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:

readEnc1, setEPWM6B

Matlab Functions Used:

SE420_serialread

Prelab:

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

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

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](image-url)
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:

\[
\frac{Vel(z)}{U(z)} = (1 - z^{-1}) \mathcal{Z} \left\{ \frac{1}{s} \frac{Vel(s)}{U(s)} \right\}
\] (6.1)

Then

\[
\frac{Vel(z)}{U(z)} = \frac{K[1 - e^{-T/\tau_m}]}{z - e^{-T/\tau_m}},
\] (6.2)

and corresponding difference equation is

\[
Vel(kT) = c_1 \cdot Vel(kT - T) + c_2 \cdot u(kT - T)
\] (6.3)

where:

\[
c_1 = e^{-T/\tau_m}, \quad c_2 = K[1 - e^{-T/\tau_m}].
\] (6.4)

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 6.3 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 that outputs a constant step value to the DC motor. This application should also 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 the first 2 seconds of your run, store both the output value and velocity value in separate arrays. The size of these arrays will need to be changed as you change sample rates for the different identification runs. Note: The Matlab function ‘SE420_serialread’ and the DSP given support software, only allows 1000 floating-point values to be uploaded in a single call to SE420_serialread. For that reason you will need two 1000 element arrays to store all the data for a 2 second run when sampling at 1ms. This will not be an issue when you are sampling at 5ms and 15ms.
Recall from Lab 3 how to declare variables and arrays you want to have access to in Matlab, e.g.:

```matlab
#pragma DATA_SECTION(amp, ".my_vars")
float amp = 1;
```

```matlab
#pragma DATA_SECTION(testarray, ".my_arss")
float testarray[100];
```

2. In MATLAB, use the function `SE420_serialread` (remember ‘help SE420_serialread’) to upload your data to MATLAB’s workspace.

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 \( \tau_m \). Record these values in the table below.

4. Using equation 6.3, 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.:

\[
y(2) = c_1 \cdot y(1) + c_2 \cdot u(1) \\
y(3) = c_1 \cdot y(2) + c_2 \cdot u(2) \\
\vdots \\
y(n) = c_1 \cdot y(n-1) + c_2 \cdot u(n-1)
\]

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 \( \tau_m \) using the equations for \( c_1 \) and \( c_2 \). K and \( \tau_m \) should be close to the K and \( \tau_m \) 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 arrays saving your data so that only 2 seconds of data is collected regardless of the sample period.

<table>
<thead>
<tr>
<th>U</th>
<th>Ts</th>
<th>c1</th>
<th>c2</th>
<th>( \tau_m ) calculated from c1 and c2</th>
<th>K calculated from c1 and c2</th>
<th>( \tau_m ) From Plot</th>
<th>K From Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 Unit Step</td>
<td>1 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 Unit Step</td>
<td>5 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 Unit Step</td>
<td>15 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Lab Check Off:

1. Complete the table above. Do K and \( \tau_m \) 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:

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

(6.2)
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 $\tau_m = .3$. 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 $-(1/0.3) = -3.3333$.
       (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
       syscl=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.