

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 $1/16\text{MHz}$. To determine the timer count to a delay of 1ms we used the equation:

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

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.

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

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.

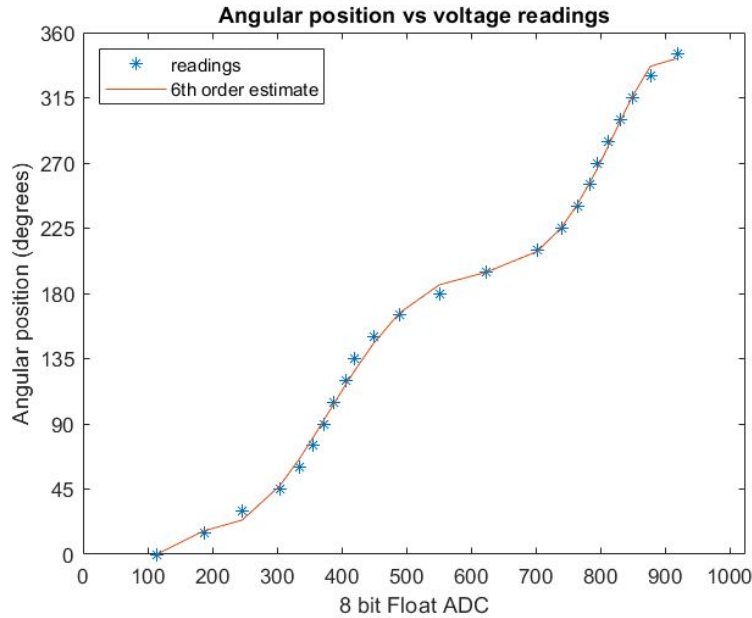


Figure 1: Position vs Voltage & Line of Best Fit

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('%0.9f ',b);
fprintf('\nA coefficients:\n');
fprintf('%0.9f ',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

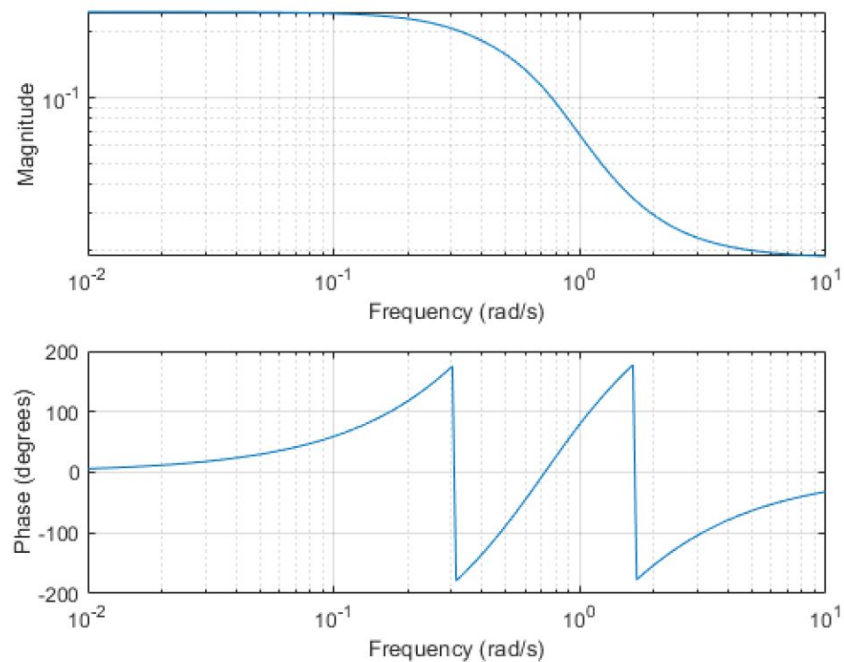


Figure 3: Bode Plot of Calculated Filter

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.

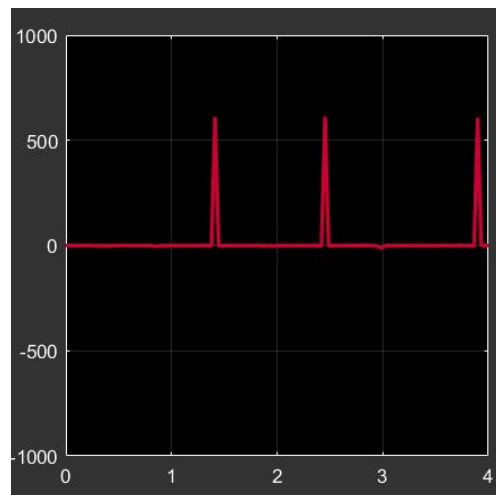


Figure 4: Velocity Readings (deg/s) with hand-turning

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.


```
1  /*
2   * Mech-Lab2.c
3   *
4   * Created: 2/12/2020 1:14:53 PM
5   * Author : Megan Shapiro
6   */
7
8  #define F_CPU 16000000L //Clock speed of Arduino 16 MHz
9  #define BAUD 9600 //Desired Baud Rate of Serial Communication
10 #define MYUBRR F_CPU/16/BAUD-1 //Calculated value of UBRR to initialize USART
11 #define TRANSMIT_READY UCSR0A & (1<<UDRE0)
12
13 #include <util/delay.h>
14 #include <avr/io.h>
15 #include <avr/interrupt.h>
16 #include <math.h>
17 #include <stdlib.h>
18 #include "Serial.h"
19 #include "Ring_Buffer.h"
20 #include "Digital_Filter.h"
21
22
23 void timer0_init();
24 void timer1_init();
25 void adc_init();
26 uint16_t adc_read(uint8_t ch);
27
28 union floatChars {
29     float asFloat;
30     char[4] asChars;
31 }
32
33 int main(void)
34 {
35     struct Ring_Buffer_F input_queue;
36     struct Ring_Buffer_C output_queue;
37     USART_Init(MYUBRR);
38     rb_initialize_F(&input_queue);
39     rb_initialize_C(&output_queue);
40
41     timer0_init();
42     timer1_init();
43     adc_init();
44     digital_filter_init(0);
45
46     //Set AI0 to Output and rest as Input
47     DDRC |= 0b00000001;
48
49     //Set output to 1 to power sensor
```

```
50     PORTC |= 0b00000001;
51
52     //Sampling frequency for converting to velocity, 1/0.001
53     float sampPer = 1000;
54     float volt = 0;
55     float angPos = 0;
56     float angPosLast = 0;
57     float angVel = 0;
58     float filteredVel = 0;
59     union floatChars printVal;
60     float convertCoeff[] = {-354.5305, 7.2116, -0.0543, 1.9698E-4, -3.5356E-7, 3.0609E-10, -1.0193E-13};
61     float tempSum = convertCoeff[0];
62
63     while (1)
64     {
65         //if TIMER0_flag
66         if((TIFR0 & (1 << OCF0A)))
67         {
68             printVal.asFloat = filteredVel; //edit so we don't drop readings during prints
69             for(int i = 0; i < 4; i++){
70                 rb_push_back_C(&output_queue, printVal.asChars[i]);
71             }
72             //reset TIMER0_flag
73             TIFR0 |= (1 << OCF0A);
74         }
75         //if TIMER1_flag
76         if((TIFR1 & (1 << OCF1A)))
77         {
78             //read voltage
79             volt = adc_read(1);
80
81             //convert to position in radians
82             // Apply 6th order best fit line found in Matlab
83             for (int i = 1; i <= 6; i++){
84                 tempSum += convertCoeff[i]*volt;
85                 volt *= volt;
86             }
87             //wrap result
88             angPos = abs(tempSum);
89
90             //convert to velocity
91             angVel = (angPos - angPosLast) *0.00277778*sampPer; // rev/s
92
93             //add angPos to queue
94             angPosLast = angPos;
95
96             //filter velocity
```

```
97         filteredVel = filterValue(angVel);
98
99         //reset TIMER1_flag
100         TIFR1 |= (1 << OCF1A);
101     }
102     if (rb_length_C(&output_queue) > 0 && TRANSMIT_READY){
103         print_byte(rb_pop_front_C(&output_queue));
104     }
105 }
106 }
107
108 void timer0_init()
109 {
110     // enable CTC for Timer0
111     TCCR0A |= (1 << WGM01);
112     // enable prescaler of 1024 for Timer0
113     TCCR0B |= (1 << CS02)|(1 << CS00);
114     // initialize counter to zero
115     TCNT0 = 0;
116     // initialize compare value for CTC
117     OCR0A = 155;
118 }
119
120 void timer1_init()
121 {
122     TCCR1A |= 0;
123     // Enable CTC for Timer1 with no prescaler
124     TCCR1B |= (1 << WGM12)|(1 << CS10);
125     // initialize counter to zero
126     TCNT1 = 0;
127     // initialize compare value
128     OCR1B = 15999;
129 }
130
131
132 void adc_init() {
133
134     //Set reference to built in channels
135     ADMUX = (1<<REFS0);
136     //Enable ADC w/ prescaler
137     ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
138
139 }
140
141 uint16_t adc_read(uint8_t ch)
142 {
143     //select channel to read
144     ch &= 0b00000111;
145     ADMUX = (ADMUX & 0xF8)|ch;
```

```
146     //start conversion
147     ADCSRA |= (1<<ADSC);
148     //wait for conversion to complete
149     while(ADCSRA & (1<<ADSC));
150     //return result
151     return (ADC);
152 }
```

```
1  #include <avr/io.h>
2  #include "Digital_Filter.h"
3  #include "Ring_Buffer.h"
4
5  struct Ring_Buffer_F inputs;
6  struct Ring_Buffer_F outputs;
7
8  //a[0] should be entered as the inverse of the coeff, for computation speed
9  //float b[5] = {0.0940, 0.3759, 0.5639, 0.3759, 0.0940}; // cutoff frequency was 250
10 //float a[5] = {1.0000, 0.0000, 0.4860, 0.0000, 0.0177};
11 float b[5] = {0.018563010626897, 0.074252042507589, 0.111378063761383,
               0.074252042507589, 0.018563010626897}; // cutoff frequency was 150
12 float a[5] = {1.0000000000000000, -1.570398851228171, 1.275613324983279,
               -0.484403368335085, 0.076197064610332};
13 uint8_t i;
14
15 /* Initialization */
16 void digital_filter_init(){
17     rb_initialize_F(&inputs);
18     rb_initialize_F(&outputs);
19
20     for(i = 0; i <= order + 1; i++){
21         rb_push_front_F(&inputs, 0);
22         rb_push_front_F(&outputs, 0);
23     }
24
25     return;
26 }
27
28 float filterValue(float newInput){
29     rb_pop_back_F(&inputs);
30     rb_pop_back_F(&outputs);
31     rb_push_front_F(&inputs, newInput);
32     float newOutput = 0;
33     for(i=0; i < order + 1; i++){
34         newOutput += b[i]*rb_get_F(&inputs,i);
35         if (i>0){
36             newOutput -= a[i]*rb_get_F(&outputs,i-1);
37         }
38     }
39     newOutput *= a[0];
40     rb_push_front_F(&outputs, newOutput);
41     return newOutput;
42 }
```