Tufts Robotics Club
Interfacing Arduino with Matlab

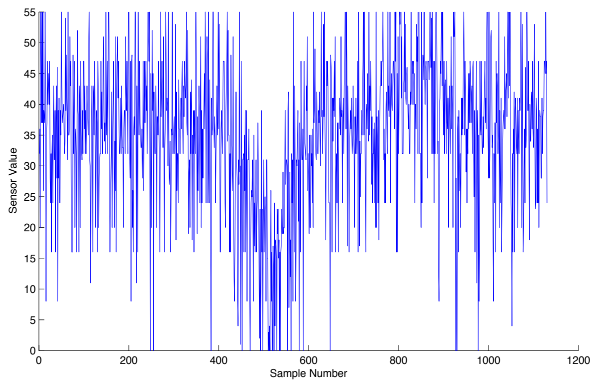
Will Langford
03/28/11

Designing a Low Pass Filter

Low pass filters are a great way of suppressing noise in a sensor measurement.

A great example of the power of these kinds of filters can be seen in our recent foray with an IR Photodiode which was used in the firefighting robots to detect a candle...

Here's what the raw data looked like:



There's some obvious trends but overall it's pretty fuzzy and getting code to decide whether or not the robot's facing a candle with this kind of data isn't so much fun.

Enter simple, first order low pass filters. For those of you with some differential equations under your belt you might represent the filter in the Laplace domain as the transfer function:

$$\frac{y}{u} = \frac{\omega}{s+\omega}$$

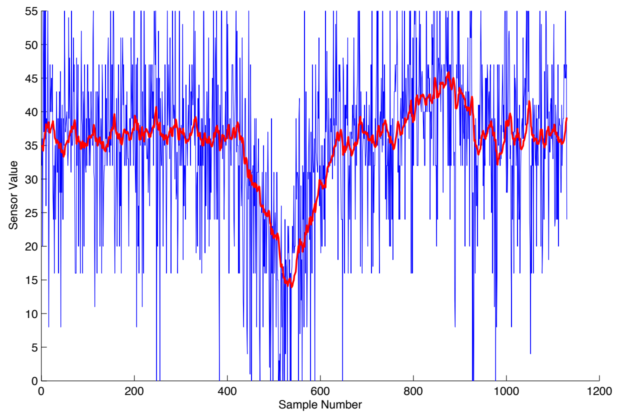
Where u is the raw data, y is the filtered data, and w is the break frequency in radians/sec.

This can then be transformed into the discrete time form that can be implemented in code on the Arduino. It becomes:

$$y_i = \frac{w * u_i * dt + y_{(i-1)}}{1 + w * dt}$$

or, in Arduino:

```
float low_pass_filter(float u_new) {
  y_old = y_new;
  y_new = (w*u_new*dt+y_old)/(1+w*dt);
  return y_new;
}
```



Once you've gotten the raw data (sensor data with timestamps) into Matlab, filters with various cutoff frequencies can be simulated quickly and easily using the same discrete-time form that will eventually be implemented on the Arduino.

Try a few different break frequencies until you find one that has a good compromise between noise suppression and latency.

We found that this simple first-order filter was roughly equivalent to a moving average of 50 samples. The thing that makes the low pass filter so awesome is that instead of a moving average where lots of values need to be stored in memory, our filter only needs to know one previous state: y_old.

That's it! Check out the attached Arduino and Matlab code for a jump-start in implementing your own filter.