MAE106 Laboratory Exercises
Lab # 4 - P-type digital control of a motor

University of California, Irvine
Department of Mechanical and Aerospace Engineering

Goals

Understand how to create a P-type velocity controller using a DC motor coupled with a rotary encoder.

Parts & equipment

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

Introduction

As a mechanical and/or aerospace designer, you will sometimes want to control actuators on a machine, such as a robot, car, plane, or space vehicle, to make the machine move like you wish. The purpose of this laboratory exercise is to learn how to use a microcontroller (e.g. an Arduino) to control a motor. One of the major benefits of using a microcontroller is that you can express the control law and control gains in software. Changing the control law just involves typing in new software code. This will be really useful for your final project. Note that an alternate way to set up controllers is with analog circuit elements, such as op-amps. Such controllers respond more quickly than a computer, but changing parameters requires changing resistors or capacitors. With the decreasing cost and increasing speed of processors, it's more common now to implement controllers with microcontrollers.

The basic idea of computer-based control is to:

1. Electronically measure the system performance (In this lab, the system performance in which we are interested, i.e. the thing we are trying to control, is the angular velocity of the motor shaft. We will measure this with a rotary encoder)

2. Read this measurement using the microcontroller (In this lab, you will read the pulses from the encoder into two of the digital inputs of the Arduino using 2 interrupt service routines (ISRs)). If the output of your sensor were analog, you could read it using one of the Arduino's analog pins;

3. Calculate an appropriate control law in a program running on the microcontroller;

4. Generate an output voltage that is sent to the motor.

Rotary encoder

A rotary encoder is a device that converts the rotation of its shaft into digital voltage pulses. Two common types of rotary encoders are optical and Hall effect encoders. Consider an optical encoder first (Figure 1), as they are easier to understand. An optical rotary encoder is made of a rotating disk with slits cut in it, and a light emitter shining light through the disk toward a light sensor. The disk is connected to the shaft of the encoder. As the shaft rotates, the disk alternately blocks or allows light to enter the light sensor, depending on whether there is currently a slit in front of the light emitter or not. Thus, each time a slit passes the light emitter, the light sensor outputs a pulse of voltage. For an "incremental encoder" or "relative encoder" such as the one we are using in lab, the microcontroller measures the amount of rotation by counting the pulses being sent by the encoder. Incremental encoders typically have two light emitters and two light sensors, with two tracks of slits, which allows them to determine the direction of motion using "quadrature encoding".

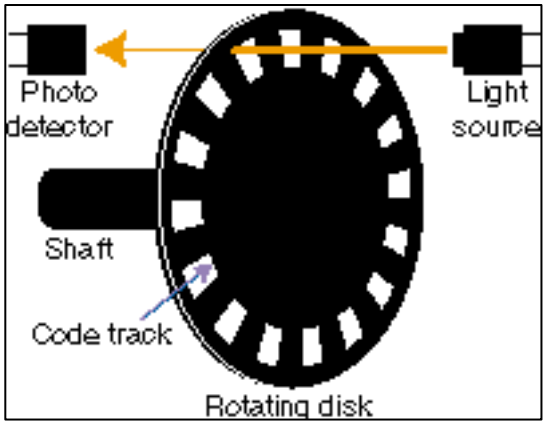

Figure 1. Optical encoder (image from National Instruments)

Hall effect sensor

For the motor in this lab, the rotary encoder is a Hall effect encoder, which produces voltage pulses just like an optical encoder (Figure 1). The difference is that the pulses are produced when a magnet on the shaft passes close to a Hall effect sensor. A Hall effect sensor make use of the Lorentz force law. When a magnetic field comes close to the sensor, it creates a Lorentz force on the charges flowing through the sensor, pushing them to one side of the sensor, thereby causing a small voltage difference in the direction perpendicular to current flow (see Figure 2). This voltage changes with the proximity and polarity of the magnetic field, so by putting magnets on a shaft, turning the shaft will cause the Hall effect sensor to output voltage pulses.

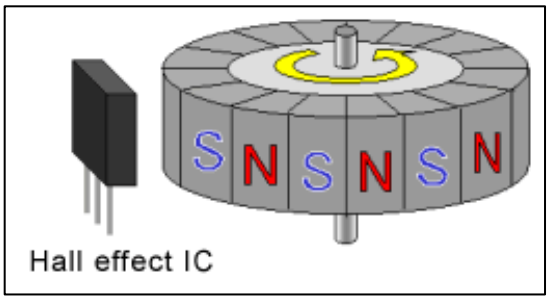

Figure 2. Hall effect sensor (image from http://www.akm.com/akm/en/product/add/magnetic_sensors/0029/)

For a detailed tutorial on using a Hall effect sensor see:

http://www.electronics-tutorials.ws/electromagnetism/hall-effect.html

Interrupts

Interrupts are hardware signals that are used to temporarily stop the main program and allow a special program to run. Many microcontrollers, including the Arduino, have dedicated hardware that can continuously monitor a voltage on a pre-defined input line, called an "interrupt line" (your Seeeduino has 2 interrupt lines). When the voltage on the interrupt input line changes, this hardware immediately tells the microcontroller to stop what it is doing and run the "Interrupt Service Routine" (ISR), which is a piece of code that does something in response to the "interruption" indicated on the interrupt line. ISR's are useful when you have a device connected to a microcontroller that provides rapidly changing information at random times, such as rotary encoders. Rather than having your main program periodically check the encoder (in which case it might miss a pulse if it isn't checking often enough), you set up an ISR to respond to the encoder immediately when it provides new information.

Reading from the encoder

There are six wires coming out of the encoder:

- Red wire: motor power,
- Black wire: motor power,
- Green wire: encoder GND,
- Blue wire: encoder VCC (5 V),
- Yellow: encoder A output,
- White: encoder B output.

The vendor of the DC motor provides instructions on using the encoder that is coupled to your DC motor (see section "Using the Encoder" in: https://www.pololu.com/product/3214, see also the video on the website: http://gram.eng.uci.edu/~dreinken/MAE106/res/motorEncoderDemoMAE106.mp4, and Figure 4).

Make sure that you understand what happens when you spin the motor in both directions.

Figure 4. Output from Encoders A and B shown in an oscilloscope. (Image from: https://www.pololu.com/product/1442)

Part I: P-type control of the velocity of the DC motor

We will use the encoder to measure the velocity of the motor and use the measurement as a feedback signal to control the motor.

Use the code provided here:

http://gram.eng.uci.edu/~dreinken/MAE106/labs/Lab4Pvelocity.ino,

to control the velocity of the motor. **Go through it and relate the parts of the code to the steps mentioned below.**

To implement the P-type (proportional-type) velocity feedback controller. The program needs to:

1. read the signals from the encoders,
2. compute the velocity of the motor,
3. calculate the control law, and
4. send the control signal to the motor driver.

Since we are able to measure the velocity of the motor, the next step is to implement the control law and send the appropriate signal to the motor driver.

The equation that describes the control law is:

$$u = -K_p(\omega - \omega_d)$$

where:

$$u = control\ signal\ to\ the\ motor\ driver,$$

$$K_p = proportional\ gain,$$

$$\omega = motor\ velocity,$$

$$\omega_d = desired\ motor\ velocity$$

The basic idea of this controller is to measure the velocity error of the motor shaft, and then to apply torques *proportional* to those errors (hence the name proportional-type controller), so that the motor shaft turns at an angular velocity closer to the desired angular velocity. Note that if the error is zero, the correction torque to be applied is zero; if the error is large, the correction torque to be applied is large.

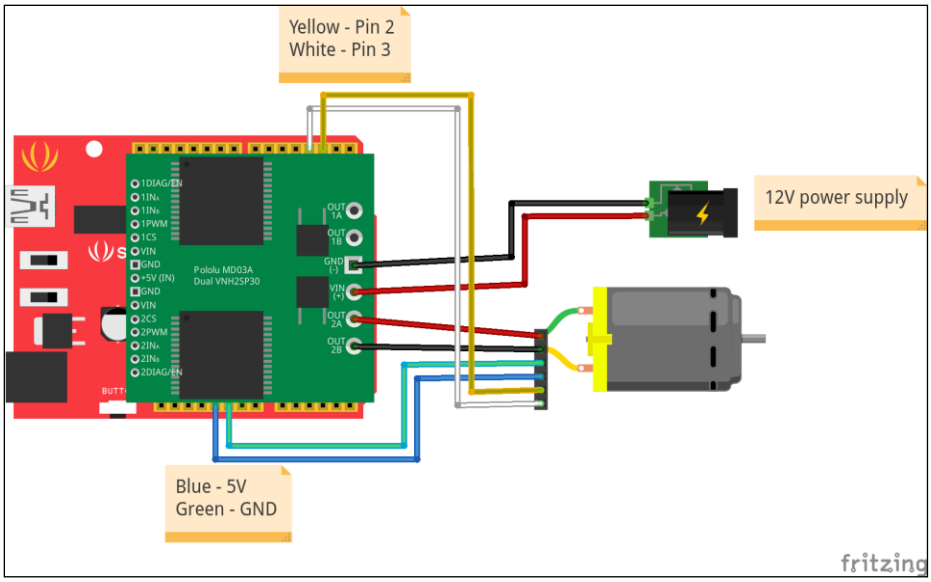

Figure 5. Motor and encoder connections to motor driver
(NOTE: the color of the wires drawn here, between the motor's 6-pin connector and the shield, correspond to the color of wire they should be connected to)

You will wire the motor and its encoder as shown in Figure 5. Make sure that you uploaded the code to the Arduino. If you reset your Arduino the motor should spin for a short amount of time and then stop following the procedure:

1. Turn the motor for a short time (~1 s) attempting to follow ω_d, while collecting data.
2. Motor stops.
3. Arduino prints data collected form Step 1 to Serial Monitor.
4. Waits for the Arduino to restart, or the Serial Monitor window to be opened, then starts over at Step 1.

Practical Exam I:

Collect the data for `Kp` = 1, 2, 3, and 4. Make sure to set `desVel_amplitude` (ω_d) to 60 rad/s. **Hold the body of the motor tightly while collecting data!** Make sure that `constantVelocity` is set to `true`. Plot the four curves on the same plot and show your TA. Be prepared to explain the effect of *Kp* on the steady state error and the time constant. **Save your data for the write-up!**

Part II: Frequency response for the velocity of the DC motor

The goal of this part of the lab is to characterize the frequency response of the system, to show that it resembles the behavior of a low pass filter. In particular, you will explore how well the system tracks the desired input velocity when the input is a sinusoid, across a range of frequencies, following the equation:

$$\omega_d = A * \sin(2\pi * f * t)$$

To characterize the frequency response of the system, follow the steps below.

REMEMBER: DO NOT LET YOUR MOTOR OVERHEAT.

1. Set `Kp` equal to 4;
2. Set `constantVelocity` to `false;`
3. Set `desVel_amplitude (`A`)` to `60;`
4. Set `desVel_frequency` (f) to the desired frequency.

Practical Exam II:

Collect the data for four different frequencies by setting `desVel_frequency (`f`)` = 1, 5, 10, 20, and 40. **Hold the body of the motor tightly while collecting data!** Make sure that `constantVelocity` is set to `false`. Plot the four curves on separate plots and show your TA. Be prepared to explain the effect of frequency on the amplitude and phase shift and explain how you would measure amplitude. **Save your data for the write-up!**

Write-Up

Present the following plots:

1. Motor speed as a function of time for all values of K_p. Also include the desired velocity (part I).
2. Use the control law to calculate the control value (u) that was applied at each data point collected. Plot u as a function of time for each value of K_p in a single figure (part I).
    - Write the control law. Briefly explain how it used the desired and actual velocities.
3. Steady state error as a function of `Kp` (part I).
    - Is this plot roughly consistent with the equation relating steady state error and Kp from lecture? Explain.
4. Time constant as a function of `Kp` (part I).
    - Is this plot roughly consistent with the equation relating the time constant and Kp from lecture? Explain.
    - Hint: It may be easier to measure 5 time constants, which is the time at which signal reaches 99.3% of steady state.
5. Amplitude of output sine wave as a function of the frequency (part II).
    - Explain why this data resembles a low pass filter. What happens to low frequencies? High frequencies?