Massachusetts Institute of Technology

Department of Mechanical Engineering

2.004 *Dynamics and Control II*

Laboratory Session 4:
Closed-Loop Position Control, and the Effect of Derivative Control Action

Laboratory Objectives:

(i) To investigate closed-loop control of the angular *position* (θ) of the rotational plant.

(ii) To investigate the manipulation of closed-loop dynamic response through the use of derivative (D), and proportional + derivative (PD) control.

(iii) To compare your experimental results with Simulink simulation.

Introduction: In Lab 3, we investigated closed-loop control of angular velocity of the rotational plant. In this lab, we switch to position control with the goal of commanding the flywheel to move to a given angular position (see Appendix A). We will see that, for this particular plant, proportional control alone does not generate satisfactory transient behavior, and that the use of PD (proportional + derivative) control allows us to manipulate the closed-loop poles to achieve much improved response characteristics.

Introduction to PD Control: Proportional + derivative (PD) control is used to manipulate the closed-loop poles to improve the transient response of closed-loop systems. A PD controller has a transfer function

$$C(s) = K_p + K_d s$$

with a block diagram and a time-domain response function shown below

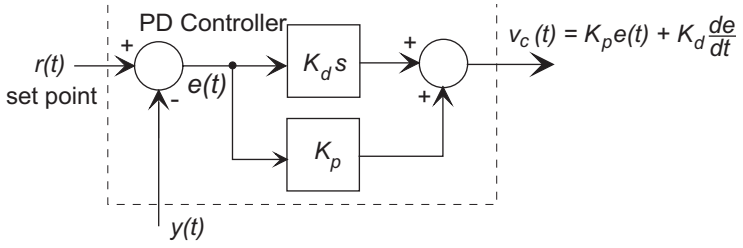

Figure 1: PD controller block diagram.

$$v_c(t) = K_p e(t) + K_d \frac{de(t)}{dt}$$

where $v_c(t)$ is the controller output.

A description of how the derivative component acts to modify the transient response is given in Appendix B. In practice pure derivative action is rarely used. Appendix B explains why, and briefly describes the use of a *band-limited differentiator*, or *pseudo-differentiator*. Please take a few minutes to read through and understand the Appendices. Before we run the actual experiments we will perform simulation to examine the performance of several PD controllers.

Experiment #1: Create a Simulink Simulation:

Simulink is one of the most widely used computer tools for control system analysis and design. It is an integral part of MATLAB, and is a drag-and-drop block-diagram time-domain simulation language. Simulink provides a graphical workspace where you can create very complex system models without writing a single line of code. Here you will build a Simulink model of the closed-loop system (shown below) to simulate closed-loop responses.

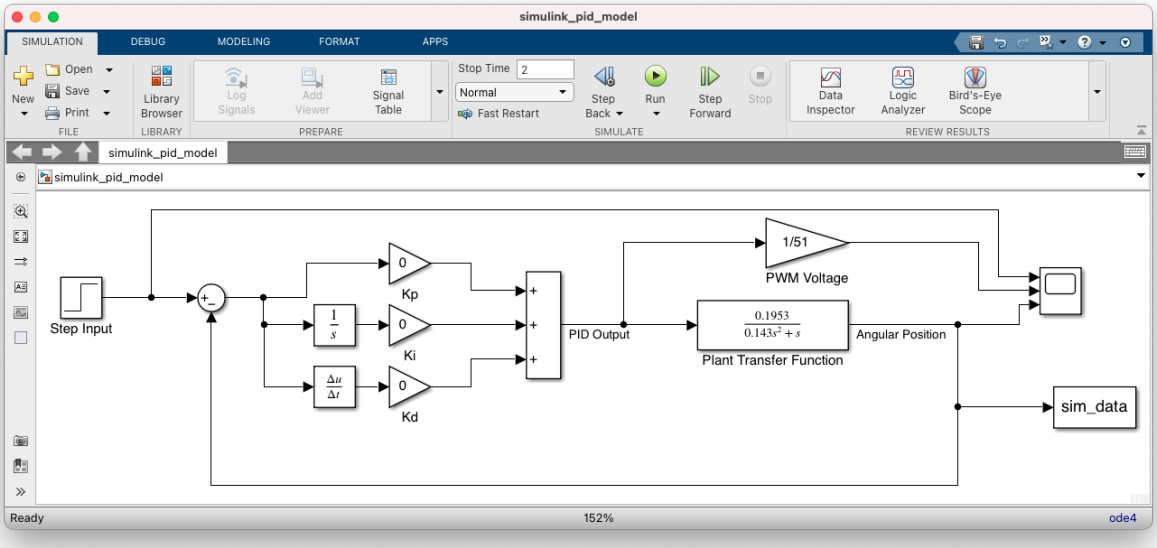

Figure 2: Simulink closed-loop block diagram.

(a) Open the Simulink template file and replace the blocks for the PID controller and plant transfer function with those shown in Figure 2.

(b) Open the "Library Browser" in Simulink to find the blocks needed to construct the model. Most blocks can be found under the "Commonly Used Blocks" or "Continuous" blockset. Drag a block from the library browser to your model and make any necessary changes. You can change parameter values or properties of a block by double-clicking it. See Appendix A for additional information on the plant transfer function.

(c) Once the model is complete, run the simulation for the three cases: (i) $K_p = 100$ and $K_d = 2$, (ii) $K_p = 100$ and $K_d = 10$, (iii) $K_p = 100$ and $K_d = 20$. Set $K_i = 0$ for

all three cases. Note that the response data from a simulation run is automatically saved to a variable named "sim_data" that you need to rename to a unique variable for each run to prevent overwriting the existing data in that variable. You can also double click the block and change the variable name before each run. Also note that the "sim_data" variable is of "timeseries" data type which is a data structure that contains both time array and data array. To plot the time response data you can simply type `plot(sim_data);`. If you wish to parse each individual array out you can type `x = sim_data.time; y = sim_data.data;`.

(d) Save the simulation data from each case for comparison with actual data to be obtained in Experiment #3.

Experiment #2: Proportional Control of Position:

Most shaft encoders, such as the one used in this DC motor, are *incremental* – that is, they do not have an inherent absolute zero position. You can set the current position of the flywheel as zero position whenever you reset the Arduino by re-uploading the code or pressing the "RESET" button on the Arduino.

Set up your controller with proportional control ($K_i = 0$, $K_d = 0$) for $K_p = 75, 100, 150$, all with a desired position of $\pi/2$ radians. Enable square wave setpoint so as to command the wheel to rotate between 0 and $\pi/2$ radians periodically. Make a single plot of all the responses. Would you classify the response as "satisfactory" based on the speed of the response? Examine your closed-loop characteristic equation and think in terms of the pole locations in the complex *s-plane*. Look at the closed loop transfer function, is it possible to move the closed-loop poles farther into the left half-plane so as to control the settling time T_s?

Experiment #3: PD Control:

(a) Start with $K_p = 100$ and $K_d = 2$. Again use a square wave with an amplitude of $\pi/2$ rad and save the step response. Select a complete positive step section of the response and create a single plot that contains both the simulated and the actual sensor response data. You may find the Matlab command `hold on` useful to plot multiple traces on the same plot.

What is the peak PWM command to the motor? Is the controller "saturating"? Is the response with PD control more satisfactory than responses with proportional only control?

(b) Keep $K_p = 100$, repeat (a) with $K_d = 10$ and then $K_d = 20$. In each case, make a plot of the step response and compare with simulation.

(c) Compare your three plots. Briefly describe how the value of K_d has affected 1) any "overshoot" in the step response, 2) the time to the peak response, and the time to reach the steady-state response. Relate K_p and K_d to the closed-loop transfer function from your prelab (can also be found in Appendix B).

(d) Compare your experimental data with the simulated data from Experiment #1. For each case, comment on any discrepancies between simulation and actual experiment.

Experiment #4: PD Controller Design:

Here we ask you to design a PD controller, using the closed loop transfer function, to produce a closed-loop response that will meet the following requirements, and test the controller with a given setpoint.

(a) The closed-loop step response has $\zeta = 1.0$ and $\omega_n = 15$ rad/s.

(b) Use $\pi/2$ as the desired position and command the wheel to follow a square wave.

(c) Test the controller with `SINE_WAVE` and/or `POT` setpoint.

★ Extra Credit Task ★:

Program a setpoint profile that mimics the "second" hand on a clock. Specifically, command the wheel to increment 36 degrees every second in the same direction.

Using a gear reduction (or program it to advance every 6 degrees) this motion can be scaled to look like a second hand on the clock. Note, however, that this is not actually how clocks work since this is a relatively power intensive method.

Appendix A: The Plant Transfer Function for Position Control

In previous labs we found the open-loop plant transfer function relating angular velocity and PWM command to the motor to be

$$\frac{\Omega(s)}{U(s)} = \frac{K}{\tau s + 1}$$

where $\Omega(s)$ is the angular velocity of the flywheel and $U(s)$ is the command signal to the motor in PWM (controller output). We have measured or calculated the following values:

- $\tau = 0.18$ sec
- $K = \frac{K_{dc}}{vc2pwm}$
- $K_{dc} = 14.6$ rad/V
- $vc2pwm = 51$ V/pwm

Since angular displacement θ is the integral of angular velocity Ω,

$$\frac{\theta(s)}{\Omega(s)} = \frac{1}{s},$$

the transfer function for this week's experiments is

$$G_p(s) = \frac{\theta(s)}{U(s)} = \frac{1}{s} \cdot \frac{\Omega(s)}{U(s)} = \frac{K}{\tau s^2 + s},$$

which is a second-order system with a pole at the origin of the s-plane.

Appendix B: Introduction to Derivative Control Action

Consider the experimental second-order system with a transfer function

$$G_p(s) = \frac{K}{\tau s^2 + s}$$

with the PD controller

$$G_c(s) = K_p + K_d s.$$

Then, the closed loop transfer function becomes

$$\begin{aligned} G_{cl}(s) &= \frac{G_c(s)G_p(s)}{1 + G_c(s)G_p(s)} \\ &= \frac{K(K_p + K_d s)}{\tau s^2 + (1 + KK_d)s + KK_p}. \end{aligned}$$

During this lab session you will see that with proportional control, the system is very lightly damped, leading to an unsatisfactory transient response. However, with PD control the characteristic equation is

$$\tau s^2 + (1 + KK_d)s + KK_p = 0.$$

Notice that each of the feedback gains affects only a single term in the polynomial. For an underdamped system the real part of the poles is affected only by K_d. Therefore, K_d may be used to control the settling time of the system. K_p may be used to control the undamped natural frequency of the system, thus with the two parameters the designer has much greater ability to set the transient response of the closed-loop system.

Note that the PD control action has introduced a zero at $s = -K_p/K_d$, which must be taken into account when computing the transient response.

The most obvious way of computing a derivative in a digital computer is by using the backward difference, that is

$$\frac{de(t)}{dt} \approx \frac{e_n - e_{n-1}}{\Delta T}$$

where e_n is the error measured at the nth iteration, and ΔT is the time step.

In practice pure differentiation is rarely used, because this amplifies high frequency noise. For example, consider the derivative of a sinusoid, $f(t) = \sin(\omega t)$,

$$\frac{df(t)}{dt} = \omega \cos(\omega t)$$

and note that the amplitude of the output is proportional to the frequency ω. Thus, if there are any unwanted high frequency noise components in a signal, it will be highly amplified by a differentiator, and, if passed on to the plant, can cause many operational problems. Therefore, practical PID controllers take preventative action to combat the noise. It is common to

- use a smoothing (averaging) filter to minimize high frequency components in the error signal before differentiation, or
- use a *band-limited* or *pseudo*-differentiator algorithm to approximate a differentiator at low frequencies, but not respond to high frequency inputs.