Labmaster Calibration

339 - Measurements Lab
Due: February 10, 2014

February 4, 2014

1 Introduction

Calibration is the establishment of the relationship between a well known quantity, and an intrinsic property of some system. This relationship then allows the use of this system to make measurements in terms of the original well known quantity. Back when the meter was first defined, there was, and still is only one platinum-iridium bar which served as the standard. All other measurements of length had to derived by comparison with this unit. This done, individuals who have measuring instruments calibrated to the original unit can discuss measurements using their instrument with one another, confident that with in the error of their instrument they agree about the values they measure. Today where ever possible units have been defined in terms of fundamental atomic properties, and the speed of light. In the particular case of this lab, the DAC provides voltages on the analog output which is controlled by the 8 bit number applied to the digital input controlled by software. Similarly the ADCs allow software to read 10 bit numbers which corresponts to a particular voltage applied to the analogue input of the device. These numbers are probably useless unless they can be converted into standard units.

To determine the relationship between the voltage upon the output of a DAC and the 8 bit number applied to its input, a set of numbers should be sent, and the response of the output measured with a voltmeter. The voltage reponse can then be plotted as a function of the number sent, and a decision made about the nature of the relationship. If the relationship is a simple function, then the conversion can be obtained quite easily. For example, if the relationship were linear, that is

$$V_{\text{DAC}} = V_0 + GN_{\text{DAC}} \tag{1}$$

where V_{DAC} is the voltage at the output in reponse to the number N_{DAC} applied to the input port. V_0 and G would be properties of the DAC. If V_0 and G can be determined then the relationship can be inverted, and used to set the output of the DAC to a desired voltage. In the case of a linear relationship we can use *linear regression* to obtain the parameters.

2 Linear Regression

If we have a linear relationship between two variables,

$$y = a + bx \tag{2}$$

and a set of N measurements $((x_1, y_1, \sigma_1), (x_2, y_2, \sigma_2), ...(x_N, y_N, \sigma_N))$, then we can establish the parameters a and b which minimize following quantity,

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{a + bx_i - y_i}{\sigma_i} \right)^2. \tag{3}$$

If χ^2 is minimized, the following will be true,

$$\frac{\partial \chi^2}{\partial a} = \sum_{i=1}^{N} \left(\frac{2a + 2bx_i - 2y_i}{\sigma_i^2} \right) = 0 \tag{4}$$

and

$$\frac{\partial \chi^2}{\partial b} = \sum_{i=1}^{N} \left(\frac{2bx_i^2 + 2ax_i - 2x_i y_i}{\sigma_i^2} \right) = 0 \tag{5}$$

Expanding the summations in eq(4) and eq(5),

$$a \sum_{i=1}^{N} \frac{1}{\sigma_i^2} + b \sum_{i=1}^{N} \frac{x_i}{\sigma_i^2} - \sum_{i=1}^{N} \frac{y_i}{\sigma_i^2} = 0 \tag{6}$$

and

$$b \sum_{i=1}^{N} \frac{x_i^2}{\sigma_i^2} + a \sum_{i=1}^{N} \frac{x_i}{\sigma_i^2} - \sum_{i=1}^{N} \frac{x_i y_i}{\sigma_i^2} = 0 \tag{7}$$

Making the following substitutions,

$$\alpha = \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \qquad \beta = \sum_{i=1}^{N} \frac{x_i}{\sigma_i^2}$$

$$\gamma = \sum_{i=1}^{N} \frac{y_i}{\sigma_i^2} \qquad \lambda = \sum_{i=1}^{N} \frac{x_i^2}{\sigma_i^2}$$

$$\rho = \sum_{i=1}^{N} \frac{x_i y_i}{\sigma_i^2}$$

The equations may be combined and written

$$\begin{bmatrix} \alpha & \beta \\ \beta & \lambda \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \gamma \\ \rho \end{bmatrix} \tag{8}$$

Multiplying eq(8) with the inverse of the square matrix,

$$\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \alpha & \beta \\ \beta & \lambda \end{bmatrix}^{-1} \begin{bmatrix} \gamma \\ \rho \end{bmatrix} = \begin{bmatrix} M_{(1,1)} & M_{(1,2)} \\ M_{(2,1)} & M_{(2,2)} \end{bmatrix} \begin{bmatrix} \gamma \\ \rho \end{bmatrix} \tag{9}$$

This is easily implementable in MATLAB using the function `inv`, which returns the inverse of a square matrix. See the example code `use_inv.m` which shows how to build a matrix and get the inverse.

To obtain the uncertainties on the parameters, we use standard error propagation. Reading off the result for parameter a from eq(9),

$$a = M_{(1,1)} \sum \frac{y_i}{\sigma_i^2} + M_{(1,2)} \sum \frac{x_i y_i}{\sigma_i^2}. \tag{10}$$

So the uncertainty on parameter a is determined by

$$\begin{aligned}
\sigma_a^2 &= \sum_{i=1}^{N} \left(\frac{\partial a}{\partial y_i} \right)^2 \sigma_i^2 && (11)\\
&= \sum_{i=1}^{N} \left(M_{(1,1)} \frac{1}{\sigma_i^2} + M_{(1,2)} \frac{x_i}{\sigma_i^2} \right)^2 \sigma_i^2 && (12)\\
&= \sum_{i=1}^{N} \left(\left(M_{(1,1)}\right)^2 \frac{1}{\sigma_i^2} + \left(M_{(1,2)}\right)^2 \frac{x_i^2}{\sigma_i^2} + 2M_{(1,1)}M_{(1,2)} \frac{x_i^2}{\sigma_i^2} \right) && (13)\\
&= \left(M_{(1,1)}\right)^2 \sum_{i=1}^{N} \frac{1}{\sigma_i^2} + \left(M_{(1,2)}\right)^2 \sum_{i=1}^{N} \frac{x_i^2}{\sigma_i^2} + 2M_{(1,1)}M_{(1,2)} \sum_{i=1}^{N} \frac{x_i^2}{\sigma_i^2} && (14)
\end{aligned}$$

Recognizing the appropriate substitutions for the summations and grouping preemptively,

$$\sigma_a^2 = M_{(1,1)} \left(M_{(1,1)}\alpha + M_{(1,2)}\beta \right) + M_{(1,2)} \left(M_{(1,1)}\beta + M_{(1,2)}\lambda \right) \quad (15)$$

But, since M is the inverse of our original matrix,

$$\begin{bmatrix} M_{(1,1)} & M_{(1,2)} \\ M_{(2,1)} & M_{(2,2)} \end{bmatrix} \begin{bmatrix} \alpha & \beta \\ \beta & \lambda \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad (16)$$

eq(15) simplifies to

$$\sigma_a^2 = M_{(1,1)} \quad (17)$$

Similarly,

$$b = M_{(2,1)} \sum \frac{y_i}{\sigma_i^2} + M_{(2,2)} \sum \frac{x_i y_i}{\sigma_i^2}. \quad (18)$$

So the uncertainty on parameter b is determined by

$$\begin{aligned}
\sigma_b^2 &= \sum_{i=1}^{N} \left(\frac{\partial a}{\partial y_i} \right)^2 \sigma_i^2 && (19)\\
&= \sum_{i=1}^{N} \left(M_{(2,1)} \frac{1}{\sigma_i^2} + M_{(2,2)} \frac{x_i}{\sigma_i^2} \right)^2 \sigma_i^2 && (20)\\
&= \sum_{i=1}^{N} \left(\left(M_{(2,1)}\right)^2 \frac{1}{\sigma_i^2} + \left(M_{(2,2)}\right)^2 \frac{x_i^2}{\sigma_i^2} + 2M_{(2,1)}M_{(2,2)} \frac{x_i^2}{\sigma_i^2} \right) && (21)\\
&= \left(M_{(2,1)}\right)^2 \sum_{i=1}^{N} \frac{1}{\sigma_i^2} + \left(M_{(2,2)}\right)^2 \sum_{i=1}^{N} \frac{x_i^2}{\sigma_i^2} + 2M_{(2,1)}M_{(2,2)} \sum_{i=1}^{N} \frac{x_i^2}{\sigma_i^2} && (22)\\
&= M_{(2,1)} \left(M_{(2,1)}\alpha + M_{(2,2)}\beta \right) + M_{(2,2)} \left(M_{(2,1)}\beta + M_{(2,2)}\lambda \right) && (23)\\
&= M_{(2,2)} && (24)
\end{aligned}$$

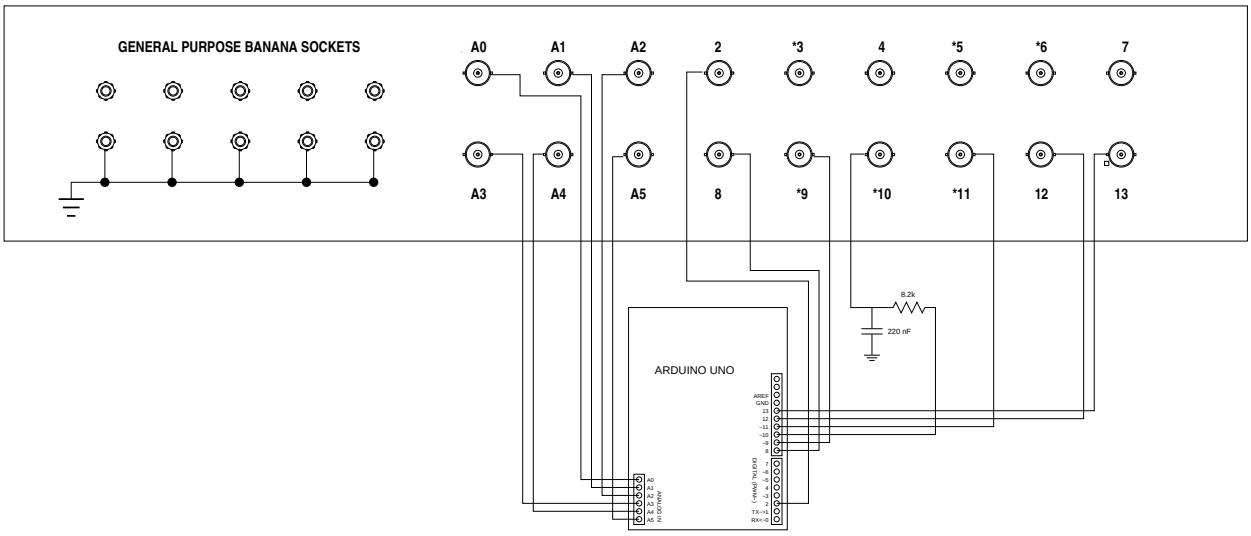

Figure 1: Arduino Front Panel

3 The Hardware

Figure 1. shows the front panel of a typical **Arduino** as we have installed them. The digital to input / outputs are located at the far right of the panel. Earlier in the text we have refered to the digital to analog converters (DAC). The Arduino Uno does not have any true DACs. It has some digital ports which are capable of pulse width modulation (PWM). These ports are identified by a tilde (˜) on the Arduino board, and by and asterisk (∗) on the front panel (since the label maker doesn't have a tilde character)[1] In order to fake a DAC, PWM output 10 has been wired up with a low-pass filter. PWM, as the name suggests gives an rectangular wave of with a duty cycle which varies according to the value requested — the minimum value with always be zero, maximum will always be 5V, 90% of maximum value will result in 90% duty cycle. PWM outputs are 8-bit, that is legal values are in the range 0 to 255.

The analogue to digital converter inputs are the six leftmost BNC connectors, there are six channels, 0 - 5. ADCs are 10 bit, that is they return numbers in the range 0 to 1023, all voltages are *limited* to 0 to 5V.

4 The Measurements

The first exercise will be to calibrate the pseudo DAC, Arduino pin 10. Use the function `analogWrite` to set the DAC and use a voltmeter to measure the resulting voltage at the output, How will you determine the error on this measurement. You should plot the response curve, and determine what the relationship is. If you are confident the relationship is linear you may use linear regression to characterize the DACs. If you correctly characterize your DACs you will be able to set the voltage of a given DAC to within the precision of the DAC. What is the precision of the DAC, what is the origin of this precision? You might consider writing a function `set_voltage` which provides a more useful interface to the DACs by accepting a voltage as a parameter. You should probably consider that the DAC is quantized and return the voltage which was actually set.

[1] Yeah, the labels are actually hand written because the label maker doesn't have any tape either!

If you have successfully calibrated the DAC, then you now have a precision voltage source, you can then use this to provide stimulus to the ADC channels, while recording the resulting numbers. You should do this with a program, rather than any manual activity. Verify that all six ADC channels work, and have similar characteristics. Again a useful function to write would be `get_voltage`.

When analysing the data, it will be useful to examine the residual of your fit, that is $y_i - f(x_i)$, this should be lie on the *x*-axis, with points scattered consistent with their uncertainty. It is not valid to claim that a fit is good because the fit-line looks identical to the data if the range of y_i is much greater than σ_i. Do not be surprised if your analysis reveals inaccuracies in the measurements reported by your voltmeter.

The final exercise is fairly open ended. You are to write a software function generator using the **Arduino** for output. A basic generator should be capable of outputing the basic functions sinewave, triangle, squarewave and sawtooth. These should have user specified amplitude and frequency. What limitations are placed upon these parameters? You should discuss in your report what your specifications on the function generator are, you will be asked to demonstrate your function generator, and will be marked upon how realistic your specifications were. The fact that the DAC is actually a PWM output with a low-pass filter will make this problem rather more interesting, since the raw signal sent to the pin will be low pass filtered. It will be possible to correct for this is you correctly characterize the behavior of the filter. I would suggest you do this by sending a series of sine waves of different frequency and measuring the actual amplitude. In order to correct for more complicated functions like square, triangle and sawtooth you might want to think about the forier transform of the wave you are attempting to generate. Do bear in mind however, that there will be limitations you can not correct away, such as rise-time.

5 Software

You are provided with the following **MATLAB** functions which are documented in more detail in the man pages listed on the wiki page for this experiment.

- `arduino_connect` - this is required to be called once before attempting to use any other arduino function.
- `analogWrite` - a simple function to set the PWM output.
- `analogRead` - a simple function to read a voltage from the analog to digital converter.
- `analogWriteVector` - Sends a vector of values to be sent to the PWM pin. This is a good candidate for the function generator. The vector will be cycled through until a new arduino function is called.
- `digitalWrite` - a simple function to set the logical value of a pin.
- `digitalRead` - a simple function to read the logical value of a pin. It might be interesting to characterize using the pseudo DAC, what range of voltages correspond to logical high and low.

These functions are contained within *M-files* which call `arduinoIO` which is what **MATLAB** calls a *MEX-file* (extension `.mexglx`). *MEX-file*. are compiled from **C**, and may burn your eyes if you try to look at them. The basic operation is as follows, **MATLAB** calls the function which parses the parameters and translates the **MATLAB** structures in to **C** structures, these are then checked for sanity and encapsulated into packets which are sent to the **Arduino** to perform the operation, results are then sent back for translation back into **MATLAB** structures. This is not important for your work, but is provided merely for those who want to know *how*.[2]

Also provided is `use_inv.m` this function gives a demo of the use of the MATLAB `inv` function, loosely based on the nomenclature developed in the **Linear Regression** section. It accepts 5 parameters, α, β, λ, γ and ρ and calculates a, b, σ_a and σ_b from them.

All provided functions can be found in `/mnt/resources/339/calib`.

[2] No useful answer to the question *why* is currently provided.