SE420 Laboratory Assignment 5
DC Motor Discrete Transfer Function Identification

Goals for this Lab Assignment:

Use a least-squares solution to identify the discrete open-loop transfer function of the DC motor with added flywheel.

Library Functions Used:

saveData, readEnc1, setEPWM3A

Matlab Functions Used:

serialreadSPIRAM

Prelab:

In MATLAB, solve the following over-determined set of linear equations for a,b,c,d using a least squares solution.

```
12    +     a*32  = b*45      + c*12      + d*2
44    +     a*100 = b*13      + c*17      + d*21
2     +     a*16  = b*19      + c*11      + d*43
32    +     a*111 = b*112     + c*33      + d*23
112   +     a*82  = b*54      + c*12      + d
84    +     a*29  = b*101     + c*88      + d*73
9     +     a*24  = b*92      + c*72      + d*81
14    +     a*234 = b*87      + c*37      + d*63
35    +     a*11  = b*39      + c*19      + d*53
```

Type "help slash" to see how to solve an over-determined set of equations in MATLAB. Show that the answer is a= -0.1111, b = 0.2603, c = 0.8810, and d = -0.6174 by printing the commands and their output in MATLAB. Note, these numbers and equations were just pulled out of the air as an exercise to familiarize you with a least-squares solution.

Laboratory Exercise

In this lab we are going to introduce you to the process of system identification, that is, to the task of identifying the *parameters* of an unknown plant of assumed dynamical structure. We will use the DC motor with attached flywheel as our plant to identify. Below is the open loop block diagram of the system. Figure 1 shows the full block diagram of the system. The DC motor only has angular feedback (optical encoder) so you will approximate its velocity with the backwards difference rule. Figure 2 shows a simplified block diagram of the system. Here we are assuming that our velocity approximation is exact and cancels the integrator in Figure 1's DC motor position transfer function. This will help simplify the identification but still allow you to identify K and τ_m. In general, the task of identification consists of applying carefully chosen inputs and measuring the plant output(s). In our case, the model is very simple and we will find that a step input works very well.

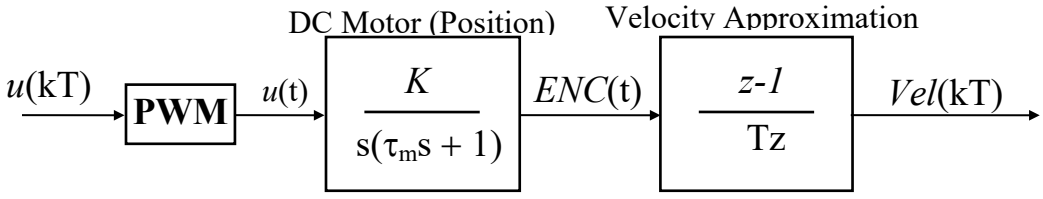

Figure 1: Full Open-Loop Block Diagram

DC Motor (Velocity)

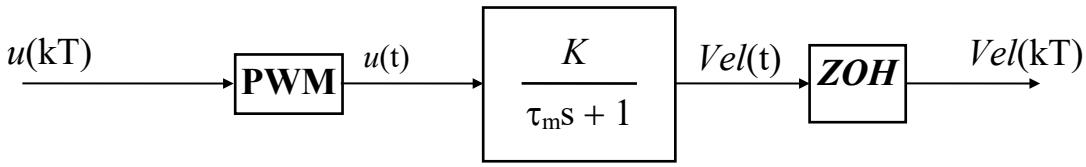

Figure 2: Simplified Open-Loop Block Diagram

Using Figure 2 as a guide, the discrete transfer function can be derived by assuming a *zero-order hold* (ZOH) and taking the z-transform of the continuous system:

$$\frac{Vel(z)}{U(z)} = (1 - z^{-1})\mathbf{Z}\left\{\frac{1}{s}\frac{Vel(s)}{U(s)}\right\} \tag{5.1}$$

Then

$$\frac{Vel(z)}{U(z)} = \frac{K[1 - e^{-T/\tau_m}]}{z - e^{-T/\tau_m}}, \tag{5.2}$$

and corresponding difference equation is

$$Vel(\mathrm{kT}) = c_1 \cdot Vel(\mathrm{kT} - \mathrm{T}) + c_2 \cdot u(\mathrm{kT} - \mathrm{T}) \tag{5.3}$$

where:

$$c_1 = e^{-T/\tau_m},\ c_2 = K[1 - e^{-T/\tau_m}]. \tag{5.4}$$

Our goal then is to identify the two model parameters, c_1 and c_2. To do this, you will apply an open-loop step input to the DC motor and store the motor's velocity response to an array. In MATLAB, you will use this response data with equation 5.3 to form an over-determined set of equations that are a function of c_1 and c_2. Using least-squares regression, you will then solve for c_1 and c_2.

Procedure:

1. Create a new DSP application, LABstarter, that outputs a constant step value, u = 5, to the DC motor. This application should, in cpu_timer0_isr, sample the optical encoder of the DC motor and use the radian values to estimate the speed of the DC motor in units of radians/second. For at least the first 2 seconds of your run, store time in seconds, the constant output value and velocity value using the "saveData" function. In Lab 3, you performed very similar tasks as you will need here for Lab 5 so make sure to copy some of your code from Lab 3, but keep in mind that we would like to start out using a sample period of 0.001 seconds and Lab 3 had you use a 0.005 second sample rate. Like in Lab 3, call the "saveData" function every sample period saving time in seconds, velocity in radians/sec and u (which is proportional to torque). We will be using 3 different sample rates for the first part of this lab exercise. The length of data points collected will need to be changed as you change sample rates for the different identification runs. Change the #define NUM_POINTS , in SpiRAM.h, to adjust the number of points so that at least two seconds are recorded. (NUM_POINTS must be divisible by 200 so when you get to the 15ms sample rate you will have to store more than two seconds of data.)

One more important issue in regards to collecting the required data, the motor needs to start from rest and then be driven with the constant input. When you are programming your code to the Launchpad, it is possible for the motor to move and stay moving. So normally you would just move the slide switch to disable the motor. Our issue is that we need to record the first two seconds and have the motor run immediately from rest when you run your code. So an easy way to solve this problem is to add a 2 second delay in your main() function giving you time to switch on the motor before the motor is driven with the constant input. A good place to put this delay is right before interrupts are enabled with the EINT and ERTM commands (around line 230). See below:

```
DELAY_US(2000000); // Two second Delay
// Enable global Interrupts and higher priority real-time debug events
EINT;  // Enable Global interrupt INTM
ERTM;  // Enable Global realtime interrupt DBGM
```

2. In MATLAB, use the function **serialreadSPIRAM** to up load your data to MATLAB's workspace after the blue LED has turned on.
3. Plot the motor's response. From the plot find the value for K (remember you are applying a step of 5). Also using the 63% rule, find an approximate value for τ_m. Record these values in the table below.
4. Using equation 5.3, starting with k = 2 and ending with k = 2000, compile the data into 1999 equations that are a function of c_1 and c_2. i.e.:

$$\begin{aligned} y(2) &= c_1 \cdot y(1) + c_2 \cdot u(1) \\ y(3) &= c_1 \cdot y(2) + c_2 \cdot u(2) \\ &\vdots \\ y(n) &= c_1 \cdot y(n-1) + c_2 \cdot u(n-1) \end{aligned} \quad ,$$

Solve for c_1 and c_2 by least-squares regression. Then to check if you found correct values for c_1 and c_2, solve for K and τ_m using the equations for c_1 and c_2. K and τ_m should be close to the K and τ_m found from the plot. Record these values in the table below with 6 decimal places of precision for c_1 and c_2.

5. Repeat these steps for a sample period of 5ms and then a sample period of 15ms. Modify the length of your data collection so that at least 2 seconds of data is collected. (NUM_POINTS must be divisible by 200) Also note that in exercise 2 of the Lab Check Off items you will be asked to use the same data you collected for the 5ms sample rate. So as a minimum, make sure to save your 5ms data set.

U	Ts	c1	c2	τ_m calculated from c1 and c2	K calculated from c1 and c2	τ_m From Plot	K From Plot
5 Unit Step	1 ms						
5 Unit Step	5 ms						
5 Unit Step	15 ms						

Lab Check Off:

1. Complete the table above. Do K and τ_m agree with the different identification methods (and with your expectations)?

2. Now assume that the motor transfer function is third-order and has the form:

$$\frac{Y(z)}{U(z)} = \frac{a_1 \cdot z^2 + a_2 \cdot z + a_3}{z^3 + a_4 \cdot z^2 + a_5 \cdot z + a_6}, \tag{5.5}$$

Pick your data run that used a 5 ms sample period and use it to identify the parameters a_1 - a_6. The procedure will be very similar to your previous identification. You will just need to modify the A and B matrices for the least squares solution. How does this higher order model compare to the lower order model? Answer this question by producing step responses and bode plots of both your original low order identified transfer function and this new higher order transfer function. Also compare the poles of each transfer function. Can you explain the faster set of poles found in the higher order model? Hint: Look at the plot of the velocity data uploaded for the 5ms run. Show these plots and the M-file used to run this identification to your TA.

3. For this question, I would like you to investigate the numerical precision needed for z domain transfer functions as sample rate is increased. To do this use a hypothetical motor with K = 60 and τ_m = .3. Use the steps below to help you in this task.
    a. Use the MATLAB script below.
    b. Form the continuous transfer function in Matlab. Note where the continuous pole is located and compare it to the poles found after a numerical error has been introduced to the discrete plant model.
    c. Find the discrete transfer function using 0.01 seconds as the sample period.
    d. Add a numerical error of 0.009 to the discrete pole. Think as if this error was caused but human error typing in a slightly wrong coefficient. Convert the discrete transfer function back to a continuous transfer function and compare this new continuous transfer function's pole to the original pole of –(1/.3), -3.3333. (Use the following Matlab code:)

```
%EXAMPLE m-FILE
%Set up transfer function
num=[60];
den=[.3 1];
sys=tf(num,den)
%Now use c2d to map your transfer function to the z-domain
sysz1=c2d(sys,.01,'zoh')
%Now add a plant numerical error by adding .009 to your discrete model.
sysz1.den{1} = sysz1.den{1} + [0 0.009]
%Now use d2c to map your transfer function back to the s-domain
sysc1=d2c(sysz1)
```

    e. Repeat the same process for a sample period of 0.001 (1000Hz) seconds then 0.0001 (10000Hz) and finally 0.00001 (100000Hz). What is happening as your change to a faster and faster sample period? If you ran a discrete simulation with the discrete transfer function that had the added 0.009, would you be simulating the correct system?

f. What can you say about numerical error in the transfer function's coefficients at different sample rates? NEVER do (what ???) with your transfer function coefficients when coding in C or even working with the transfer function in Simulink? This is why I love the "LTI System" block in Simulink instead of the "Discrete Transfer Function" block. Make sure to remember this when working on your future labs.