Lab 4: Closed-Loop DC Motor Control

Introduction

Your team should have found the open loop transfer function of your DC motor system in Lab 3. In this lab, you will place your DC motor under feedback control and begin experimenting with PD control. Additionally, you will simulate the closed-loop response of your DC motor system and overlay those simulation results with experiments.

Verification of Open-Loop Results

There are two things that your team was supposed to finish in Lab 3 that are essential to this week's lab. Your team must

1. be able to run open-loop pulse tests with correct signs using your Arduino
2. have the open-loop transfer function of the DC motor system

You cannot successfully complete this lab unless your system is setup so that a positive input to your motor h-bridge results in positive change in your encoder output, as shown in Figure 1. Verify that for an open-loop pulse test, positive commands lead to positive increases in the encoder output and negative commands lead to decreasing encoder output.

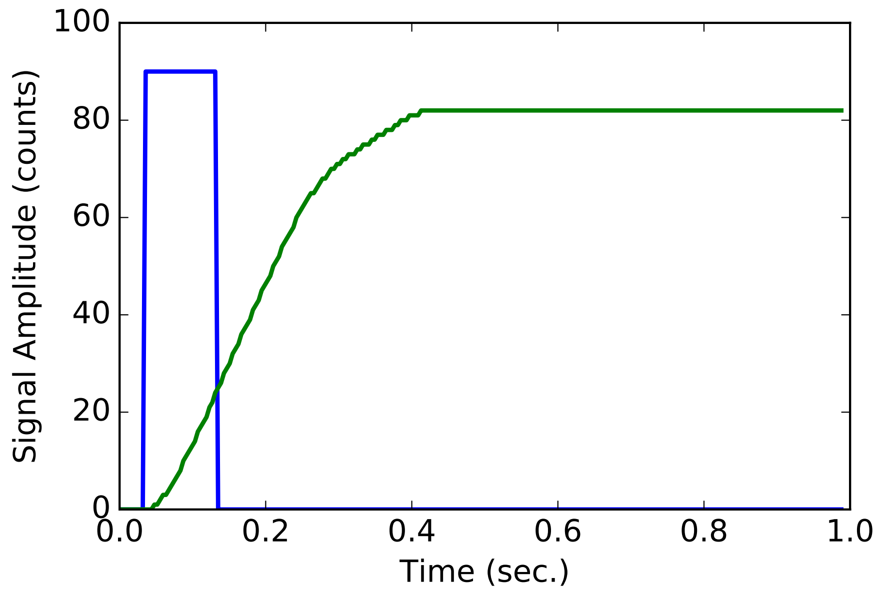

Figure 1: An open-loop pulse test showing correct positive signs.

Software Updates

You will once again need to update Dr. Krauss' Python modules:

```
pip install --upgrade krauss_misc
```

```
pip install --upgrade py_block_diagram
```

```
pip install --upgrade pybd_gui
```

Mac Users: You probably need to use `pip3` instead of `pip`.

Learning Objectives

Students will

- implement feedback control using the `pybd_gui` approach
- develop a basic understanding of how the `Kp` and `Kd` gains of a PD controller impact the step response of a DC motor
- learn how to find a closed-loop transfer function
- simulate the closed-loop response of a DC motor system

P Control

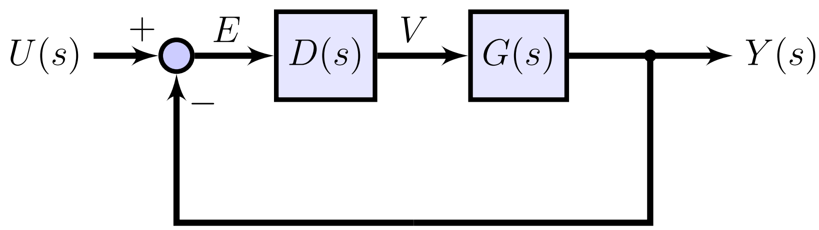

Figure 2: Block diagram of a general feedback control system.

A general feedback controller is shown in Figure 2. $G(s)$ is the transfer function of the DC motor system, also known as the plant. For P control, $D(s) = K_p$.

You will need to create a closed-loop block diagram model that has a DC motor plant, a step input, a summing junction, and `P_controller` block. You will also want to insert a saturation block between D and G or you may get strange results. Note that output blocks are still not really supported.

Your block diagram model should probably look very similar to the one shown in Figure 3, though yours with have a `P` controller rather than `PD` at this point.

Running Tests with Different `Kp` Values

Once you have the model created, use it to perform step response tests with different values of `Kp` and observe the effects of `Kp` on the step response. In order to efficiently run tests with different `Kp` values, set the `Kp` value of your controller block as a menu parameter so that the Arduino asks you for the `Kp` value before each test. To do this, go to the top menu for `pybd_gui` and chose “Block Diagram –> Set Menu Parameters”.

- run tests with multiple values of `Kp` and find values that lead to responses that are lightly damped, moderately damped, and over-damped
- investigate what happens with large values for `Kp`

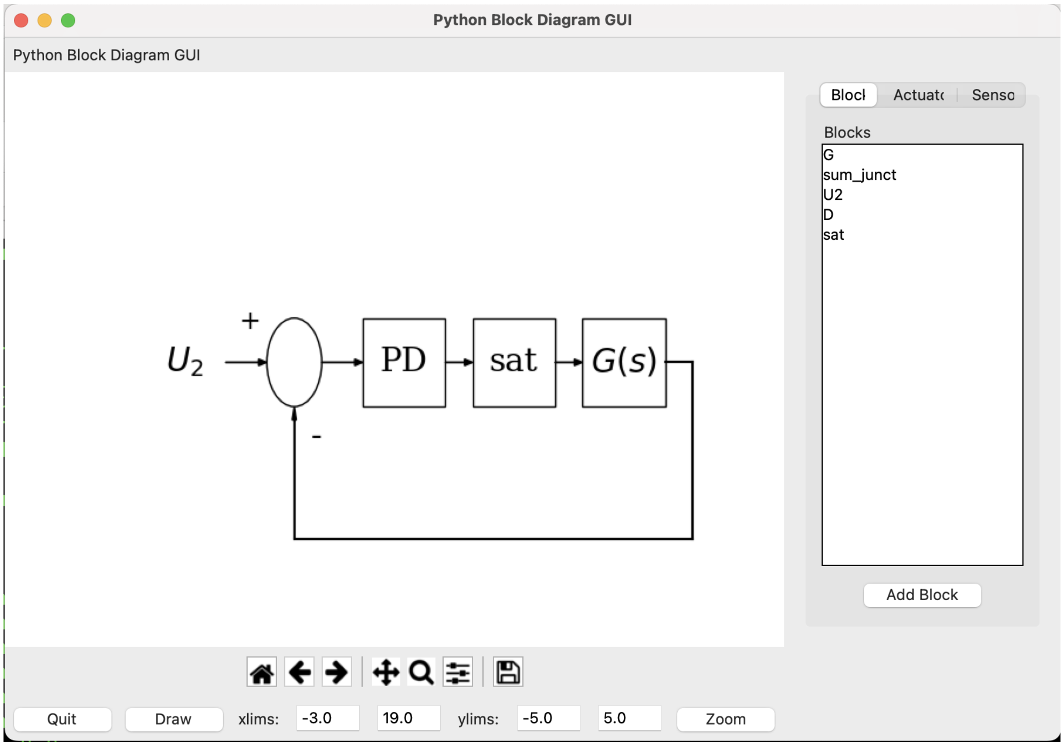

Figure 3: Block diagram of the DC motor system created in 'pybd_gui'.

You may want to consider running the tests from a Jupyter Notebook using `pyserial` and Dr. Krauss' `serial_utils`. This will allow you to plot data very quickly and easily after running a test. See the notebook `jupyter_nb_test_runner_v1.ipynb` as a starting point.

Developing Intuition: Second Order Step Responses Varying ζ and ω_n

In order to further develop your intuition regarding how ζ and ω_n affect the step response of a second-order system, overlay two groups of step responses:

- holding ω_n constant, zary ζ
- holding ζ constant, zary ω_n

What do you learn?

Simulations: P Control

Once you have found values for `Kp` that lead to lightly damped, moderately damped, and over-damped step responses, overlay those experimental results with simulations from Python. Find the `CLTF` based on the open-loop transfer function $G(s)$, known values for `Kp`, and `control.feedback`. It will probably be cleanest to use `control.forced_response` with the `CLTF` and the closed-loop input vector u from the Arduino data.

PD Control

Once you have proportional control working, create a new block diagram to perform PD control (proportional + derivative control). Do this by using a `PD_controller` block.

- choose a `Kp` value that leads to a lightly damped response and then gradually increase `Kd` and observe how the response changes
- Again, it will be best to run these experiments from a Jupyter Notebook so that you can quickly and easily plot the results.

For PD control, the output of the controller is

$$v(t) = K_p\, e + K_d\, \dot{e} \tag{1}$$

Simulations: PD Control

In order to use the Python `control` module to simulate a system under PD control, you need to find the transfer function associated with equation 1.

If

$$D(s) = \frac{V(s)}{E(s)}$$

what is $D(s)$ for PD control?

Once you have found $D(s)$, use it with $G(s)$ to find the `CLTF`. Then overlay PD simulation results with experimental data.

Comprehension Questions (30 points)

1. If you wanted to command a DC motor to rotate a specific number of encoder counts without using feedback, how would you do it? What would be the disadvantages of such an approach?
2. How do the the PD gains K_p and K_d each affect the step response of a DC motor?
3. What are the limitations of using P control by itself without the D term when controlling a DC motor?
4. Find the `CLTF` for P control by hand, leaving K_p as a variable. How does changing K_p affect ζ and ω_n? As K_p affects ζ and ω_n, how do those changes affect the step response?
5. Find the `CLTF` for PD control by hand, leaving K_p and K_d as variables. How do K_p and K_d affect ζ and ω_n? What should the effects on the step response be ideally?
6. When doing PD control of a DC motor, why would it theoretically be a good idea to have fairly large values for K_p and K_d? What happens in practice if K_p and K_d are too large?