Colorado School of Mines
Department of Mechanical Engineering
MEGN 540: Mechatronics

Laboratory 2: Timers and ADC, Sensor Calibration, Data Streaming, Filtering Version 3.0

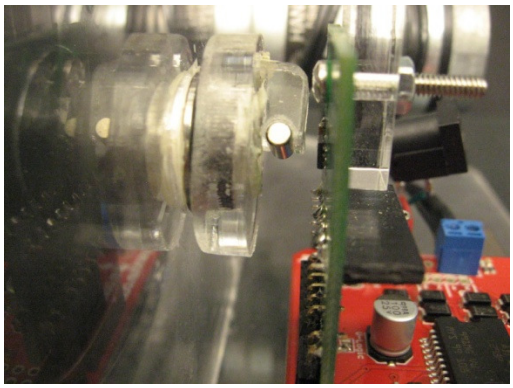
Objective:
In this lab, you will familiarize yourself with the Dual-Axis Hall Effect sensor (EM3242) used for handle/knob position sensing on the BRL haptic paddle. You will be reading analog voltage values from the sensor and converting those values to angular position. You will estimate angular velocity based on position readings, and design and implement an appropriate filter for the velocity signal. You will learn how to use the timer functions to adjust the sampling and loop rate of your program.

Equipment:

- Haptic Paddle
- Atmel Studio
- Multimeter
- Computer (with Windows OS)
- Matlab
- AVRISP mkII programmer

Background:
The Dual-Axis Hall Effect sensor on the paddle produces an analog voltage proportional with the paddle handle position, and allows position measurement across a full 360 degree rotation. The datasheet for the EM3242 sensor you will be using on the haptic paddle is available on the course website (Canvas).

The Hall Effect
The Hall effect refers to an electric potential that is caused by a magnetic field that is not parallel to the motion of moving charges. As the electrons in the paddle's sensor move through the magnetic field induced by the permanent magnet attached to the handle, they experience a force called the Lorentz force. This causes the paths of the electrons to deflect in the direction of the magnetic field, resulting in a higher concentration of charged particles on one side of the conductor than the other, resulting in an electric potential. As the handle rotates, the magnetic field orientation also changes with respect to the sensor and therefore the electric potential across the conductor can be used to determine the angle of the handle.



Notice that a small magnet is positioned directly against the Hall-Effect sensor

Sampling Rate, Program Loop Rate, Control Loop Rate
Sampling rate refers to the rate at which a continuous (analog) signal gets sampled at, as it is being digitized/discretized, as in the case for reading any analog sensor voltage into a computer

or a microcontroller, through an analog input channel. In this process, it goes through analog to digital conversion (ADC). The faster the sampling rate (or the shorter the sampling period) is, the closer the sampled version of the signal will be to the original analog signal. The actual voltage values also cannot be read at infinite intervals, rather there is a finite number of signal levels that can be resolved by the analog-to-digital converter in the microcontroller or the data acquisition device, dictated by its resolution. For example, an 8-bit analog input channel will read the voltage level and output the closest voltage level out of the 2^8 levels. Again higher resolution allows more accurate representation of the analog signal.

In our setup, sampling rate refers to the number of samples per unit time that are being read into an analog input channel on the microcontroller. This sampling rate will fully depend on the code you will have on the microcontroller. The sampling rate will be equal to the main program loop rate (a common scenario), if in every loop of the main program, only one sample is read. If more than one sample is read during a single loop of the main program, the sampling rate will be higher than the program loop rate. Sampling of a signal using ADC can be done much faster than 1kHz. This is very helpful in the sense that, completing one ADC cycle can take much shorter than 1ms (in the order of μs), so that the rest of the time can be used by other functions in the main loop of the program, and the main loop can consistently keep running at 1kHz.

Yet another *rate* that is important is the *control loop rate,* which will again usually be equivalent to the main program loop rate. In any feedback control system, one control loop contains the actions that take place from reading a new value from the sensor, doing all the comparison and control effort calculations, to sending out a control effort signal to the actuator. The time it takes for one control loop is referred to as the control loop period. The reciprocal of this period is the control loop rate. Faster control loop rates are almost always desirable because they add to the controlled system's stability and performance. For haptic interfaces, a control loop rate of 1 kHz (which corresponds to a 1 ms loop and sampling period) is very common, since the realism and the continuity of haptic interactions quickly degrade with slower loop rates.

You will use the timer functions of AVR to control these rates. For this lab, you want to achieve a 1 kHz sampling rate, which will also be your main loop rate. Since you will not yet have a closed-loop controller, this is not your control loop rate, but will become so when you add a control law to your main loop in future labs. Also, you want to achieve a data transmission (serial communication update) rate of 100 Hz from the microcontoller to Matlab.

Preparation Materials / Tutorials

The following are recommended reading that will prepare you for the tasks in this lab assignment which involve TIMER and ADC functions of AVR microcontrollers. Going through them in the order below is recommended.

http://maxembedded.com/2011/06/introduction-to-avr-timers/

http://maxembedded.com/2011/06/avr-timers-timer0/

http://maxembedded.com/2011/06/avr-timers-timer1/ -content here is similar to timer0

http://maxembedded.com/2011/06/avr-timers-timer2/ -content here is similar to timer0/1

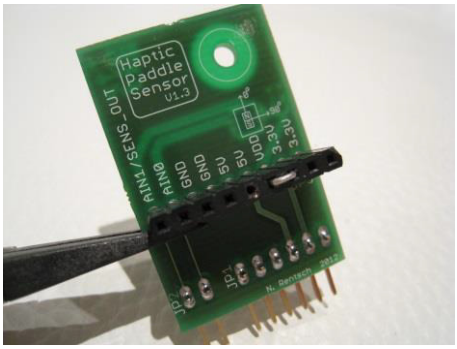
http://maxembedded.com/2011/07/avr-timers-ctc-mode/ -this is the most important one where everything comes together for timer and interrupt functions.

http://maxembedded.com/2011/06/the-adc-of-the-avr/ -this is on ADC

Here is an example on using TIMER0 together with ADC:
http://www.embedds.com/adc-on-atmega328-part-2/

Procedure / Assignment:
This lab is divided into three steps. You will learn how to read and transmit voltage values from the Hall-Effect sensor at the desired rate, and display them continuously in Matlab (analog input (or ADC) combined with timing and continuous/streaming serial communication). Using these voltage values, you will calculate angular position and angular velocity and transmit those values from the board to be displayed in Matlab (sensor calibration). Finally, you will design and implement a digital filter to clean up your noisy angular velocity signal and transmit the filtered signal to Matlab (signal conditioning/filtering).

Reading the Sensor
1. From inspection of your custom-built circuit board plugged into the Ardumoto board, you should be able to see that you will be reading voltage values from AI1 channel of the microcontroller. Make sure that the sensor board is correctly connected to the Uno (or to the Ardumoto stacked on the Uno) by checking the matching pins. Use a short, single strand wire as a jumper to connect the pins VDD and 5V to each other (this wire is mostly likely already in place).



2. Connecting VDD and 5V effectively powers the sensor with 5V. However, the sensor needs to be "enabled" by first setting channel AI0 to be an output, and then setting it to "1". This will apply 5V to AI0, which will enable the sensor. Until you do so, you may not see any voltage change from the sensor as you move the paddle handle.

On workbenches in Brown W220, there are multimeters, but you also need probe cables for them. There are a few probe cables in the class cabinet (on the right of shelf 5 from top) you can use. Make sure to store them back at their original place after your use so that your classmates can also find and use them when they need them.

Using a multimeter, make sure that the voltages you are reading from the sensor match the expected voltage output of the sensor, as you move the paddle handle. Compare these voltage values to what you would expect from the Hall-Effect Sensor datasheet. If your sensor does not seem to respond properly, troubleshoot accordingly.

3. It should be apparent from the previous lab assignment that, over the serial port, sending regular ASCII characters (which are 8-bit = 1 byte) is easier, but that floating point numbers (floats are 32-bit) require some more work. Arduino IDE serial monitor, or other serial terminal programs are limited to handling only 8-bit data and none can handle exchange of float (32-bit) information over the serial port. Within the context of the current lab (position sensing for the handle), it is easy to notice that reading a sensor with only 8-bit precision would be very coarse and limiting. Using floats for communication between the microcontroller and Matlab allows significantly better precision for sensor data.

Write a code in Atmel Studio (or build on code from the previous lab) that reads the AI1 pin voltage with 10-bit precision (1024 different values) at a main program loop rate of 1kHz, and communicates (streams) the voltage values to Matlab at a rate of 100 Hz. For sending the float voltage values from Atmel to Matlab, use the posted `printfloat` function from Lab 1 solutions. For timing of your main loop at 1ms, use TIMER1. For timing your serial communication rate at 10 ms, use TIMER0. Getting exactly 1ms or 10ms may not be possible: choose the closest possible value.

For the streaming part, two Matlab files are provided to you: COMmonitor.m allows continuous streaming of float values from the microcontroller to the command window. RealTimePlot.m does the same, in a plot format. See the information included in these files for further details. A baud rate of 9600 is recommended. The correct COM port should be written in these files (search for PortName in the files and type in the correct port number). Start with streaming a constant float value and make sure that you are receiving it correctly on the Matlab end. Then move onto streaming of the sensor reading in Volts.

Create a real-time plot of this data in Matlab using RealTimePlot.m. You can update the y-axis limits of the plot to match the specific data you are streaming (for example, sensor voltage will change only between 0-5V, whereas position in degrees can take values between 180 to -180).

Important: For both COMmonitor.m and RealTimePlot.m, receiving streamed values correctly can take hitting the reset button on your microcontroller several times. This is because the 8-bit data packets will arrive continuously, without any indicator of which packet is the first packet of a 4byte float. Matlab's reading them in an incorrect order will give you values that are too small (<1e-10) or too large (>1e10), or ones that jump up and down pretty drastically. You will get extra credit if you develop a way to resolve this data syncing problem (without adding delay functions). You can modify the Matlab files for this purpose.

Calculating Angular Position and Angular Velocity

1. Calculate the angular position as marked on the paddle, as a function of output voltage from the sensor. For this, collect 8 to 10 data pairs for voltage versus degrees and do a line fit in Excel or Matlab. The equation for the line should give you the calibration equation of your sensor, for voltage input to degrees output.

If everything is working as they are supposed to, the data will be highly linear. If your data is significantly nonlinear, then check the vertical alignment of the magnet with the Hall effect sensor while varying the handle position. If the magnet is moving up/down as you rotate the handle with respect to the sensor, that would be the reason for the nonlinearity. You can try to align them better to get a more linear response.

2. Implement the calibration calculation into your code on the microcontroller so that now position in degrees is streamed instead of voltages, and make sure that the angular position you are reading in Matlab matches the position of the handle in real-time. Note that if the handle's position with respect to the shaft (hence the Hall Effect sensor) changes (for example if it slips), your calibration would change.

3. Calculate the angular velocity from your angular position signal, using the finite-difference method (FDM). In this method, velocity is estimated as

$$\omega(t + \Delta t) = \frac{\theta(t + \Delta t) - \theta(t)}{\Delta t}$$

where ω is angular velocity; θ is angular position; and Δt is the sampling period (also called the time step).
In your case, the sampling period (time step) is equal to the main loop period, which is the time it takes for one iteration of the main loop on the microcontroller. This should be set to 1ms by making use of the timer functions.

4. Implement this calculation into your code and stream angular velocity in deg/sec to Matlab. Pay special attention to the discontinuous point in the voltage/position (implement a velocity saturation function for this discontinuous point).

Designing and Implementing a Filter

It should be apparent that the angular velocity you have calculated is too noisy of a signal to be useful for any practical purpose. You will need to filter this signal.

1. Design a low-pass filter with proper cutoff frequency (determine its discrete transfer function) and apply it to your angular velocity signal, in the form of a difference equation in your code. You may want to first attempt to design and test the filter on the velocity signal in Matlab to verify that it functions properly, before implementing it in your microcontroller code. You may want to consult to the "filtering.m" file covered in class to remember how to design a filter. Again, your sampling period (time step) is an important parameter for your filter's discrete

transfer function. For obtaining a difference equation representation from a Z-domain transfer function, you can check Matlab help documentation for discrete filters ("filter" command), or the relevant slides covered in lectures.
2. Output your filtered angular velocity signal to Matlab and plot the filtered angular velocity in real time.

Deliverables:

1. Demo: Demonstrate your Atmel code that displays the correct angular position in real-time as you move the haptic paddle handle.
2. Demo: Demonstrate your Atmel code that displays a real-time plot of your filtered angular velocity. The plot updates should be smooth.
3. Lab Report: Explain how you implemented your sampling and loop period of 1ms, and serial communication period of 10ms.
4. Lab Report: Report your calibration procedure, plots and line fit. Report how you calculated angular position and angular velocity from voltage readings.
5. Lab Report: Include your Atmel code in the appendix of your report. Make sure it is well-organized, clear and commented .
6. Lab Report: Explain how you designed and implemented your filter. If you used Matlab for this purpose, include your m code. How did you select the cutoff frequency? How did you choose the filter order? What code did you use to obtain its transfer function? Include its Bode plot in your report. How did you convert your TF to a difference equation?
