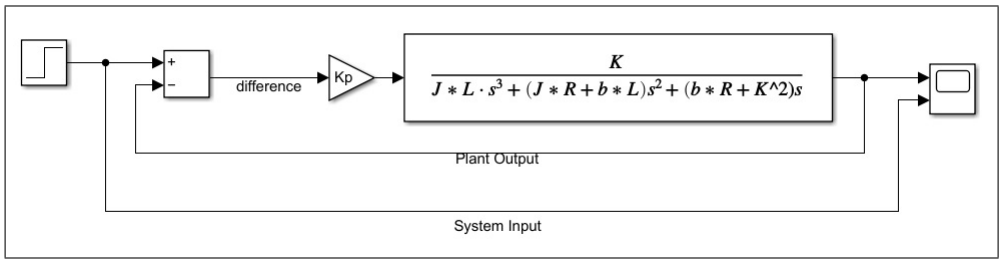
- Keep the step response step time at 0.02s, but modify the final value to 60. Since the output of the transfer function is the angular position of the motor, this changes the input from being measured in Volts to being measured in Degrees. In effect, we are asking the motor to acquire the position of 60 degrees.
- Change the simulation stop time to 0.4s.
- Run the simulation, and take a look at the scope output. Answer the following questions:
  - Approximately how much does the plant output overshoot the input by? How much does it overshoot by as a percentage of the steady state value?
  - How long, would you say, does it take the response to "stabilize"?
  - Steady state error is the error at steady state. i.e., the difference between the plant output and the system input once the plant output has stabilized. Does this plant exhibit steady state error?
- Report your answers to these questions to your TA to complete *CHECKPOINT 2*.
- For your own interest, you can also feed the signal labelled "difference" on the diagram above, and observe how it changes as the output value of the plant changes.

5.3 Proportional Gain

In Part 2, we've improved our system so that it is reactive to it's own output, but we still aren't exhibiting much *control* over it. Let's see if we can, for example, reduce or eliminate the overshoot observed in Part 2.

In electronic systems, electronic signals are often controlled using amplifiers. The degree to which an amplifier scales the input signal to produce the output signal is known as the **Gain** of the amplifier. Gain is something we can control, just like adjusting the volume on your headphones. We can control how much the difference signal affects the plant by introducing **proportional gain**.

- In the model explorer, add a variable to the model workspace called `Kp`. Give it a value of 1.7.
- Add a **Gain** block to the model (Simulink / Math Operations), and set the gain to Kp.
- Place the gain block between the difference signal coming from the adder and the input to the plant's transfer function. It should look like the following:



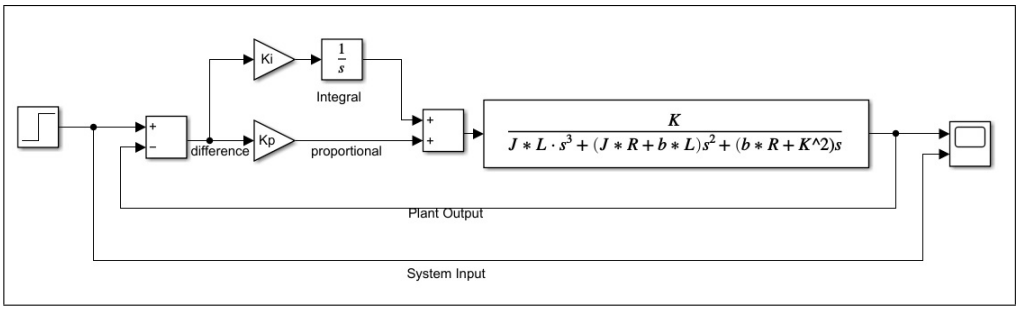
- Change total simulation time to 0.3s
- Run the simulation and take a look at the output.
  - How has the overshoot and settling time been effected?
- Modify the value of Kp in the model explorer until the system no longer produces an overshoot. Report this Kp to your TA, as well as your answer to the question above to complete *CHECKPOINT 3*.

5.4 Integral Gain

In Part 3, we demonstrated that we can have a bit effect on our system's characteristic response by introducing an amplifier. This amplifier is our first **controller**. Let's make our controller more robust!
So far we haven't had to worry about steady state error. This is because our model of a DC motor contains an integrator term that is already compensating for the steady state error. However, if we consider angular velocity rather than angular position (i.e., if we consider the derivative of the current transfer fucntion), we'll be able to see the error!
In this part we will introduce the **Integral Gain**. Roughly speaking, the integral component of the error signal is proportional to the *integral* of the difference between the input and output signals. The integral gain *accumulates* error over time. If the system has steady state error, that error will accumulate over time in

the integral component. The more error is accumulated, the stronger the signal to the motor which attempts to compensate it, thus the steady state error is reduced over time, and eventually eliminated entirely.

- Let's begin by modifying our plant so that we have some steady state error to compensate (and therefore know when we've done so.) In the transfer function's properties dialogue, remove the terminating zero from the denominator coefficients vector. As you can see from the formula presented on the block, this lowers each power of s in the denominator by 1. This is effectively the same as multiplying the entire equation by s, which, if you recall your Laplace transformations, *is* taking the derivative of the equation.
- Leave Kp at whatever you found in Part 3.
- Modify the step time to 0.01s, and the total simulation time to 1s.
- Run the simulation. Note how the output never reaches the input!
- Now let's add an integral component to our controller. Create a new variable within the model workspace called Ki. Give it a value of 1.
- Create a new gain block, an **Integrator** block (Simulink / Continuous), and an add block.
- Assign K_I to the value of the gain block.
- Route the difference signal into the gain, the gain into the integrator, and the integrator into one of the inputs of the add terminal.
- Remove the wire from the K_P gain block into the plant, and reroute it into the adder. Route the adder output into the transfer function. When finished, it should look like the following:

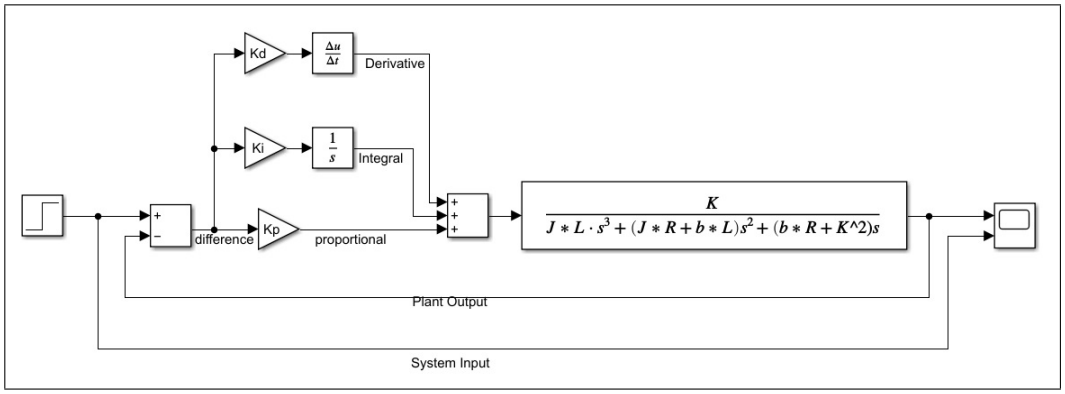


- Run the simulation and take a look at the scope. Note how, over time, the response approaches the desired value, somewhat asymptotically. This takes some time, however.
- Without modifying the proportional gain, modify the integral gain until the response reaches the desired value within 0.3s. It may be helpful to modify the total simulation time to 0.35s to see more clearly what's going on.
- Record this value and present it to your TA to complete *CHECKPOINT 4* .

5.5 Differential Gain

We've covered the P and the I of our PID controller, the only component remaining now is D, the differential gain. Differential gain will send input to the plant proportional to the *derivative* of the difference signal. That is, proportionally to the rate of change. The faster the response is changing, the more is contributed by the differential gain. While this can drastically improve the response time, overdoing it can cause controller instability. "Jittery" motor movement can be a result of an improperly configured differential gain.

- Leave K_I and K_P at the values you found in the previous parts.
- Add a new variable, `Kd` to your workspace, and give it an initial value of 0.01.
- Add a gain block, and assign K_D to it.
- Add a **Derivative** (Simulink / Continuous) block.
- Wire the difference signal into the differential gain, and the differential gain into the derivative block.
- Wire the output of the derivative block into the same adder that the proportional and integral branches are using.
- You should get something like the following:



- Congratulations! You have manually created a PID controller! Run the simulation and note the results. Notice how overshoot has been re-introduced.
- Tune the differential gain downwards until the overshoot has just been eliminated. Take a note of this gain.
- You may notice around this gain that the output has acquired some degree of irregularity. Normally, a physical system with inertia will reject this type of low-level noise, or it will be barely perceptible, but it is interesting to note that that this jitter has only appeared after the introduction of differential gain.
- Zoom into this noise in the scope graph, and present it to your TA, along with your differential gain value to complete *CHECKPOINT 5*.

5.6 Simulink Quarc Library

QUARC is Quanser's rapid prototyping and production system for real time control. It is integrated seamlessly with Simulink to allow Simulink models to be run in real-time on many types of targets, including NI MyRIOs, which are used in our lab.

In our following lab projects in this course, we will use Simulink and QUARC library blocks to develop Simulink models, and deploy and run the models in real-time on NI MyRIOs, which act as the DAQ devices to interact with Quanser's SRV02s or Quanser's Ball and Beams.

The basic procedure to build Simulink/Quarc models and run the models on targets is here:
`https://docs.quanser.com/quarc/documentation/quarc_procedures.html`

Some useful information for Quarc MyRIO support, such as the channel numbers used for configuring Simulink Quarc blocks for specific MyRIO pins, can be found here:
`http://quanser-update.azurewebsites.net/quarc/documentation/ni_myrio_1900.html`

The complete documentation for Quarc can be found here:
`https://docs.quanser.com/quarc/documentation/quarc.html`

As an introduction, a simple Simulink model, **SRV02.slx**, built with Quarc library blocks is provided in this lab folder on Avenue. In the model, a sinusoidal signal is send to SRV02's motor, which will drive motor gears to turn back and forth at a frequency of 0.5 Hz. The analog electrical signal from the potentiometer, which measures SRV02 gear's position, is read and displayed on a scope. The pulse count from the quadrature encoder, which can be used to measure the load gear's position change, is read and is shown in a display box. The pulse count is processed and the load gear's velocity is calculated and ploted on a scope. Please download this model to your Z: drive or to other local drives and open it using Simulink. Try to understand this model with the help from Simulink or with the help from Quarc documentation. Take this chance to browse the Simulink Quarc library to get to know what blocks are available from Quarc Library.

Make hardware connection according to the model settings. Ask help from TAs if you are not confident at this stage. Turn power on to your MyRIO and your power supply module for your SRV02.

In Simulink, go to QUARC >> Preferences. Then on the Model tab, modify the "Default Target URI" by changing the IP address section to your MyRIO's IP address. If wireless connection is used in our lab, your MyRIO's IP address is on the sticker on the MyRIO. Save your model.

Run the model by clicking the "Monitor & Tune" button under the "HARDWARE" menu in Simulink. Then select and click "Monitor & Tune" to build and deploy the model to MyRIO and run the model. Stop the model by hitting the stop button under the "HARDWARE" menu. Show to your TA to complete *CHECKPOINT 6*.

6 Closing Notes

In this lab, we focused on reducing the plant's response time to fractions of a second, but there are some physical considerations you need to be aware of, if you apply this style of control to a physical system. Unlike simulations, physical systems are subject to physical laws, such as $F = ma$. Be aware that the faster you ask your plant to react, the greater the internal forces will be within that plant. While it is satisfying to try to have the plant output match a step response as closely as possible, bear in mind that you are essentially asking the machine to teleport between two positions. As the amount of time a motor (for example) has to transition itself between two positions decreases to zero, the amount of acceleration, and therefore force, that must be applied to the motor in order to achieve it approaches infinity. You can very easily damage or destroy expensive equipment that may or may not belong to you by requesting behaviours that aren't physically possible.