ME3534 - Lab 3 Report

Name: Zachary Rohloff

Date: 3/29/2026

Lab Bench # 9:

This LiveScript documents the results of the pre-lab and post-lab processing for ME3534 Lab 3. A PDF version of this document will serve as your Lab 3 report. After you have collected data in the Lab, you will then add your own Matlab code to process this data outside of the lab. Finally, you must convert this LiveScript to a PDF for submission to Canvas.

Task 1: Develop a Transfer Function Model for a DC Servo-Motor (Pre-Lab)

Derive a transfer function model of the open-loop DC Servo Motor: $P_{\text{motor}}(s) = \frac{V_{\text{fbk}}(s)}{V_{\text{cmd}}(s)}$. Note that the units of this open-loop transfer function are $\left(\frac{\text{Volts}}{\text{Volt}}\right)$.

You must use the equation editor that is integrated in the Matlab LiveScript interface to write your equations and **SHOW YOUR WORK**! You are also welcome to use the Symbolic toolbox to derive the transfer function; however, the use of that toolbox was not explained in the Lab notes and may not have been explalined in the main lecture. It is simply an alternative option that you could use if you are already familiar with it.

For this task do NOT include numerical parameter values in your derivation!!!!! This is a symbolic (i.e. formula) result.

You must **display** the final numerator polynomial and the denominator polynomial for the $P_{\text{motor}}(s)$ transfer function.

Work to arrive at the below equation is attached at the bottom of the PDF file as I did it by hand to arrive here (that made more sense for showing my work). If that is unacceptable please advise for future reports upon grading, thanks. -Zach

$$G(s) = \frac{\Theta(s)}{V_{cmd}(s)} = \frac{K_a K_t}{RJs^2 + (Rb + K_t K_e)s}$$

Task 2: Define a numerical LTI object for the Motor Model (Pre-Lab)

For this task, you will first define Matlab variables for each parameter and assign numerical values, then construct numerical representations of the numerator and denominator polynmomials, and finally generate an LTI transfer function object representing the motor model. Make sure you display the final LTI object as a way to document your result! Also remember to save your motor model to a .mat file because you will need this for Lab 4.

You should have discovered from Task 1 that the gain product $G = (G_{\text{cmd}}G_{\text{fbk}})$ shows up in the numerator, and that neither of these gain values were provided. After you have collected data in the lab you will need to estimate a value for this unknown gain. Initially, you can assume that $G = 1$; however, remember that this is not the correct value and you will need to re-run this LiveScript with a more accurate estimate. This is an iterative process once you have experimental data!

```
%Given Constants
R  = 2.6;
J  = 2.0e-3;
b  = 4.0e-3;
Ke = 0.00767;
Kt = 0.00767;
%Gain product found via itteration of below code in part 5
%Could not get std error below .1212 .0629 gave me .1212
gain_product = .0629;
%Create motor system
num = gain_product;
den = [R*J, (R*b + Kt*Ke), 0];
motor_sys = tf(num, den);
%Display
disp('Motor Transfer Function:');
```

```
Motor Transfer Function:
```

```
disp(motor_sys);
```

```
  tf with properties:

      Numerator: {[0 0 0.0629]}
    Denominator: {[0.0052 0.0105 0]}
       Variable: 's'
        IODelay: 0
     InputDelay: 0
    OutputDelay: 0
      InputName: {''}
      InputUnit: {''}
     InputGroup: [1×1 struct]
     OutputName: {''}
     OutputUnit: {''}
    OutputGroup: [1×1 struct]
          Notes: [0×1 string]
       UserData: []
           Name: ''
             Ts: 0
       TimeUnit: 'seconds'
   SamplingGrid: [1×1 struct]
```

```
%Save
save('motor_model.mat', 'motor_sys');
```

Task 3: Collect System Identification Data in the Lab (In-Lab)

See lab3_instructions.pdf for details.

There is nothing to include here for the submitted report.

Task 4: Load the Experimental System ID Data and Pre-process it (Post-Lab)

This task must be completed outside of the lab. For this task you must first load the saved .mat file from your experimental system ID test. You will only need the time (`t`), command (`cmd`), and the feedback (`fbk`) signals for this task.

```
load('Controls lab 3 data.mat');
cmd = cmd(:);
t   = t(:);
fbk = fbk(:);
```

Now that you have loaded the experimental system ID data into the LiveScript, you will need to pre-process the `fbk` signal to remove any effects of drift. Normally, this step is not necessary; however, performing this step may significantly improve your results in the validation tasks. The Matlab code to pre-process the `fbk` signal is provided for you, so there is nothing extra you need to do except run the provided code using your data.

First generate the coefficients of a 2nd order Butterworth high-pass digital filter.

```
order = 2;  % filter order
fbreak = 0.1;  % high-pass break frequency [Hz]
fnyquist = 1000/2; % Nyquist frequency [Hz]

[numhpf,denhpf] = butter(order,fbreak/fnyquist,'high');
```

Apply the high-pass filter to the `fbk` response data to remove any low-frequency component of the signal such as "drift".

```
fbk = filtfilt(numhpf,denhpf,fbk);  % overwrite the original data
```

Task 5: Time-Domain Validation (Post-Lab)

In Task 2, you developed a numerical open-loop model (LTI object) of the motor transfer function. In Task 3 you applied a random band-limited excitation to the experimental servo-motor and measured the position response. In this task, you will apply the exact same experimental excitation from Task 3 to the motor model from Task 2 to obtain a simulated position response. If your model is correct, then the experimental response should approximately match the simulated response.

You must use the `LSIM()` function in Matlab to apply the experimental `cmd` signal to your motor model. It may be helpful to apply the same filter (`filtfilt`) from above to the simulated response.

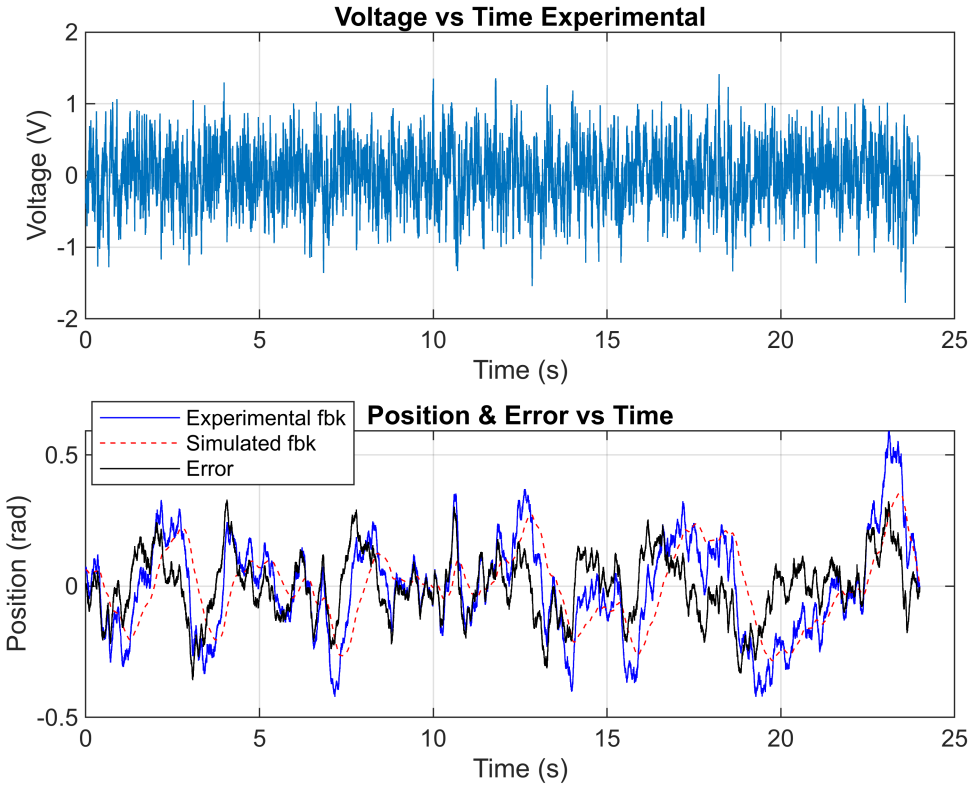
```
t_1 = linspace(t(1), t(end), length(t))';
sim_fbk = lsim(motor_sys, cmd, t_1);
sim_fbk_filt = filtfilt(numhpf, denhpf, sim_fbk);
%Find error
error_sig = fbk - sim_fbk_filt;
```

To document the time-domain validation, you must generate a figure with two subplots. The first subplot will show the experimental excitation vs. time. The second subplot will show three signals: 1) the experimental response (`fbk`) vs time, 2) the simulated response vs time, and 3) the error between the experimental and simulated responses. All plots must be properly annotated with grid lines, axis labels, signal units, and legends.

After generating this figure, you should immediately see that your error is too large because your assumed gain from Task 2 is incorrect. You now need to use a guess-and-check approach to find a gain that reduces the time-domain error. If you would like a specific target error, you should be able to achieve a standard deviation of the error signal that is less than 0.1. Remember that when you change the gain in Task 2, you will need to rerun the LiveScript to see the effect. You will have another opportunity to adjust the estimated gain in the next task as well; although, **you can only use one final gain** for both validation tasks.

```
%Plot 1
subplot(2,1,1);
plot(t, cmd);
title('Voltage vs Time Experimental');
xlabel('Time (s)');
ylabel('Voltage (V)');
grid on;
%Plot 2
subplot(2,1,2);
plot(t, fbk, 'b');
hold on;
plot(t, sim_fbk_filt, 'r--');
plot(t, error_sig, 'k');
title('Position & Error vs Time');
xlabel('Time (s)');
ylabel('Position (rad)');
legend('Experimental fbk', 'Simulated fbk', 'Error');
grid on;
hold off;

subplot(2,1,2)
legend(["Experimental fbk", "Simulated fbk", "Error"], "Position", [0.1345 0.3874
0.2357, 0.0976])
```




```
%Display Error
std_error = std(error_sig);
fprintf('Standard Deviation of Error: %.4f\n', std_error);
```

```
Standard Deviation of Error: 0.1212
```

Task 6: Frequency Domain Validation (Post-Lab)

In Task 5, you validated your motor model in the time domain using experimental system ID data with a time-domain simulation. In this task, you will validate your numerical motor model in the frequency domain using the same experimental system ID data. You will need to generate and compare the experimental frequency response to the numerical frequency response of the model.

For the experimental frequency response estimation, you will use the `TFESTIMATE()` function with `NFFT=2^13`, `NOVERLAP=NFFT/2`, `WINDOW=NFFT`, and `FS=1000`. Make sure you save the frequency vector output from `TFESTIMATE` with units of Hz for plotting. You will also need to use the `MSCOHERE()` function to estimate the coherence between the command and feedback signals using the same configuration parameters that you used for `TFESTIMATE`. See the lab notes for details.

```
NFFT = 2^13;
NOVERLAP = NFFT / 2;
WINDOW = NFFT;
FS = 1000;
%Freq response
[Txy, F] = tfestimate(cmd, fbk, WINDOW, NOVERLAP, NFFT, FS);
```

```
%coherense
[Cxy, F_coh] = mscohere(cmd, fbk, WINDOW, NOVERLAP, NFFT, FS);
```

Next, use the FREQRESP() function to compute the frequency response of the LTI motor model, evaluated at the exact same frequencies that were output from the `TFESTIMATE` function. Remember to use the `SQUEEZE` function on the output of the `FREQRESP` data.

```
H_sys = freqresp(motor_sys, F * 2 * pi);
%Process
mag_sys = squeeze(abs(H_sys));
phase_sys = rad2deg(unwrap(squeeze(angle(H_sys))));
mag_exp = abs(Txy);
phase_exp = rad2deg(unwrap(angle(Txy)));
```

You are now ready to demonstrate the validation result by plotting the frequency domain results. You will generate a single figure with three subplots. In the first subplot, you will need to plot the coherence vs. log frequency in Hz. In the second subplot, you will need to plot the phase angle in degrees vs. log frequency in Hz, for both the experimental (TFESTIMATE) data and the analytic (FREQRESP) data. In the third subplot, you will plot the dB magnitude vs. log frequency in Hz, for for both the experimental (TFESTIMATE) data and the analytic (FREQRESP) data. All plots must be properly annotated with axis labels, signal units, grid lines, and legends (for more than two signals per axes).

In all three subplots, you should use the `XLIM()` function to **limit the frequency range to only show data up to 20 Hz**. You are welcome to look at the entire frequency range (i.e. up to the Nyquist frequency), but you will see that the coherence is very poor above 20 Hz. We only need the model to be reasonably accurate below 10 Hz for this lab, and that is the frequency range where you excited the experimental servo-motor! Note that all three subplots MUST use the same frequency limits for direct comparison.

```
%Plot 4
figure
subplot(3,1,1);
semilogx(F_coh, Cxy, 'k');
title('Coherence vs Frequency');
xlabel('Frequency (Hz)');
ylabel('Coherence');
xlim([0.1, 20]);
grid on;
%Plot 5
subplot(3,1,2);
semilogx(F, phase_exp, 'b');
hold on;
semilogx(F, phase_sys, 'r--');
title('Phase Angle vs Frequency');
xlabel('Frequency (Hz)');
ylabel('Phase (degrees)');
legend('Experimental', 'Analytic Model', 'Location', 'southwest');
xlim([0.1, 20]);
hold off;
grid on;
```

```
%Plot 6
subplot(3,1,3);
semilogx(F, 20*log10(mag_exp), 'b');
hold on;
semilogx(F, 20*log10(mag_sys), 'r--');
title('Magnitude vs Frequency');
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
legend('Experimental', 'Analytic Model', 'Location', 'southwest');
xlim([0.1, 20]);
grid on;
hold off;

subplot(3,1,2)
legend(["Experimental", "Analytic Model"], "Location", "none", "Position", [0.1548
0.4937 0.2107, 0.0631])
```

