MAE106 Laboratory Exercises
Lab # 5
PD Control of DC motor position

University of California, Irvine
Department of Mechanical and Aerospace Engineering

Goals

Understand how to implement and tune a PD controller to control the position of a DC motor.

Explore the frequency response of the PD controller by testing how it responds to sinusoidal inputs of different frequencies.

Parts & equipment

Qty	Part/Equipment
1	Seeeduino board
1	Motor driver
1	DC motor with encoder

Introduction to PD Control

The most common controller used by engineering designers to control the movement of a motorized part is the PD (proportional-derivative) controller (sometimes an integral control term is added to create a PID controller, but we will not explore I control in this lab). In this lab you will implement a PD position controller. Such controllers are also used in robot arms, radars, numerically controlled milling machines, manufacturing systems, and control surfaces on aerospace vehicles. The PD control law is:

$$\tau = -K_p(\theta - \theta_d) - K_d(\dot{\theta} - \dot{\theta}_d)$$

where:

θ_d	desired motor angular position	τ	desired motor torque
$\dot{\theta}_d$	desired motor angular velocity	K_p	proportional gain
θ	actual motor angular position	K_d	derivative gain
$\dot{\theta}$	actual motor angular velocity		

Note that the controller has two terms – one proportional to the position error (the “P” part), and one proportional to the derivative of position (i.e. velocity, the “D” part). Thus, it is called a “PD” controller. The implementation of this controller for a DC motor with inertia, J, is shown in Figure 2.

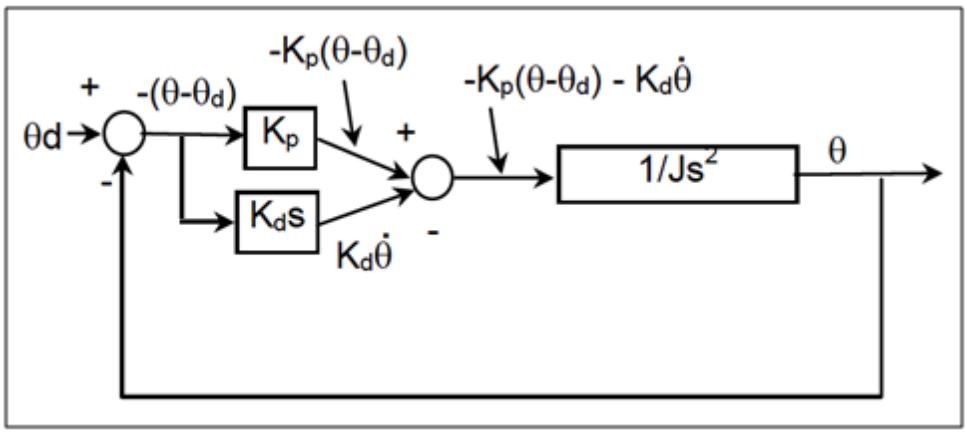

Figure 1. Block diagram that you will implement to make the PD controller for the motor. J is the inertia of the motor shaft.

To understand how the actual system behaves we need to first understand its dynamics. First, let's look at the dynamical equation that describes how θ evolves with time when the controller is attached to the motor.

Dynamics of the motor and shaft:

$$\tau = J\ddot{\theta}$$

Dynamics of the controller system:

$$\tau = J\ddot{\theta} = -K_p(\theta - \theta_d) - K_d(\dot{\theta} - \dot{\theta}_d)$$

Re-writing to make input-output clear:

$$\boldsymbol{J\ddot{\theta} + K_d\dot{\theta} + K_p\theta = K_d\dot{\theta}_d + K_p\theta_d}$$

This differential equation has similar dynamics to a mass-spring-damper system with Force as the input and Position as the output. That is, it follows the same equations of motion. This allows us to use our intuition about mass-spring-damper systems when designing and tuning a PD controller.

Recall that the differential equation of motion for a mass-spring-damper system is given by:

$$m\ddot{x} + B\dot{x} + Kx = F$$

and thus using the analogy to the PD controller we have that:

mass-spring-damper system		**PD controller**	
m	mass	J	motor inertia
B	damper	K_d	derivative control term
K	spring	K_p	proportional control term
F	input force	τ	desired motor torque

In a mechanical system, if you wanted the system to respond more quickly, you would increase the natural frequency (ω_n) by picking a stiffer spring (higher K). Which variable would you change in your differential equation for the PD system to make your system respond more quickly (i.e. increase its natural frequency)?

Note: there is a limit to how big you can make this variable because of the time delays in this sampled data system.

By adding the derivative gain (K_d) to control the position of the motor we must now take into account the concept of damping when designing and implementing the controller. With damping in the controller we can have four types of behaviors:

Undamped (i.e. zero damping): The system oscillates at its natural frequency. These oscillations are a function of the controller's gain, K_p.

Underdamped: The system will move to its desired position and oscillate about this position with is oscillations gradually decreasing to zero.

Critically damped: The system will move to its desired position as quickly as possible without oscillating.

Overdamped: The system will move asymptotically towards its desired position without oscillating, but at a slower rate than the critically damped case.

Suppose you didn't want your motor to oscillate too much. *This is an important issue!* You usually want your motor to go to a desired value quickly and accurately without oscillating. You will have to change the derivative gain K_d until you get the system to be critically damped.

Part I: Step response of the actual system

Construct the Arduino circuit in Figure 2. Download the code for this lab from the website and run it on the Arduino. As in previous labs, when you open the Serial Monitor (or when you press the 'reset' button on the Arduino), the code will re-initialize, then run a step response and send time, position, and desired position of the motor to the monitor for the duration of the response. You can copy and paste this into Excel to display the step response.

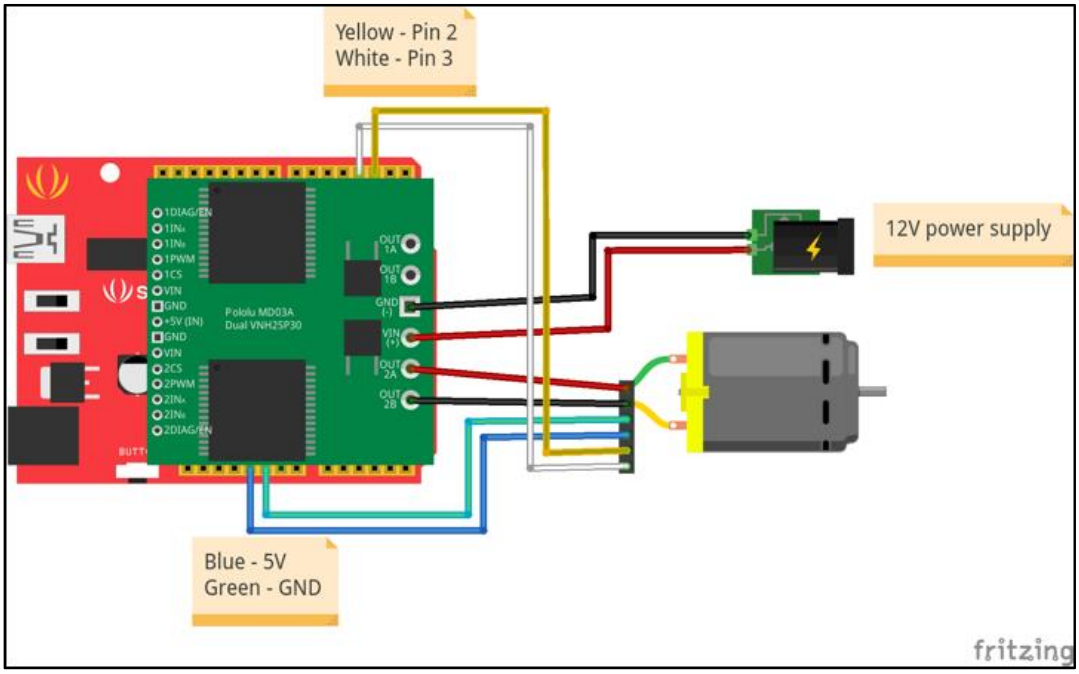

Figure 2. Schematic of connections to motor driver. Same as in the previous lab.

To control your motor you will have to go through a process of 'tuning' the gains of the controller. You tune a PD controller based on the response that you want to get from your system (i.e. how fast do you want it to reach its desired position? Do you want to allow oscillations around the desired position?) You can follow a procedure such as this one:

1. **Set the derivative gain, Kd, to zero and only change the proportional gain, Kp**. Change Kp so that you have an acceptable response in regards to how fast your system reaches its final position. Note that since Kd is zero you will get oscillations around the desired position (undamped system). *Hint: Kp=100 is a good value to start with.* **Increase Kp it until you get a rise time (defined in Figure 3) close to 0.038 seconds**.
2. Once you have an acceptable response in terms of how fast your system reaches its final position, you can now begin to add damping so that the oscillations will decrease (underdamped system). **Increase the value of Kd to achieve the desired response. Do not change Kp!** Because you are now adding damping to your controller, you can expect your system to react slower.
3. You can keep increasing the derivative gain until your system no longer oscillates as it reaches its desired position. This will give you a critically damped system. What happens if you keep increasing the damping gain past this point?

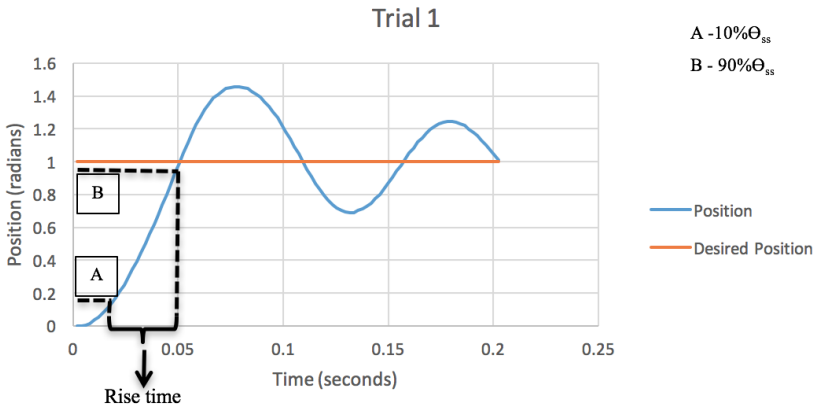

Figure 3. Underdamped system. The rise time is the time it takes for your system to go from 10% to 90% of the desired position.

Practical Exam I

Show your TA that your system is critically damped and that it has a rise time close to 0.038 seconds (your system should respond similar to Figure 4).

- What is the time constant at these settings?
- What is the value of Kp and Kd that you used?

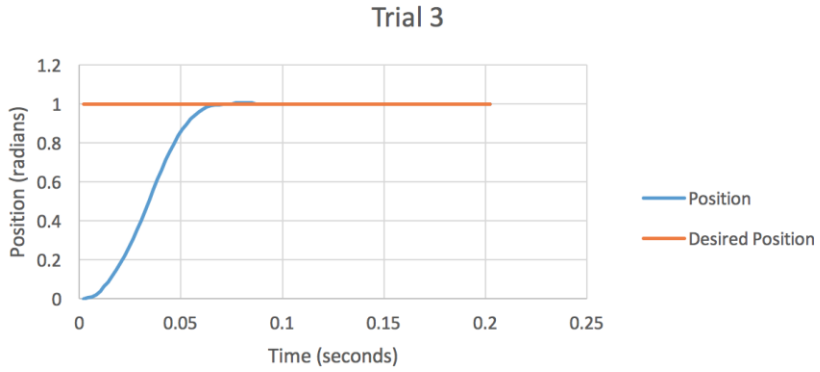

Figure 4. Expected output for the critically damped system.

Part II: Frequency response of the actual system

The goal of this part of the lab is to characterize the frequency response of the system. In particular, you will explore how well the system tracks the desired position when the input is a sinusoid, across a range of frequencies. Remember, you can view linear systems such as this one as "filters". The PD controller acts like a low-pass filter, although it has a resonant peak if the damping is not great enough.

To characterize the frequency response of the system, follow the steps below. **REMEMBER: DO NOT LET YOUR MOTOR OVERHEAT**.

1. Set the variable 'desPos_amplitude' to 1.
2. Set Kp equal to the value that you found in Part I for a rise time close to 0.038 seconds and Kd = 0. Set the variable delayToTakeData = 2.0. This will allow the controller to run for 2 seconds before saving data to the virtual oscilloscope in the code, allowing initial transients to die out.
3. Change the Arduino code to input a sinusoid. To do this, change the Boolean 'stepInput' to FALSE. Record the output amplitude and phase shift at 1, 2, 4, 8, 12, and 16Hz *(you will need these for your write-up)*. You can change the frequency using the variable '*desPos_frequency*'.
    a. Does the system have a resonant frequency? You may have to search between the frequencies suggested above to find it. What should the phase difference between desired and actual position be at the resonant frequency? (Answer: 90 degrees; Hint: You could use this to more accurately find the resonant frequency).
    b. Which signal should be larger at the resonant frequency?
    c. What is the resonant frequency? Do you notice any high frequency oscillations in your output signal? What do you think might be causing those?

Practical Exam II:

Show your TA that you can find the resonant frequency of your system. What is the frequency, in Hz, at which resonance occurs? What would happen if you used the derivative gain from part 1?

For the write-up you will need to save the data you just collected. You will also need to collect data while the system is critically damped. Do this by setting K_d to the value you found in part I. Collect data at the same frequencies you did before (1, 2, 4, 8, 12, 16 Hz). Read the write-up prompt below before leaving lab to make sure you have all the data you need.

Write-Up

1. Following the procedure in Part I for tuning the Proportional and Derivative gains, make 3 plots of your system. One for each case of Underdamped, Critically Damped, and Overdamped. Label the value you used for Kp. Label each of the values of Kd that you used to obtain the three behaviors of damping.
2. Make two Bode plots of the frequency response of the system:
    a. One for the undamped system ($K_d = 0$).
    b. Another where the system is critically damped. Indicate the value of K_d you used.

    To make each Bode plot you will plot:

    - 20log(output amplitude/input amplitude) vs. input frequency (on a log scale), and
    - phase shift vs. input frequency (on a log scale).
    - Hint: In MATLAB you can use the 'loglog' and 'semilogx' plotting functions.
    - See the example below for a rough idea of what is expected.