Introduction to Mechatronics (ME/AE 6705)
Lab Assignment 9

Characterizing a DC Motor Using Encoders, MSPM0 LaunchPad and PWM Signals

9.1 Objective

The objectives of this lab are three fold:

i. to build an electromechanical system using a DC motor and an encoder,

ii. to implement an open loop controller using an MCU, and

iii. to characterize the step response and transfer function using experimental data.

9.2 Deliverables and Grading

To get credit for this lab assignment you must:

1. Demonstrate proper operation of your code to TAs or instructor during office hours or demo hours. You must write your own code for this lab – no lab groups are allowed. (20 points)

2. Submit all plots and answer all questions listed at the end of the lab assignment as a PDF file to Canvas before the due data specified on Canvas. (40 points).

3. Submit the commented final version of your code on Canvas.) (Pass/Fail)

9.3 Setup

This lab requires Code Composer Studio, the MSPM0 LaunchPad, and an electromechanical system comprised of a DC motor and an encoder.

The lab uses onboard features of the MSPM0 LaunchPad including GPIOs, TIMG, PWM, and Timer Capture/Compare and interrupts. We will also use the motor driver circuit built in Lab 8.

9.4 Problem Statement

Create an electromechanical system consisting of a DC motor and an encoder sensor attached to the motor. The microcontroller will be used to apply a variable PWM signal to the motor driver circuit built in Lab 8.

This lab will characterize a fundamental property of the dynamic system – the system's *step response*. The first-order transfer function of the system can be modeled by extracting the system's *time constant* and *gain* from the step response.

A series of experiments will be performed in this lab. First, a step input will be applied by taking speed readings with a duty cycle of 100%. Speed readings will be taken in real-time until the speed stops changing (less than 2 seconds). The step response can be shown by plotting this time series, and a first-order transfer function for the DC motor can be derived based on that.

For the next experiments, the same process will be performed for all duty cycles (20-100%). The plots of step responses for different duty cycles can then be constructed.

9.5 Hardware

We reviewed the background on brushed DC motors, H-bridge ICs, flyback diodes and breadboards in Lab Assignment 8. In this lab, we will be also using the encoder attached to the DC motor.

9.5.1 Encoder

An encoder is a sensor with digital outputs that relate to rotational speed. The DC motor used in this lab is equipped with an encoder board that senses the rotation of the magnetic disc and provides a resolution of 12 counts per revolution of the motor shaft. The encoder board has 6 pins: the motor leads are connected to the + and − pins along the edge of the board; the remaining four pins are used to power the sensors and access the two quadrature outputs.

The encoder is powered through the VCC and GND pins. To be used with the MSPM0 LaunchPad, VCC should be 3.3 V, and the quadrature outputs A and B are open-drain digital signals that need to be pulled up to the 3.3V voltage of the system using two 10k resistors. **The motor spec sheet says the encoder must be powered with 4.5-7.5 V, although it may work fine with a 3.3 V supply voltage. You can try 3.3 V first and if that is giving you problems, you can use another voltage source to power the encoder at 5 V instead.**

The encoder produces a pair of pulse trains, called outputs A and B, that look like the signals shown in Figure 2. Each output pulses 960 times per rotation. If we measure the Period (in sec) of one of the encoder signals, we can calculate the motor speed in rpm as

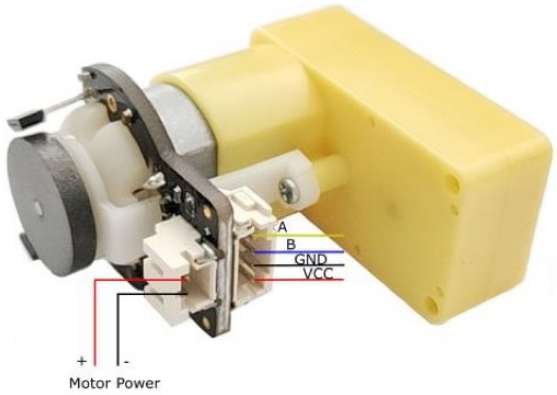

Figure 1: Motor with encoder.

$$Speed\ (rpm) = 60/(960 * Period)$$

We can use the MSPM0 *input capture mode* to make time measurements on input signals. The MSPM0 microcontroller has 14 General Purpose Timer Modules called `TIMG`. Each timer has one 16-bit timer and two associated capture/compare registers.

In this lab, we will use `TIMG6` to create a capture input for the encoder interface. There will be an interrupt on each falling edge of output A, and the timer will measure the period of the encoder output.

The second encoder signal (output B) can be connected to a GPIO pin to detect the motor direction: A falling edge for output A and a high signal for B signifies one direction, while a falling edge for output A and a low signal for B signifies the other direction. The direction associated with each case can be tested by running the motor and visually noting the direction of rotation for the tested case.

9.5.2 Circuit Schematic

Figure 3 shows the circuit we use to interface the brushed DC motor and the encoder with the MSPM0 LaunchPad. You have already assembled part of this circuit (everything except the encoder) on the breadboard in Lab Assignment 8. **The schematic shows the MSP432 board but it is similar to the MSPM0 board, just with different pin names.**

Two $10K\Omega$ pull-up resistors [1] are connected to the encoder output pins (pin A and pin B), as the outputs are open-drain digital signals.

The encoder outputs of the DC motor are connected to `PB6` for the capture input, and a GPIO pin of your choice (shown in the figure as `P1.7`). Note that `PB2` corresponds to `TIMG6_C0` which can be used as Capture/Compare Input 0 on TIMG instance 6 (`TIMG6`, that is Register 0).

[1] Resistor color codes: brown-black-orange ($10K\Omega$)

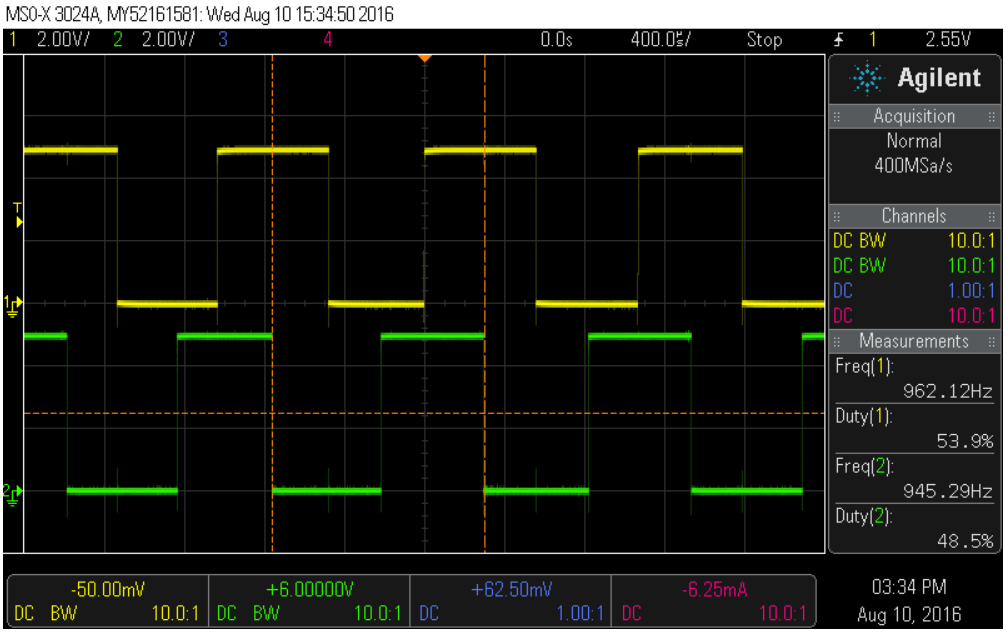

Figure 2: Scope trace of the two outputs of the encoder, period*960 is the time for one revolution (Image courtesy of Pololu). The two vertical dashed lines indicate a period of the green signal (the time between two falling edges).

Note: Part of the circuit shown in Figure 3 is the same as the motor driver circuit built in Lab 8.

Note: Remember to connect a common ground between the LaunchPad and the driver circuit.

Warning: While driving the motors using the H-bridge circuit, continue to check the temperature of the MSPM0 and SN754410 ICs. If it is getting hot, disconnect the LaunchPad immediately and debug the circuit.

9.6 Electromechanical System Modeling

As discussed in the class, the open-loop transfer function of a DC motor plant is given by:

$$P(s) = \frac{\omega(s)}{U(s)} = \frac{K}{(Js+b)(Ls+R)+K^2} \qquad ({}^{rad/sec}/_{V}) \tag{1}$$

Often times, the motor inductance can be neglected (assuming L is too small), and the motor transfer function is written as a first order transfer function.

The plant *input* is the voltage (or the duty cycle of the PWM signal) that drives the motor, and the plant output is the motor speed (which can be measured as the period of the encoder output signal).

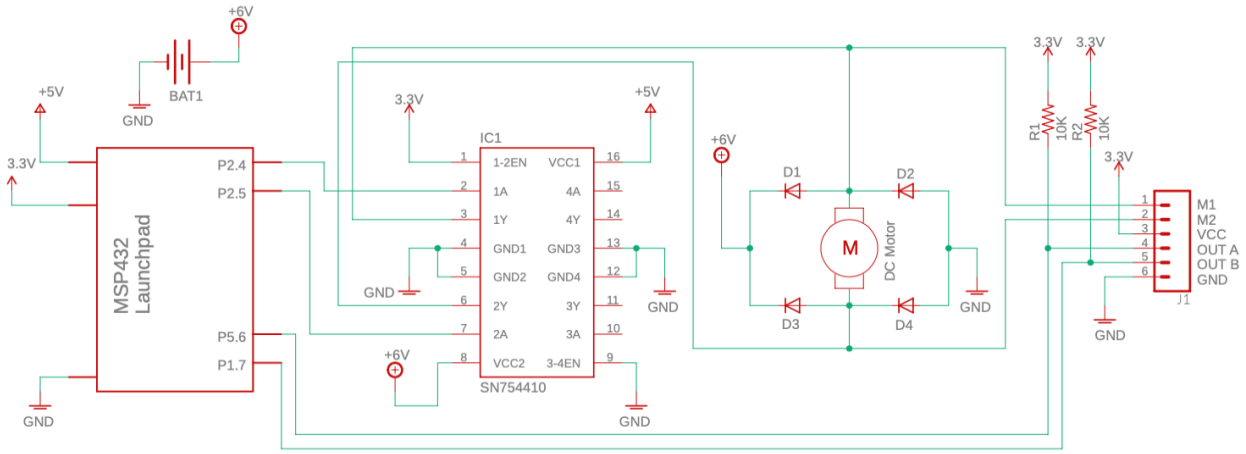

Figure 3: Circuit diagram for the H-bridge interface with motor and encoder. **The MSP432 is replaced with the MSPM0 launchpad with associated changes in the pin names.**

For a motor with unknown parameters, the transfer function can be obtained by either a systematic or an experimental approach.

9.6.1 Systematic approach

With this approach, the motor parameters including inertia J, damping b, inductance L, resistance R and motor torque constant K need to be measured or calculated.

Given the fact that the torque produced by the rotor (τ_m) is directly proportional to the current (i) in the armature windings through the torque constant of the motor (K):

$$\tau_m = K \times i, \tag{2}$$

this approach requires advanced lab equipment for torque measurements. Therefore, for the purpose of this lab assignment, we will use an experimental approach to estimate the transfer function of the DC motor.

9.6.2 Experimental approach

In this lab, the transfer function will be *estimated* by extracting key parameters from a measured *step response*. The plant input will be a *step function* (a change in duty cycle of the PWM signal). The step change in duty cycle is produced by the MSPM0 LaunchPad. The resulting response at the plant output is also recorded by the microcontroller and stored in an array of its internal memory, which can be examined by the debugger.

Suppose the plant's step response looks like a simple exponential, as illustrated in Figure 4. This suggests that the plant can be modeled as a *first-order* system with input $u(t)$, measured output signal $y(t)$, time constant τ, and gain K. The model is given by the linear ordinary differential equation:

$$Ku(t) = \tau \frac{dy(t)}{dt} + y(t)$$

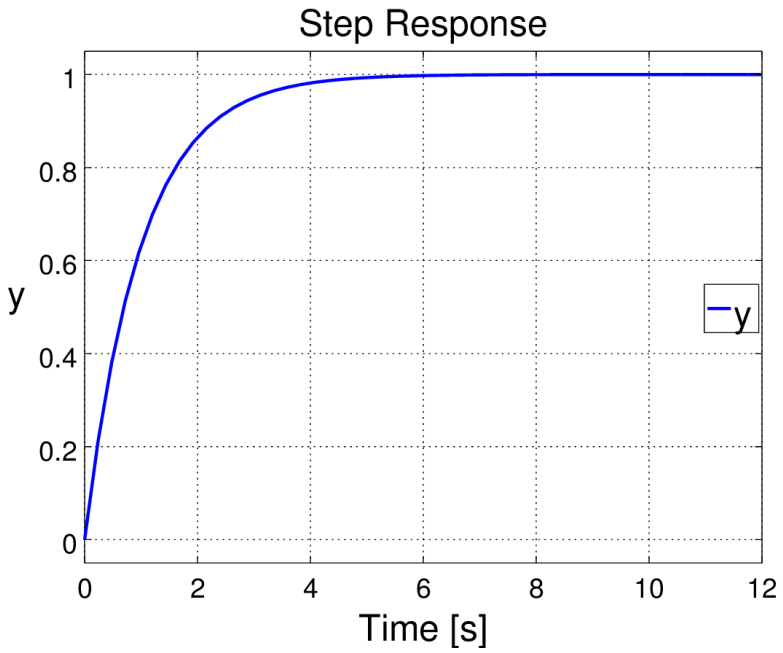

Figure 4: First order step response.

Solving this equation produces the step response for time $t > 0$:

$$y(t) = K\Delta u(1 - e^{-\frac{t}{\tau}})$$

where Δu is the step change in the input. The gain K is simply the change in y divided by the change in u, computed when the system has reached steady state (i.e. for large t).

The time constant τ is the time needed for the output to make 63.2% (e^{-1}) of the change from its initial to its final value.

Knowing the time constant and gain, the transfer function of this first-order system can be written as:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{K}{\tau s + 1} \tag{3}$$

MATLAB/Simulink can be used to verify the experimental data via the model, which is that of a first order system driven by a step input. If the experimental data fits a first order behavior, then using the computed values of K and τ in the Simulink model should produce a simulated response that matches the experimental result.

9.7 Software

Note: To create a new CCS project for the MSPM0 target device, please use the steps followed in previous labs.

The first task for the software is to use `TIMG6` to measure period from the encoder. Set up `TIMG6` to create a capture input for the encoder interface. There needs to be an interrupt on

each *falling edge* of output A (connected to PB2), and the encoder output can be measured by comparing the "capture/compare count" from the previous falling edge to the current falling edge.

Note: Once the `TIMG` Capture mode is initialized for the Register 0 and Instance 6, the `TIMG6_INT_IRQn` timer interrupt needs to be enabled, which results in triggering the `TIMG6_IRQHandler()` interrupt service routine at each timer interrupt.

Note: In addition to the "Capture Mode", `TIMG6` needs to be configured to run with a certain clock frequency. Source `TIMG6` from the BUSCLOCK and use a clock divider of 8 so that the timer frequency is 4 MHz.

Note: The timer capture/compare count can be accessed using the DriverLib function: `DL_Timer_getCaptureCompareValue(...)`. Please consult the DriverLib User's Guide for more details on the associated input and output arguments.

An example input capture code has been uploaded to Canvas for you to see how to configure the input capture for this lab.

The second task is to use the encoder period to determine the motor speed. Since there are 960 pulses per rotation, the Speed in rpm can be calculated as

$$Speed\ (rpm) = 60/(960 * Period)$$

If you choose the timer clock as the BUSCLOCK with a clock divider of 8, this results in a 4 MHz timer clock rate, and thus the period measurement resolution will be 250 ns (this is the time it takes for one timer increment). The second input of the encoder can be used to determine which direction the motor is spinning (optional). You will write software that counts the number of timer ticks between two consecutive encoder pulses as the motor spins, and store them in an array, as described in the following.

Write a program that starts with applying a PWM signal with zero duty cycle to the motor. The PWM signal can be generated using `TIMG12`, similar to Lab 8. Pressing the S1 pushbutton on the LaunchPad should adjust a desired speed variable to different PWM duty cycles (in 20% increments) (while maintaining the motor off). Pressing the S2 pushbutton should then instantaneously drive the motor with the desired speed (according to the specified PWM duty cycle), thus producing the required step input.

As soon as the step input is produced, the program should log the motor speed by measuring the period of encoder pulses. This can be done by storing the *timer tick*s at each timer/-capture interrupt caused by the encoder output A into a buffer. Record data until speed is stabilized (for several seconds). This represents the step response of the system. After the motor has reached steady state, convert the stored timer ticks to *speed values* (in RPM) and *timestamps* and print out the array of sampled speed values along with their timestamps into the CCS console. This allows one to plot the step response ("speed" vs. time). The motor time constant and gain parameters can then be extracted from the plot.

Apply different control inputs and record the resulting step responses and steady state speeds. Start at 20% duty cycle. Record encoder measurements until they stop changing, and print out the data into the CCS console. Repeat these steps for duty cycles 40-100%, in 20% increments. Compute the average steady state speed for each duty cycle applied.

9.8 Questions

1. Generate a step input with 100% duty cycle and measure the open-loop motor response by capturing sampled motor speed values and storing it in an array. Print out this array to the CCS console after the motor reaches steady state. Repeat this experiment for different speed changes (i.e. 20%, 40%, 60% and 80% duty cycles) so that the results can be compared.

2. Use a program such as MATLAB, Excel, etc. to plot the motor responses for each tested case versus time. Use the response corresponding to 100% duty cycle to measure the motor time constant and gain K. Write the open-loop transfer function for this case. This transfer function will describe the input-output relationship between the motor speed and input duty cycle. In addition, compare the step responses for each test case. What do you notice happening as the duty cycle increases? What is the reason for this?

3. Verify the measured time constant and gain by creating and running a Simulink model for a first order system. Use this to simulate a step input. Your simulated response should be similar to your experimental results.

4. Calculate the transfer function using step responses resulted from other duty cycles. Is the resulting transfer function the same for all the experiments? If not, discuss why.

Make sure to cover each of the questions above fully, as points will be deducted for incomplete answers.