Assignment 07: Under Pressure

Now that we've learned how to design filters, how about we apply what we've learned to do some real DSP?

At Autonomy Lab, we recently redesigned our EV's braking system to pull on the brake pedal using feedback from a pressure sensor attached to the brake line. The signal from the pressure sensor feeds into one of the built-in ***analog-to-digial converters (ADCs)*** of an ESP32-S3 microcontroller where a brake control loop runs. The issue with this setup is that the signal from this sensor has a lot of noise, and to add insult to injury, the ADC found on the ESP32-S3 is of poor quality; all of this results in a feedback signal that is useless for PID control.

When designing this system, we gathered preliminary ADC data to see if we could even salvage the signal. Here are some things to note about the data you'll be working with:

- We sampled at 80 kHz with no hardware low-pass filter to determine how much we can lower the sample rate.
- The input signal was between 0 and 3.3 V and quantized to 12 bits by the ADC. Note that for this test, we hadn't yet removed the inherent DC bias of the pressure sensor nor amplified the sensor's signal to utilize the full range of the ADC.
- We pressurized the brake line using a Hydrastar trailer brake at 40% power for 1000 ms after a delay of 10 ms.
- You can find the CSV with the data here: `https://raw.githubusercontent.com/jacobkoziej/jk-ece210/master/src/assignments/07-under-pressure.d/40p_1000ms.csv`

Aside— Typically, you'd add a hardware low-pass filter to the input of an ADC at the Nyquist frequency to mitigate aliasing, but you'll see that we can get away without this!

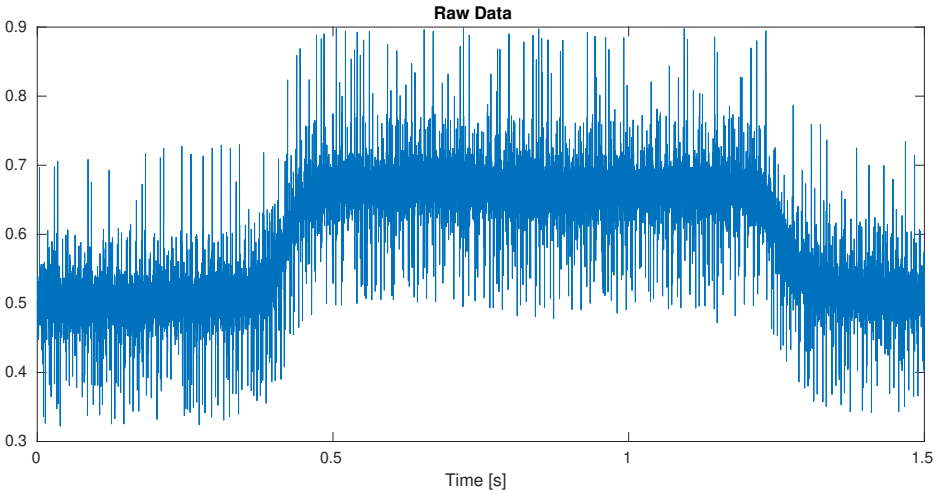
1. Use `readmatrix()` to read the CSV. The first column enumerates the samples, and the second column contains the quantized ADC sample.
2. Normalize the signal between 0 and 1, take the DFT, and remove the negative frequencies since this a real signal. Next, plot the DFT on a dB scale between 5 and 40 Hz. Add a title and axis labels, and use `yline()` to add dashed lines at -20 and -40 dB from the peak frequency on the plot.
3. Using the -20 dB dashed line on the previous plot, determine the corner frequency for an elliptic low-pass filter. Have at most 0.1 dB of passband ripple, attenuate by 40 dB in the stopband, and allow for a 10 Hz transition band.
4. Obtain a SOS digital filter and plot its magnitude response on the dB scale up to $f_{\text{stop}} + 20$ Hz. Also, add horizontal and vertical dashed lines to indicate

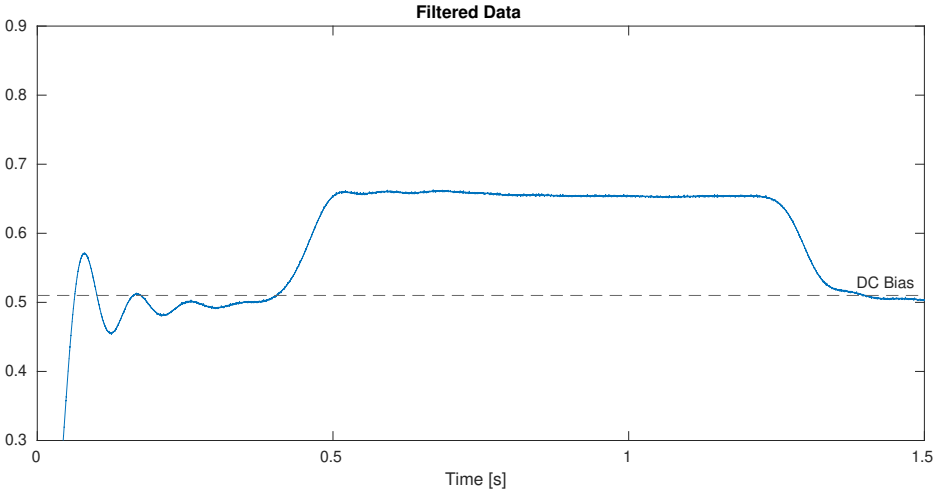
Aside— Since this is running on a microcontroller with strict timing requirements, we make the transition band larger than desired to decrease the filter order and as a result, the CPU time spent filtering.

the passband and stopband.

5. Apply the SOS digital filter to the original signal and plot it alongside the original signal. Additionally, add a dashed line around the DC bias for the filtered signal.

If done correctly, you should achieve something similar to the following:





Aside— The initial ripple in the filtered data is due to the jump discontinuity at $t = 0$ and the low order of the filter. This isn't an issue in our application, as our brake control will operate on the signal far beyond $t = 0$.

