Ikhlas Attarwala / Fall 2019
EECS 331 / Computational Photography
Group #9 (w/ Yunhao Li, Deanna Dimonte)

Measuring the calibration settings of our NVIDIA Shield

In this homework, we measured the noise properties of our NVIDIA shield's camera by taking hundreds of photos, averaging the captured light intensities across photos, evaluating their noise variance, and using this to calculate the camera gain (sensitivity measured in photo-electrons). To begin, we wrote a short program in our tablet via Android Studio to capture 50 images of a static scene, at a desired sensitivity. The code I attached titled 'backbone_hw2.zip' is the same as my partner's, Deanna Dimonte. In order to avoid the effects of high frequency lighting, we shut off the lights in Mudd library and took the photos right near the front entrance so there was optimal lighting. We captured 50 photos of a gradient image (gradient image was used so that we can get a range of possible brightness values) at low sensitivity (100), and 50 photos at a high sensitivity (500). We made sure to be very careful not to move the tablet or paper so that 100 pictures were near identical. We did capture other photos at different sensitivities but didn't include them because we didn't realize the tablet was too slow and not recording all 50 images, so we had to slow it down and delete numerous sets.

Once we had the images, we processed them via MATLAB. I loaded each raw file and converted 50 of them per sensitivity into a 3d-array. I chose to select a region of 550 by 550 pixels, as that was the approximate range of the square gradient on the image. Figure 1 below shows my approximate selection.

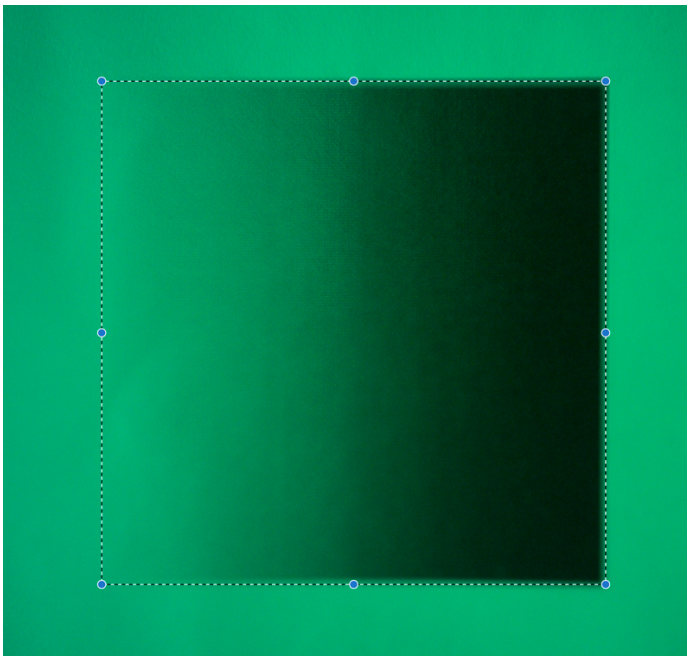

Figure 1 – Screenshot of my selection range for cropping the images.

After selection, of the 550x550 pixels, I needed to plot a histogram of each of the count of intensities per pixel, per sensitivity. The following are histograms for 9 chosen pixels of the low sensitivity (Figure 2) and high sensitivity (Figure 3).

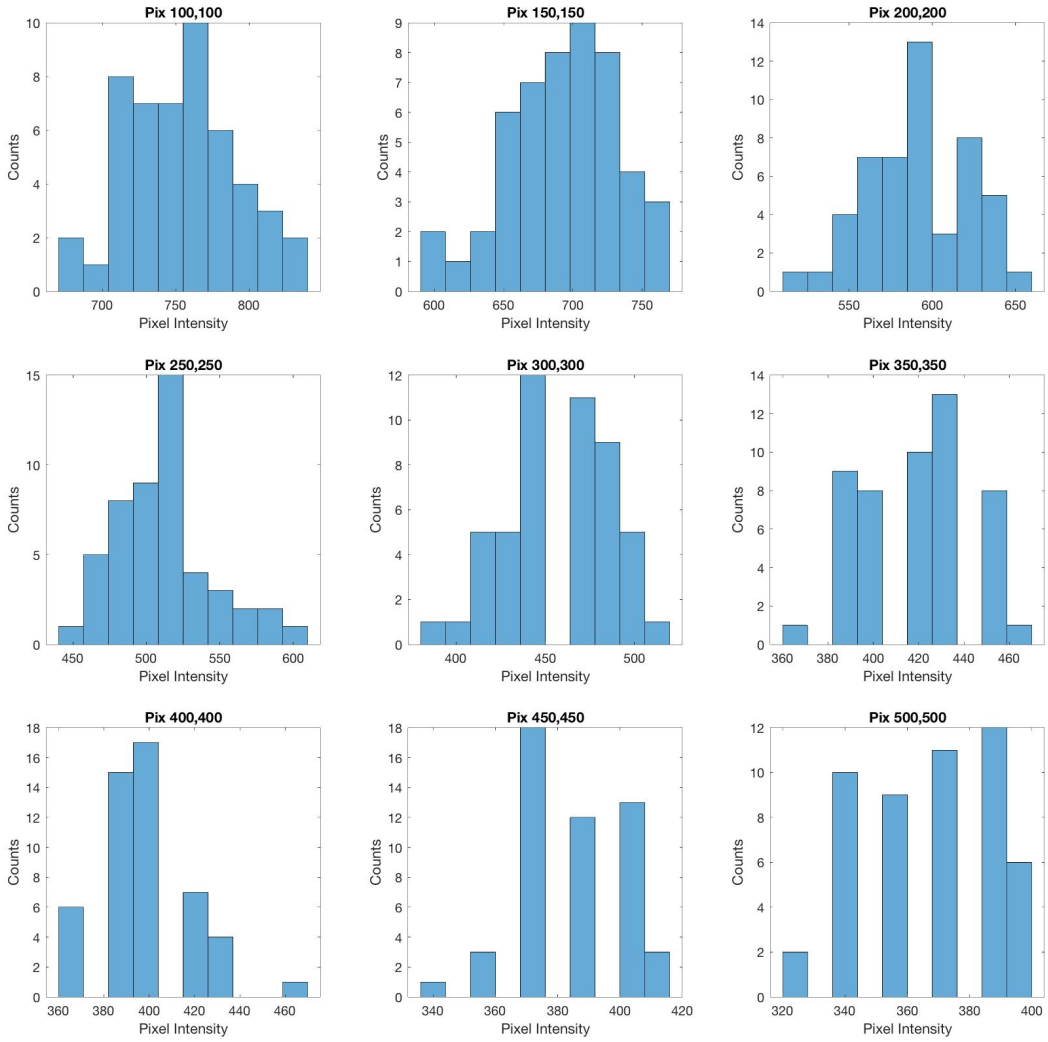

Figure 2 – Low Sensitivity (100) Pixel Intensity counts

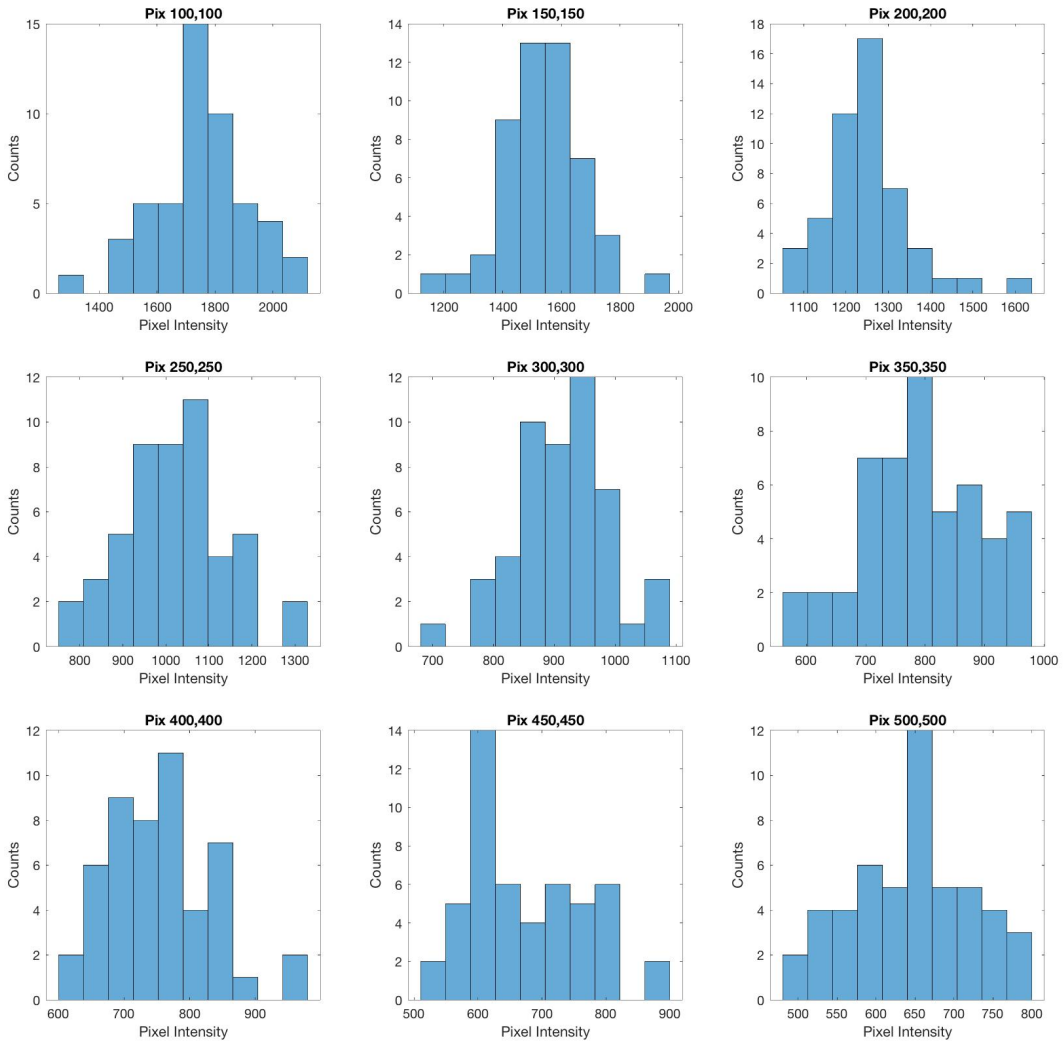

Figure 3 – High Sensitivity (500) Pixel Intensity counts

The PDF shapes per pixel tend to be a normal distribution, which is to be expected. From the top right of both figures, we choose pixel at location 100x100, and we move diagonally downward to 500x500, and as the gradient changes, we observe a shift in pixel intensities. It's still hard to observe here as the intensities weren't hugely different, but we can see the change much better in Figure 4, below.

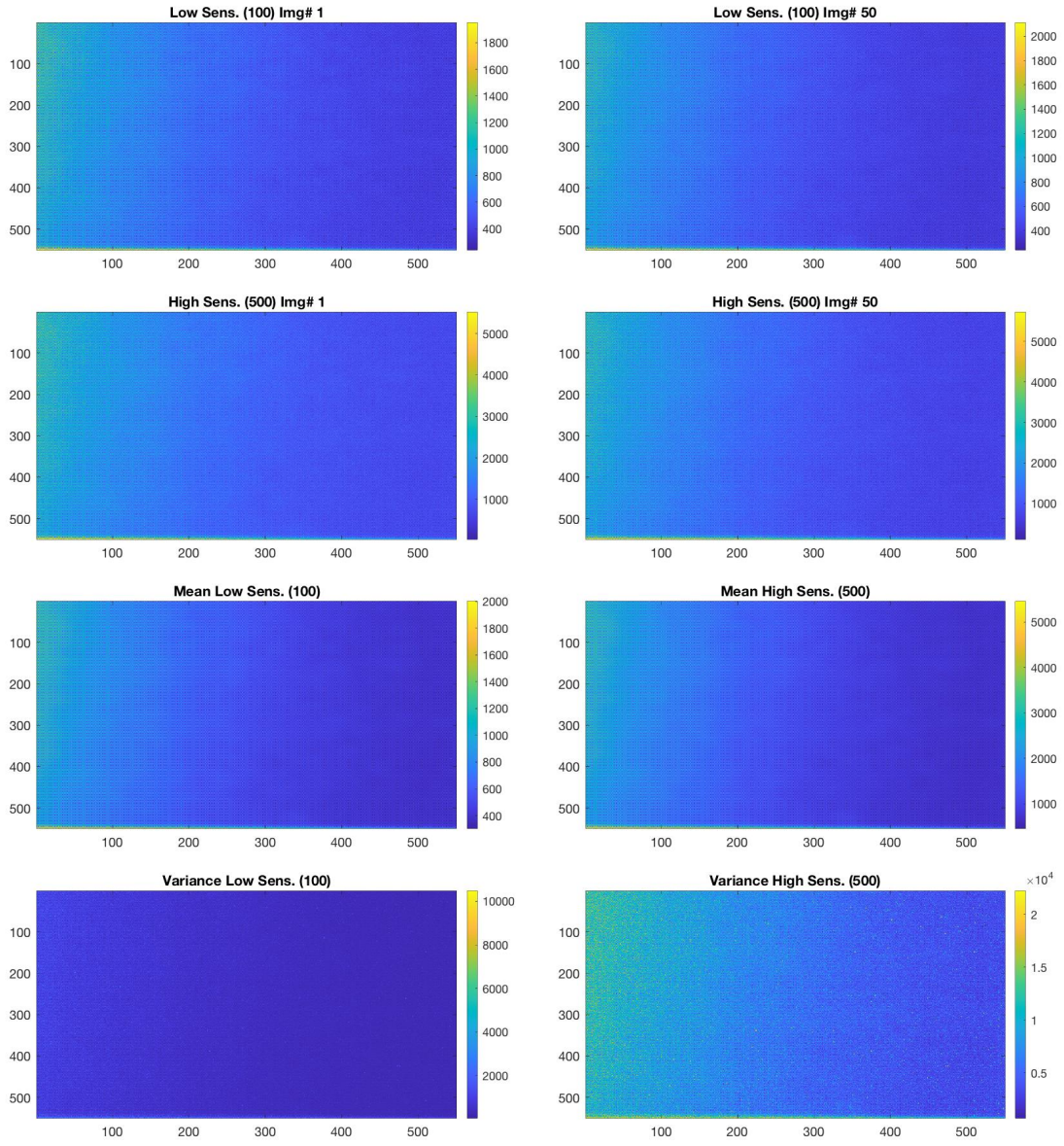

Figure 4 – Low/High sensitivity maps for original images, along with mean & variance

While the top 4 plots show the original images, the bottom 4 show the mean & variance per sensitivity. After this, in order to calculate the noise variance, I calculate the variance as a function of the mean. This begins on line 65 in my code, and I've plotted it below in Figure 5.

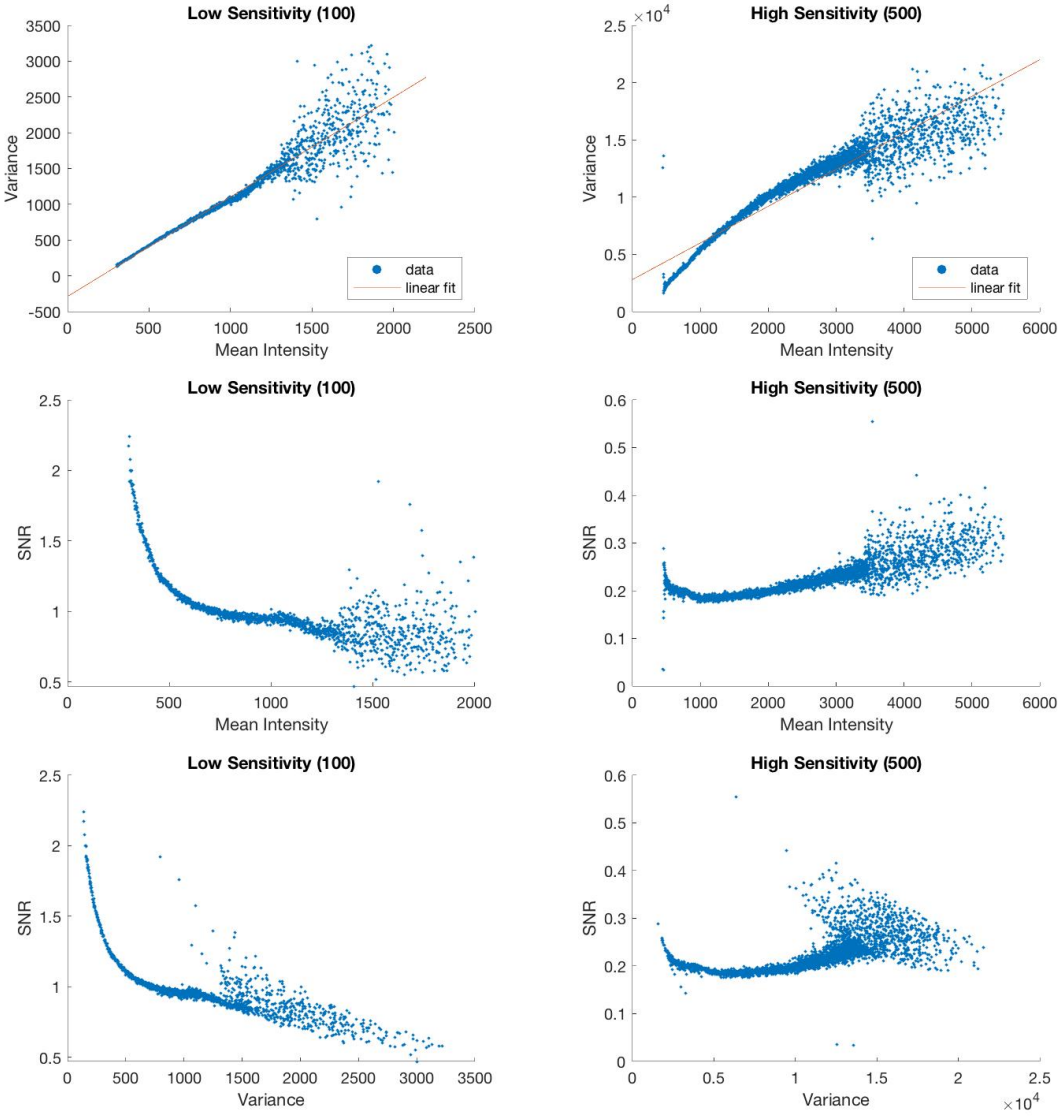

Figure 5 – 3 graphs per sensitivity (mean vs variance; mean vs SNR; variance vs SNR)

As you can see from the top 2 graphs, the recorded data (mean intensity) fits along a linear fit to variance, which is to be expected of normal distributions. As sensitivity increases, the image is might brighter and we can see less clarity in the exact fit. Finally, we need to calculate the Signal-to-Noise ratio, which is a ratio of the relative strength of a signal vs. interference. This is simply the mean over the variance. Across this array, we have now calculated the SNR, and can plot it against our mean intensities (middle 2 graphs) and against our variance (bottom 2 graphs). The bottom two, left two graphs shows us exactly the curve I expected. As signal intensity increases, our variance slows down, and thus we have a shape that looks a lot like a hook. The heavy noise at the end makes sense too as the images approach the darkest end, it is harder to read the image and the calculations are more scattered. This may have been an effect of the picture that we took, or the selection range of my cropping. The maximum SNR to be achieved by my camera seems to be around 2.5. I feel like this is

wrong, as most of the tablets should be in a similar range, and Florian's example showed a much higher SNR score, but these are results of the photos I took. Read noise is generated by the electronic components of the camera itself as the signal charge per pixel is transferred to the camera. ADC is a code transition noise that represents the uncertainty of the analog voltage as it transitions. Gain is the noise generated by the variance in the amplification of the signal. I believe the Signal-to-Noise ratio represents the ratio of the mean of these 3 to its standard deviation. Even the equation shows the variance as the summation of the mean per pixel times gain, gain squared times variance of read noise, and the variance of the ADC noise.

Florian's Update:

Of course, the method for calculating the linear fits for our data were provided to us by the use of the polyfit() function. For our **low sensitivity**, the mean by variance linear fit was of the equation:

$$y_{low} = 1.3916x - 288.1235$$

(we were told not to worry about the negative y-intercept)

and for our **high sensitivity**, the mean by variance linear fit was of the equation:

$$y_{high} = 3.2003x + 2799.55$$

Now as for the ADC and Read noises, to calculate all noise variance, we use the equation:

$$\sigma_i^2 = \mu_i * g + \sigma_{read}^2 * g^2 + \sigma_{ADC}^2$$

Taking the low sensitivity linear fit, y_{low}, the camera gain ***g***, is the slope or **1.3916**.
As for the high sensitivity linear fit, y_{high}, the camera gain ***g***, is the slope or **3.2003**.

Now I did attempt to calculate the ADC/read variances various ways, and got different results per each until I found one that made sense. You see the camera gain, *g*, is the dark noise divided by the # of electrons (aka. conversion factor between photo-electrons and gray-levels (GL/e^-)). I substituted values from y_{low} into the above variance equation at a random mean intensity of 2122, such as the following, and I'm left with:

$$2665 = 2122 * 1.3916 + \sigma_{read}^2 * 1.93655056 + \sigma_{ADC}^2$$

and this doesn't really help me. Even if I were to calculate the read variance by knowing that the i^{th} pixel is measured in photo-electrons/sec and solve for photo-electrons per pixel, where our low exposure was set to 100ms (0.1 sec), we could multiple each intensity value by 10 (or 50 for high being set to exposure 500ms), but this method's variance per pixel ends up being astronomical in size. I ended up using the polyfit's slope and intercept to determine the $\sigma_{read}^2 = 371.7601$, and $\sigma_{ADC}^2 = -1{,}008.1$. I plugged this back into the above equation and it's near perfect:

$$2665 = 2122 * 1.3916 + 371.7601 * 1.93655056 - 1{,}008.1 \rightarrow$$
$$2665 = 2{,}952.9752 + 719.9322 - 1{,}008.1 \rightarrow$$
$$\mathbf{2665 \cong 2{,}664.8}$$

I was confused for so long because the slides say: "Estimate the camera gain (DN/electron)" and that gain is GL/e^-, and this turns out to be ***2.712***. Anyway, I finally got it ^. My code is also in MATLAB.