Lab Team 11: The Pink Requirement
Members: Anna Corman, Joslyne Lovelace, Megan Shapiro
2/23/2020

Lab 2 Report

Introduction

The purpose of this lab is to begin using the Hall Effect sensor to ultimately read knob positions. We will read analog voltage values from the sensor, convert them to angular position and then to angular velocity. A filter is then applied to the angular velocity to produce a relevant plot of the data.

Procedure

Reading the Sensor

The first step to being able to read the voltage from the microcontroller is to build on previous code. We started by creating pseudo code to illustrate how we would use ring buffers and serial communication capabilities from previous labs. Tutorials for timers and ADC functions were researched and incorporated in the pseudo code.
Once the pseudo code was completed we set up the haptic paddle assembly ensuring that the VDD and 5V were connected with a single strand of wire. We then began to check our code and debug.
We then tested the voltage reading from the sensor compared to the reading on a multimeter to make sure the sensor was responding properly.

Implementing Timers

The clock speed of the arduino is 16MHz. This value was used to determine the clock time period as 1/16MHz. To determine the timer count to a delay of 1ms we used the equation:

$$timer\ count\ = \frac{required\ delay}{clock\ time\ period} - 1 = \frac{.001}{1/16M} - 1 = 15999$$

Since TIMER1 is a 16-bit timer (with 65535 counts) this works without a prescaler. However TIMER0 is only an 8-bit timer (with 255 counts). Calculating the timer counts gives 159999 counts, since it is larger than 255 counts we need a prescaler.

$$timer\ count\ = \frac{required\ delay}{clock\ time\ period/prescaler} - 1 = \frac{.01}{1/(16M/1024)} - 1 = 155.25$$

The prescaler of 1024 gives us a decimal number, however using the next lowest prescaler of 256 will give us a timer count of 624 which is above the 8-bit timer count still. The exact desired time is not possible, so the closest possible is obtained by rounding the decimal value calculated using the 1024 prescaler and rounding to the nearest integer, 155.

Calculating Angular Position and Angular Velocity

Next, in order to calculate the angular position we collected data points of the angle in degrees of the knob versus the voltage response. We then plotted the data and got a polynomial equation for the corresponding line of best fit. This calibration equation was then incorporated into the code. After checking the calibration equation we then incorporated the calculation to convert the angular position signal into angular velocity.

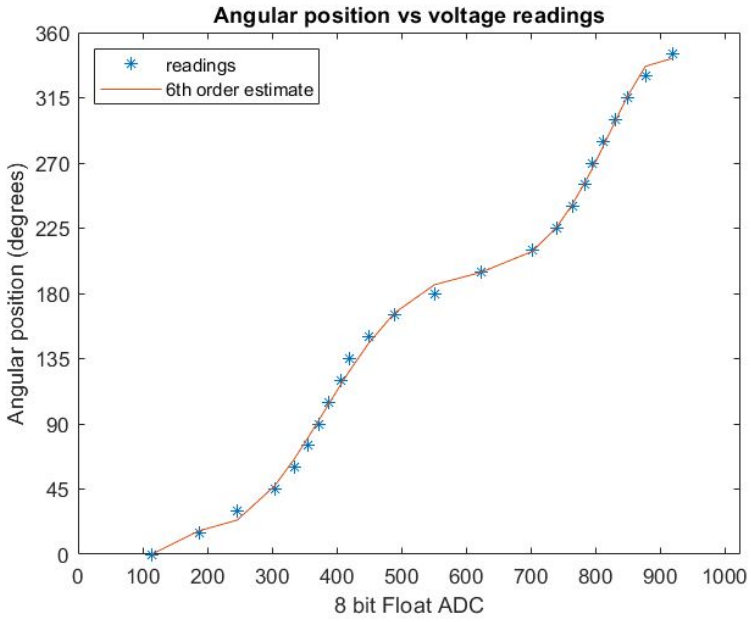

Figure 1: Position vs Voltage & Line of Best Fit

The polynomial line of best fit from the figure above is the following, where angPos is angular position and v is voltage as read from the hall effect sensor:

$$angPos = -1.0193E^{-13}(v^6) + 3.0609E^{-10}(v^5) - 3.5356E^{-7}(v^4) + 1.9698E^{-4}(v^3) - 0.0543(v^2) + 7.2116v - 354.5305$$

To then convert the angular position found above to angular velocity, the previous angular position was saved, and the difference between the current and last angular position was divided by the sampling period. The angular velocity was then also divided by 360 degrees in order for the output to be in revolutions per second, to make verification of the data easier.

Designing and Implementing a Filter

We then designed a low-pass filter with a cutoff frequency of 150 Hz, which is 30% of the nyquist frequency of our system. We chose to use a 4th order Butterworth filter after referencing a filter wizard site (analog.com/filterwizard). The 4th order Butterworth filter was a happy medium between the fewest stages and fastest settling. The coefficients were calculated by Matlab using 4 for the order and a cutoff frequency of 150 Hz. We then used the calculated ‘a’ and ‘b’ coefficients in our filtering functions described in our pseudo code.

```
clear; clc;
sample = 1000; %Hz
nyquist = sample / 2; %Hz
cutoff = 150; %Hz
Wn = cutoff / nyquist;
[b, a] = butter(4, Wn);
fprintf('Cutoff Frequency, %.0f Hz, is %.2f of nyquist, %.0f Hz.
\n',cutoff, Wn,nyquist);
fprintf('Filter is 4th order\n');
fprintf('B coefficients:\n');
fprintf('%.9f ',b);
fprintf('\nA coefficients:\n');
fprintf('%.9f ',a);
fprintf('\n');
```

```
Cutoff Frequency, 150 Hz, is 0.30 of nyquist, 500 Hz.
Filter is 4th order
B coefficients:
0.018563011 0.074252043 0.111378064 0.074252043 0.018563011
A coefficients:
1.000000000 -1.570398851 1.275613325 -0.484403368 0.076197065
```

Published with MATLAB® R2019b

Figure 2: Matlab Code and Results

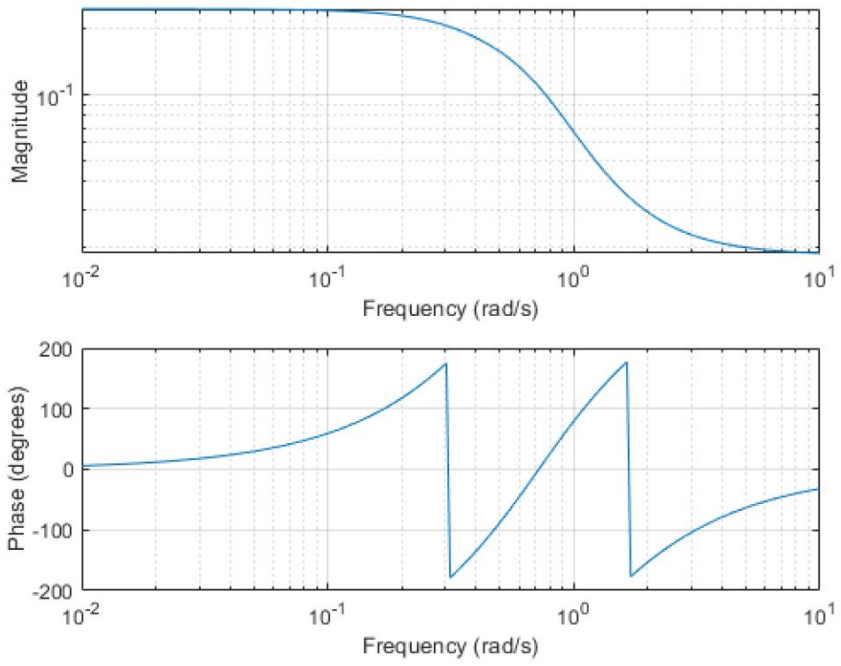

Figure 3: Bode Plot of Calculated Filter

Results

The plot gives a fairly decent flatline when the knob is untouched, so detecting zero is doing so sufficiently without noise. Originally, during demo, as the knob was moved via turning it slightly with the hand, the plot would react appropriately with the speed of the hand movement. Filtering, however, was not optimal during the demo. There were also some occasional spikes that seem relatively periodical that must be due to improperly handling wrapping in the code.

An error was found after a series of debugging, in the digital filter code where elements in the ‘a’ coefficient array were not properly being accessed. Once this was addressed, filtering positional results appeared as expected in the plot. Using this filtered position to then convert to velocity, however, was still not resulting in great looking plots.Small velocities made by spinning the wheel by hand gave good spike values on our plot of deg/s, but any constant velocity, such as that generated when the motor is connected directly to a battery, resulted in very noisy data.

We will endeavour to improve our velocity calculations before the next lab begins.

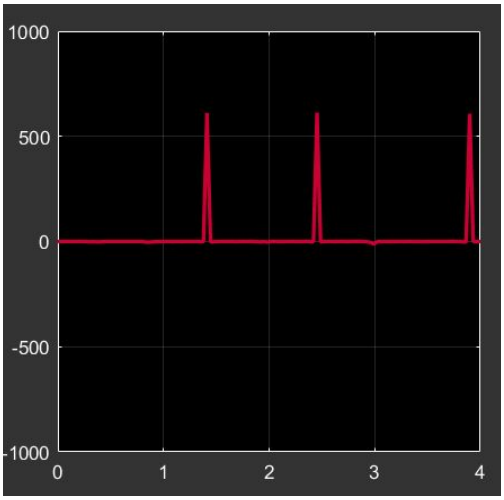

Figure 4: Velocity Readings (deg/s) with hand-turning

Discussion/Conclusions

We had difficulties with our filter due to the timing of the code. The amount of time it takes to print float is about 3 ms, which prevents the rest of the code from continuing to process commands, such as taking a measurement every 1 ms. To alleviate this issue, it was necessary to divide the print float into four separate print bytes (at about .8 ms each). This means that other parts of the code can operate while waiting for the serial register to empty.
The most important takeaway from this lab is to pay attention to the timing involved with different steps in the code. By not paying attention to the timing of different actions, the idea or plan of the code might not be possible. It is important to make the code as efficient as possible and not create unnecessary sitting and waiting periods.

Appendix A: Psuedocode Changes

PROGRAM: LAB2

DESCRIPTION: Reads voltage from sensor, converts voltage to angular position and angular velocity. Digitally filters information and sends it to Matlab via serial communication.

FUNCTION: main

INPUTS: None

RETURNS: 0

Set channel AI0 as output pin to Hall Effect sensor and set to 1

Connect analog input pin from Hall Effect sensor

Initialize: Serial Communication, Output Buffer, timers, Digital Filter, interrupts w/ CTC, adc, lastPosition

FOR all time DO:

if TIMER0_Flag

print_float(dequeue output)

Reset flag

elseif TIMER1_Flag

get analog input

~~angularVelocity(input)~~

Convert analog voltage to position

Convert position, lastPosition to velocity

Filter Velocity

add velocity to output queue

update lastPosition

Reset flag

END FOR

RETURN

FUNCTION: init_TIMER0

INPUTS: int pre-scaler

RETURNS: 0

DESCRIPTION: Initializes timer for correct prescaler if necessary and sets timer to 0

Set prescaler register to desired value

Set count to 0

RETURN

~~FUNCTION: ISR for TIMER0~~ If statements were used in main while instead of ISR functions

FUNCTION: init_TIMER1
    INPUTS: int pre-scaler
    RETURNS: 0
    DESCRIPTION: Initializes timer for correct prescaler if necessary and sets timer to 0

    Set prescaler register to desired value
    Set count to 0
RETURN

~~FUNCTION: ISR for TIMER1~~ If statements were used in main while instead of ISR functions
~~FUNCTION: angularVelocity~~ Included in main loop, no function necessary

FUNCTION: adc_init
    INPUTS: None
    RETURNS: None
    DESCRIPTION: Initialize on-board Analog-Digital-Converter

    Set ADC reference values to internal sources
    Set ADC prescale
RETURN

FUNCTION: adc_read
    INPUTS: Channel to convert
    RETURNS: Converted value
    DESCRIPTION: Trigger an analog-to-digital conversion on a certain channel

    Mask input channel to values 0-7
    Begin conversion on channel
    Wait for conversion to complete
RETURN converted value

PROGRAM: Digital Filter
    DESCRIPTION: Takes values and filters for smooth signal

FUNCTION initFilter
    INPUTS: None
    RETURNS: None
    DESCRIPTION: Initialize variables used in filter process
    Initialize a array, b array
    Initialize rb_F input
    Initialize rb_F output
    Initialize float angVel

```
FUNCTION: filterValue
        INPUTS: Angular Velocity
        RETURNS: Filtered Angular Velocity
        DESCRIPTION: Implements a n-order filter

        FOR each i in b
                Multiply b[i] by input[i]
                Add to input buffer
                Accumulate in angVel variable
        END
        FOR each i in a, where i>0
                Multiply by a[i] by output[i]
                Accumulate in angVel variable
        END
        Multiply angVel with inverse of a[0]
        Add angVel to output buffer
RETURN angVel
```

Appendix B: Atmel Code

5. Lab Report: Include your Atmel code in the appendix of your report. Make sure it is well organized, clear and commented .