2.017 Design of Electromechanical Robotic Systems, Fall 2009

Lab 4: Motor Control

Assigned: 10/5/09

1 Overview

So far we have learnt how to use the Arduino to acquire various types of signals from sensors such as the GPS receivers, temperature sensors, potentiometers, photo resistors, push buttons, Reed switches, etc. We will now turn our attention to actuator control, which is a critical part of an electromechanical robotic system. With both sensing and actuator control capabilities, the robotic system can actively interact with the environment that it is in.

The goal of this lab is to learn how to control a DC motor using the Arduino microcontroller board and the Motor Shield. We will use the reading from a rotary encoder attached to the back of the motor as our feedback signal. You will then learn how to interpret the encoder signal, design a controller for the motor, and drive the motor to a set-point or to follow a pre-defined profile. After that we will experiment with an RC servo by commanding the servo to go to a set position.

We will spend the last half hour of the lab for project discussion.

2 DC Motor Experiments

One Maxon F2140.937 DC brushed servo motor, one Arduino motor shield with encoder interface circuit, and one external power supply will be given to each group. Connect up the above components to your Arduino board and PC according to the photo shown in Figure 1.

- Download the file "Lab4files.zip" from http://web.mit.edu/hchin/Public/2.017/ and upzip to your lab4 folder.
- Unzip "ServoTimer1-fixedv13.zip" and "AFMotor_18-2-09.zip".
- Put the above two unzipped folders in your "C:\...\Arduino\hardware\libraries" folder.
- The rest of the files are described below:
    - "DC_Motors_Encoders.pdf" – Reference document on DC motors, PWM, and encoders.
    - "Maxon_motor_specs.pdf" – DC motor, encoder and gearhead data sheets.
    - "94_pc6_datasheet_0.pdf" – Decoder circuit board document.
    - "LS7183_LS7184.pdf" – Decoder IC chip document.
    - "MotorControlEncoderTemplate3.pde" – Arduino template code for DC motor control.
    - "Servo1.pde" – Arduino template code for servo motor control.
- Read the above documents.

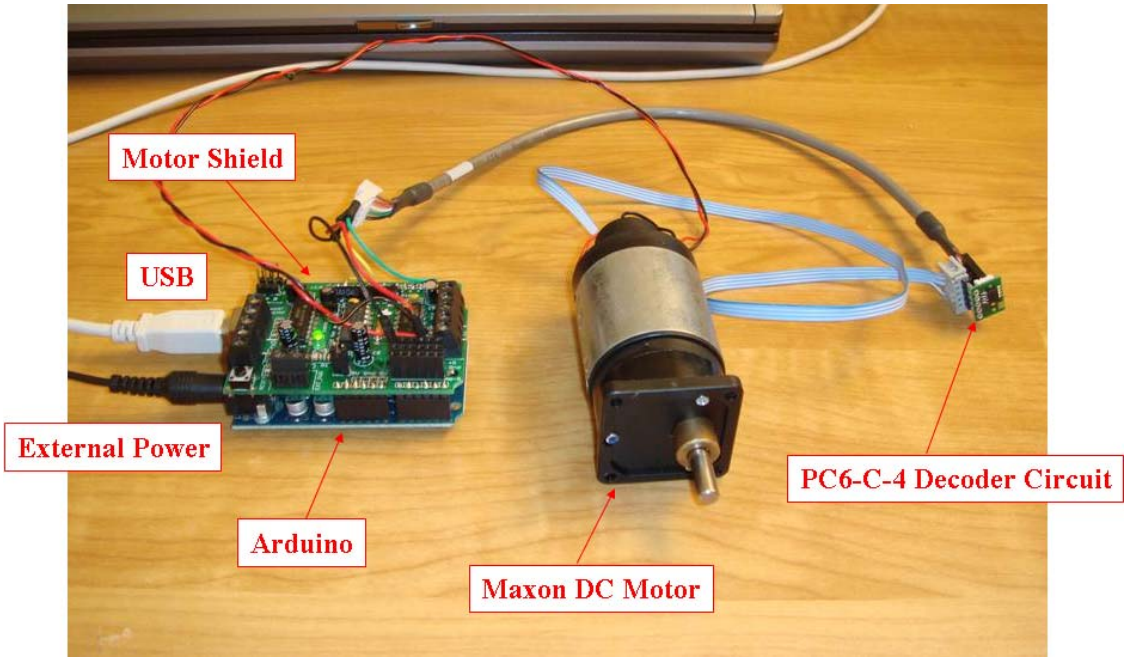

Figure 1. DC motor control lab setup.

2.1 Processing Encoder Signals

The encoder signals are processed by the decoder circuit provided with your motor. It helps count the quadrature outputs and give a decoded signal to the Arduino. We've chosen this setup because the Arduino has only two interrupt pins available, one of which is used by the motor shield. This makes it difficult to keep an accurate record of the two channels of the quadrature input.

The dedicated decoder chip translates the quadrature outputs into a single clock channel and a direction channel (see Figure 2). The clock channel changing status indicates that a single "tick" for the encoder in the direction indicated by the direction output. Now the Arduino needs only interrupt on one signal, read the direction on a standard digital pin instead of an interrupt, and update its count accordingly.

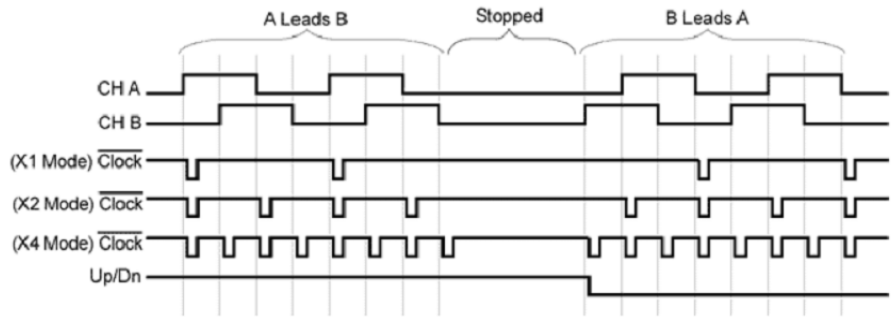

Figure 2. PC6-C-X decoder circuit timing diagram. The circuit we use is set up to be the X4 mode.

- Use an Oscilloscope to measure the raw encoder signal from either encoder channel.
- Measure the clock channel from the decoder circuit.
- How many encoder counts per revolution of the motor shaft? Use the provided spec sheets to find the answer.

For the purposes of this lab, you can simply use the "encoder0Pos" variable to give you the current encoder position and allow the provided interrupt function to keep it updated.

- Capture encoder data by turning the motor by hand:
  - Upload the "MotorControlEncoderTemplate3.pde" sketch to Arduino.
  - The Arduino program continually prints time and the current encoder position. You can capture this data with the "RealTerm" software.
  - Run "RealTerm". In the Display tab, set "Display as" to the second "Ascii", then under the Port tab, set the correct COM Port and the baud rate to match up with the one declared in the Arduino program, then click the change button with the green check mark. You should now be able to see reasonable data coming from the Arduino.
  - You can also send commands to the Arduino if you have set the Arduino program up to receive serial commands.
  - To capture data, go to the capture tab. Specify the output location, then begin capturing data of interest by clicking "Start Overwrite". When finished, click "Stop Capture". Now you have a text file you can import into Matlab and graph. Be sure that the first and last lines are complete to avoid uploading partial data. Delete these lines if necessary.
- Using the captured data, plot encoder position vs. time. Also write code to calculate the derivative of the position, and plot velocity vs. time.
- Find the calibration factor between angles in radians and encoder counts by manually turning the motor. Is the number the same as the one you found based on the information on the spec sheets?

2.2 Implementing Closed-Loop Position Control

Now implement your position controller on the Arduino.

- The command to drive the DC motor in the Arduino code is called "setSpeed()" which can be set between 0 and 255, where 0 corresponds to no voltage, 255 corresponds to full voltage, and the duty cycle (the percent of the time the voltage is set to high) varies linearly in between.
- Start with a proportional controller and add a derivative term to make a PD controller. See how well your controller works by manually changing the set point. You can also try using the provided function generators which give you a slow square wave and a sine wave.
- Capture your controller's performance and make a plot in Matlab once you've found effective gains. A plot of the root locus has been provided in Figure 3 to help you think about your controller design.

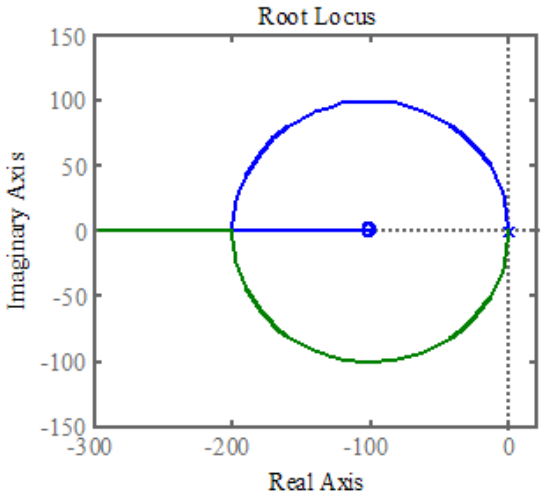

Figure 3. A root locus plot of an open-loop PD controlled motor. Your task is to adjust the controller gains until the response of the closed-loop system is reasonably fast and well damped.

2.3 Higher Performance from the Control System

Those of you who have taken a more advanced class in control systems may wonder why we are using PD control, rather than PID or Lead-Lag/Lag-Lead control. We can, in fact, use PID control, but there are a few real-world considerations that need to be addressed. For extra credit, try to make a PID controller that follows the square and the sine waves provided in the code. Remember that a PID compensator is a filter of the following form:

$$G_c(s) = K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{s}$$

It has two zeros and one pole. For the moment, we will not think about these as separate gains, but as zero locations:

$$G_c(s) = K_o \frac{(s - z_1)(s - z_2)}{s}$$

We will assume that the plant is just two poles at the origin, so the forward transfer function of the system takes the form:

$$G(s) = G_c(s)G_p(s) = K_o \frac{(s - z_1)(s - z_2)}{s^3}$$

Use the root locus tool (i.e., sisotool) in Matlab to pick values of z_1 and z_2 that produce a reasonable root locus for this open-loop transfer function. The performance of the controller will be dominated by the location of the closed-loop pole/pole pair closest to the imaginary axis. Once you have chosen these values, use the equations above to solve for the gain values.

Once you have successfully implemented this PID controller and have it tracking the reference sine wave, connect it up to a square wave reference. You will notice that the controller exhibits a huge amount of overshoot. This phenomenon is not predicted by the linear model of our system. It is due to the fact that the motor response is rate limited.

When a reference is given that the motor cannot follow, the integral error will increase very quickly. Once you are back on the reference, the integral error effectively acts as a disturbance, causing overshoot. Control designers refer to this problem as integrator wind-up. It can be solved either by pre-filtering the reference signal by rate limiting and smoothing so that the controller only receives inputs that it can follow, or by placing saturation limits on the integrator so that it can't wind up too far. Experiment with various ways to reduce wind-up. You may want to try setting saturation limits on the integrator first.

Print out one graph each showing the closed-loop response of the PID controlled system with the square and the sine wave reference input respectively. Is it better than the PD control and why?

2.4 Velocity Control

Some applications require motor speed control instead of controlling its position. If you have time, design a simple PI controller to control the speed of the motor by modifying the template code.

3 Controlling a Servo

RC hobby servos are the easiest way to set up for motor control. They have a 3-pin 0.1" female header connection with +5V, ground and signal inputs. The motor shield simply brings out the 16bit PWM output lines to 2 3-pin headers so that it is easy to plug in a servo and start sending command signals to it.

Typically an RC servo can be positioned from 0 to 180 degrees. Inside the servo there is a DC motor connected to a potentiometer. PWM signals sent to the servo are translated into position commands by the feedback circuitry inside the servo. When the servo is commanded to rotate, the motor is powered until the potentiometer reaches the value corresponding to the commanded position. RC servos are often used in small-scale robotics applications due to their affordability, reliability, and simplicity of control by microprocessors.

The servo has three wires: ground (usually black), power (red) and control (white). However the servo we have in the kit has a different color scheme which is wired as brown (negative), red (positive) and orange (signal).

The servo will move based on the pulses sent over the control wire, which set the angle of the actuator arm. The servo expects a pulse every 20 ms in order to gain correct information about the angle. The width of the servo pulse dictates the range of the servo's angular motion. The PWM pins of the servo connectors on the Arduino motor shield are setup to provide the required duty cycle to drive a typical servo.

- Attach one of the shaft attachments to the servo.
- Connect the servo to “SER1” male connectors on the motor shield. Make sure the brown wire is connected to ground pin.
- Open and upload the “Servo1” sketch.
- Open the “Serial Monitor” and try sending different shaft angles to the servo.
- Modify the code so that you can use a potentiometer to control the shaft angle.

4 Project Discussion

- Proposal feedback.

5 Deliverables

- Answer all the questions above.
- Plots.
- Show the teaching staff your lab notebook.