CCD LABORATORY WRITE-UP

B. CONNOR MCCLELLAN, JULIEANNA BACON, JOHN MICHAEL DELLA COSTA
University of Florida

ABSTRACT

This lab report outlines the procedures and results of analyzing dark-frame and flat-frame images on a CCD to experimentally measure read noise, dark current, gain, and linearity. I reduce my data in Python 2.7. The objective of the lab is to understand the basic operating principles of a CCD. A large component of this experiment and the following analysis focuses on quantifying error. It is crucial to understand the misgivings of the instrument one is working with before pointing it at actual targets; then, not only can one optimally configure the instrument to acquire the best data possible, but also its error will be well-defined and easily corrected for.

1. EXPERIMENT SETUP

Our setup consists of a "dark room" made from two large wooden platforms standing on end, with a heavy, opaque cloth laid on top. At the far end, a piece of flat white paper is suspended vertically on a metal mount. The SBIG CCD, fitted with a Nikon AF NIKKOR 70-300mm lens with a 3-D printed adapter, sits in the middle of the rig with the field of view centered on and perpendicular to the sheet of paper. A large laptop is placed at the opposite end, with the screen casting light from just above the CCD toward the sheet of paper. When we take data, the opaque cloth is set over the entire rig and remains in the same place for the duration of imaging, to preserve consistency in what small amount of light manages to leak through from the outside environment. A small flap on one side is lifted to access the camera, cover the lens, and alter the laptop screen brightness as needed for the experiment. A second laptop is wired to the CCD, and operated from outside the cloth. This laptop, running CCDOps, controls the exposure time, number of images acquired, filters used, and type of exposure.

2. MEASURING THE READ NOISE

2.1. *Objective*

In this section, I quantify the read noise of the detector using 18 0.1-second dark exposures. I then use the same method to find the read noise for increasing exposure times, using 6 1s images, 6 10s images, and 3 100s images. The overall goal of this section is to, first, measure the read noise of the detector accurately, and second, prove that exposure time has no effect on the read noise of the detector. Since read noise is only introduced when the CCD is read out, it only happens once per exposure and should be constant no matter how long the exposure is. To prove this, a linear fit of the read noises at each exposure time should have no appreciable slope; that is, there should be very little correlation between exposure time and read noise.

2.2. *Method*

I start by loading in the 18 0.1-second images into a 3-dimensional numpy array. Two of the axes are the dimensions of the image, and the third axis is the dimension along which the 18 images are stacked. I then calculate the RMS of the 3-dimensional array along the third axis, using

$$RMS = \sqrt{|\langle x^2 \rangle - \langle x \rangle^2|} \tag{1}$$

This returns a new 2-dimensional array containing the RMS of each pixel in that pixel's location. As stated in the lab manual, the median or mean of this array should be the average read noise of the CCD. The associated uncertainty is

$$RMSE = \frac{\sigma}{\sqrt{n_{pix}}} \tag{2}$$

where σ is the RMS of the array and n_{pix} is the number of pixels in the CCD. For any set of images in this lab, I will use this definition to calculate the uncertainty of the average of the associated array of data values.

An alternate method of measuring the read noise involves plotting the data number (DN) of all the pixels on a histogram, and fitting a Gaussian to the histogram (Figure 1). The central peak of the histogram returns the average read noise of the CCD, and the Gaussian fit's σ describes the statistical spread, which can then be converted to an uncertainty using equation (2).

2.3. *Results*

2.3.1. *Quantification of Read Noise*

Table 1. Read Noise, in DN

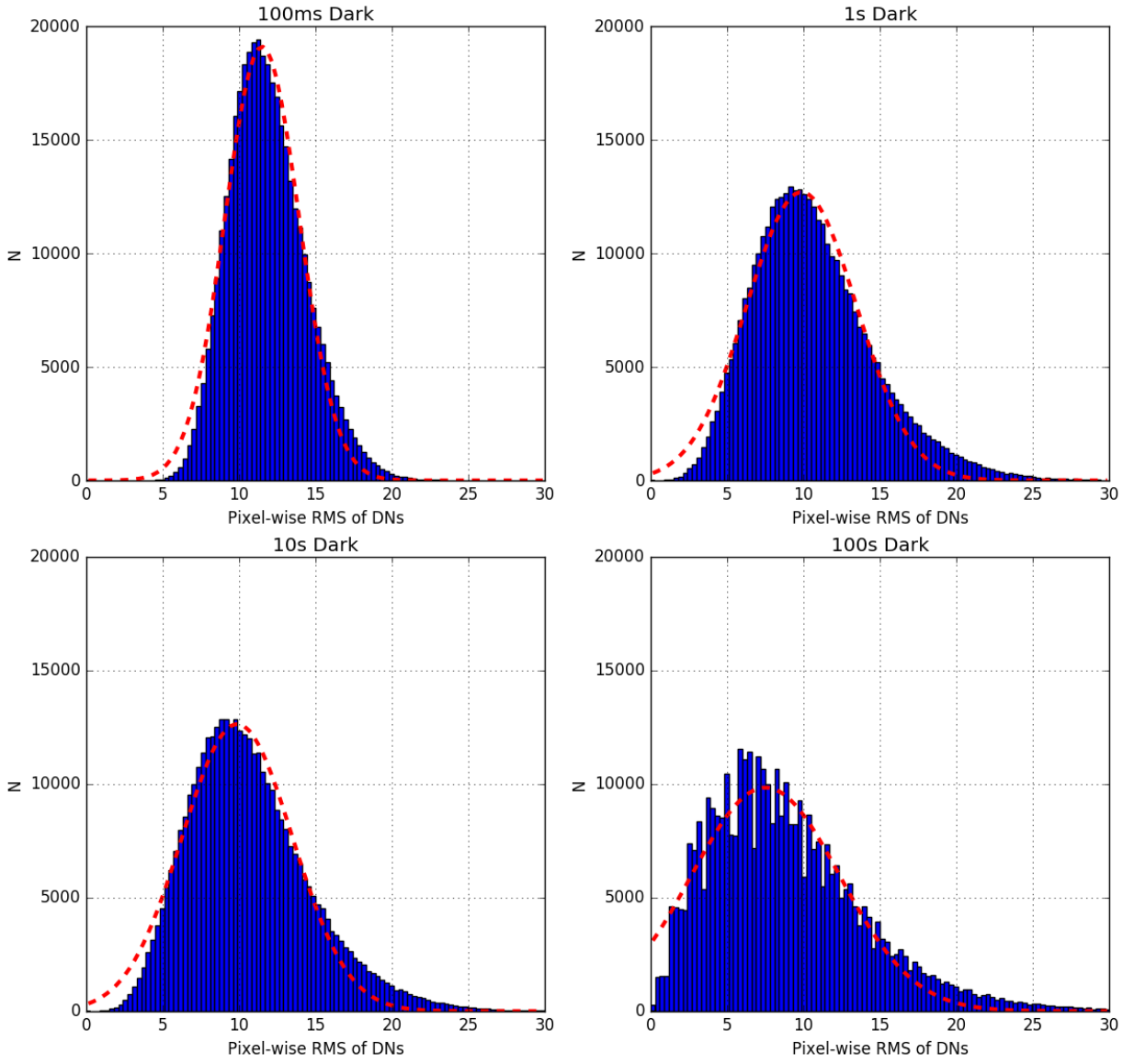
Measurement	0.1 s		1 s		10 s		100 s	
	Average	Gaussian	Average	Gaussian	Average	Gaussian	Average	Gaussian
Mean	11.832	11.497	10.650	9.871	10.657	9.874	8.982	7.512
RMS	2.512	2.424	3.996	3.587	4.008	3.606	5.618	4.848
RMSE	4.021×10^{-3}	3.881×10^{-3}	6.397×10^{-3}	5.743×10^{-3}	6.417×10^{-3}	5.773×10^{-3}	8.995×10^{-3}	7.761×10^{-3}

Average Read Noise of All Exposures: 10.12 ± 0.49 DN

The values under "Average" in Table 1 are acquired using np.mean, np.median, and my function for calculating RMS and RMSE. They represent the mean, median, and RMS of the stacked data from the 18 0.1-second exposures. The values under "Gaussian" are the returned parameters of the Gaussian fit of the distribution, with RMS = σ of the curve, and RMSE calculated according to Eq (2). They are indeed similar, though the Gaussian Fit values are smaller across the board. Since the distribution of DNs for these images is not Gaussian, but in fact Poisson, the fit looks noticeably skewed (see Figure 1). The Gaussian lies just right of the true peak of the histogram, but what really throws the measure of central tendency off is the bins on the right-hand side of the histogram. The bin values are higher on the right side of the peak than the left, so both the raw mean and median calculations get thrown off by the unequal weight. The Gaussian, which is roughly

centered at the histogram's peak, does not suffer the same fate, and as such has a lower mean and σ.

Figure 1. Read Noise Histograms and Gaussian Fits



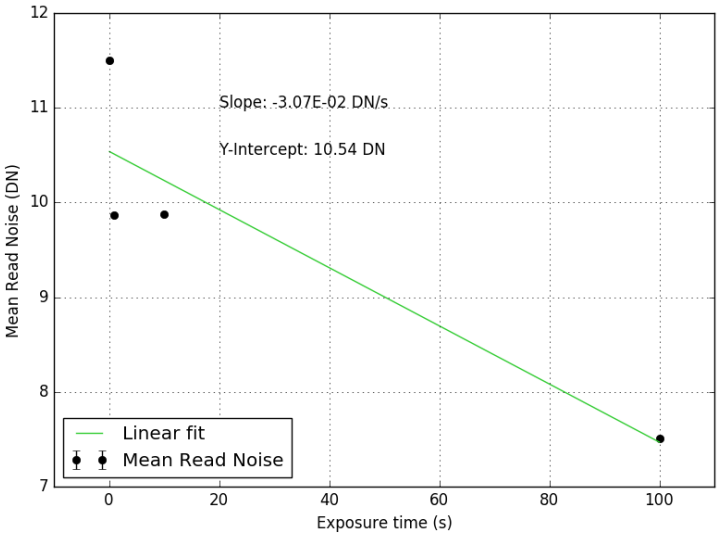
2.3.2. *Read Noise Dependence on Time*

Read noise is not a function of exposure time. Plotting the peaks of the Gaussian fits in the previous section against their corresponding exposure times yields a linear fit with a slope close to 0. This is expected, since the read noise should not increase or decrease appreciably as exposure time increases. This is shown graphically in Figure 2, in which the fit has a slope of order 10^{-2} and the data points themselves don't seem to have any sort of trend. In fact, the only trend the linear fit was able to extract is that read noise actually *decreases* slightly with time, which doesn't make

much physical sense. So, conclusively, read noise is not a function of time, since the read noise doesn't increase or decrease as exposure time increases.

This confirms the answer to another question: is there a light leak in the camera? If light was leaking into the CCD, though the shutter was closed and the lens cap was on, then the CCD would be collecting photons as the exposure was being taken. The longer the exposure, the more photons the CCD would collect. Thus, with a light leak, longer dark frames would have higher counts per pixel than shorter ones, and the fitted line in Figure 2 would have a positive slope (increasing DN with time). Since this is not the case, I'm pleased to report there is not a light leak in our setup.

Figure 2. Mean Read Noise v. Exposure Time



Note: Error bars are plotted on each data point, but they are too small to see.

2.3. *Summary*

Examination of the visual data in this section deftly confirms the theory behind read noise. Noise generated by a predictable, one-time electronic procedure should be consistent each time the CCD is read out, and, judging by the results of this section of the lab, it is. Two different statistical methods yield highly similar results for the mean, RMS, and RMSE of the DNs for each of 4 exposure times. This shows that, with reasonable accuracy, read noise can be approximated as either the mean of the RMS of all the pixels, or as the peak of a Gaussian fit of the pixel-wise RMS.

3. MEASURING THE DARK CURRENT

3.1. *Objective*

I measure dark current of the detector by taking a series of longer and longer dark exposures, subtracting the bias from each, and calculating the change in DN with time (dDN/dt). The purpose of measuring the dark current is to find how many thermally-generated electrons accumulate in the CCD before it is read out. It can be mitigated by effective cooling, but not eliminated entirely, and therefore it's important to be able to measure its impact on one's images.

3.2. *Method*

I begin by measuring my baseline, taking a 0.1-second dark exposure. I then increase the exposure time by a factor of 5, taking one image per exposure time. The exposure times I collected were 1s, 5s, 25s, 125s, and 600s. This is a wide enough range to ensure that even small slopes are noticeable, and not plagued by error. A bias image is taken every other exposure in order to ensure that the offset doesn't drift. As a precaution, I examine the bias level in-between each image. The average bias is consistent at 982 DN, with an RMS of 18.7, for all 5 bias images. It is safe to say that the electronic offset did not drift during this part of the experiment. A median-combined master bias is created from these 5 images, and subtracted from the longer exposures. Since the dark current is now isolated, it is possible to measure dDN/dt. This calculation can be performed

by hand, using the rise over run between each data point. These results are shown in Table 2, alongside the more effective way of fitting a line to the Gaussian peaks of each image's pixel value distribution, with DN on the y-axis and exposure time on the x-axis. The results of this plot are shown in Figure 4. The uncertainty in the calculated slope is returned from the square root of the first coefficient in the covariance matrix—this is dependent on the distance of each of the data points from the fitted line, and therefore serves as an effective measure of the uncertainty in the slope.

3.3. *Results*

Table 2. Dark Current

Exposure Time	Mean DN (in DN)	dDN/dt (in DN/s)	Gaussian Mean DN (in DN)	Gaussian Stddev (in DN)	Gaussian Uncertainty (in DN/pix)
1 s	-0.222	--	0.221	12.903	2.07×10^{-2}
5 s	0.517	1.85×10^{-1}	0.928	12.911	2.07×10^{-2}
25 s	1.543	5.13×10^{-2}	1.663	13.049	2.09×10^{-2}
125 s	8.363	6.82×10^{-2}	6.897	13.185	2.11×10^{-2}
600 s	38.165	6.27×10^{-2}	29.284	13.904	2.22×10^{-2}

Note: "Mean DN" was calculated using numpy.mean. Columns labeled "Gaussian" are values returned by the Gaussian fit. Gaussian Uncertainty is calculated using Eq. (2).

Gaussian dDN/dt: $4.80 \times 10^{-2}\ DN/s$ Uncertainty in dDN/dt: $1.06 \times 10^{-3}\ DN/s$

Figure 3. Dark Current Histograms (Exposure times in filenames)

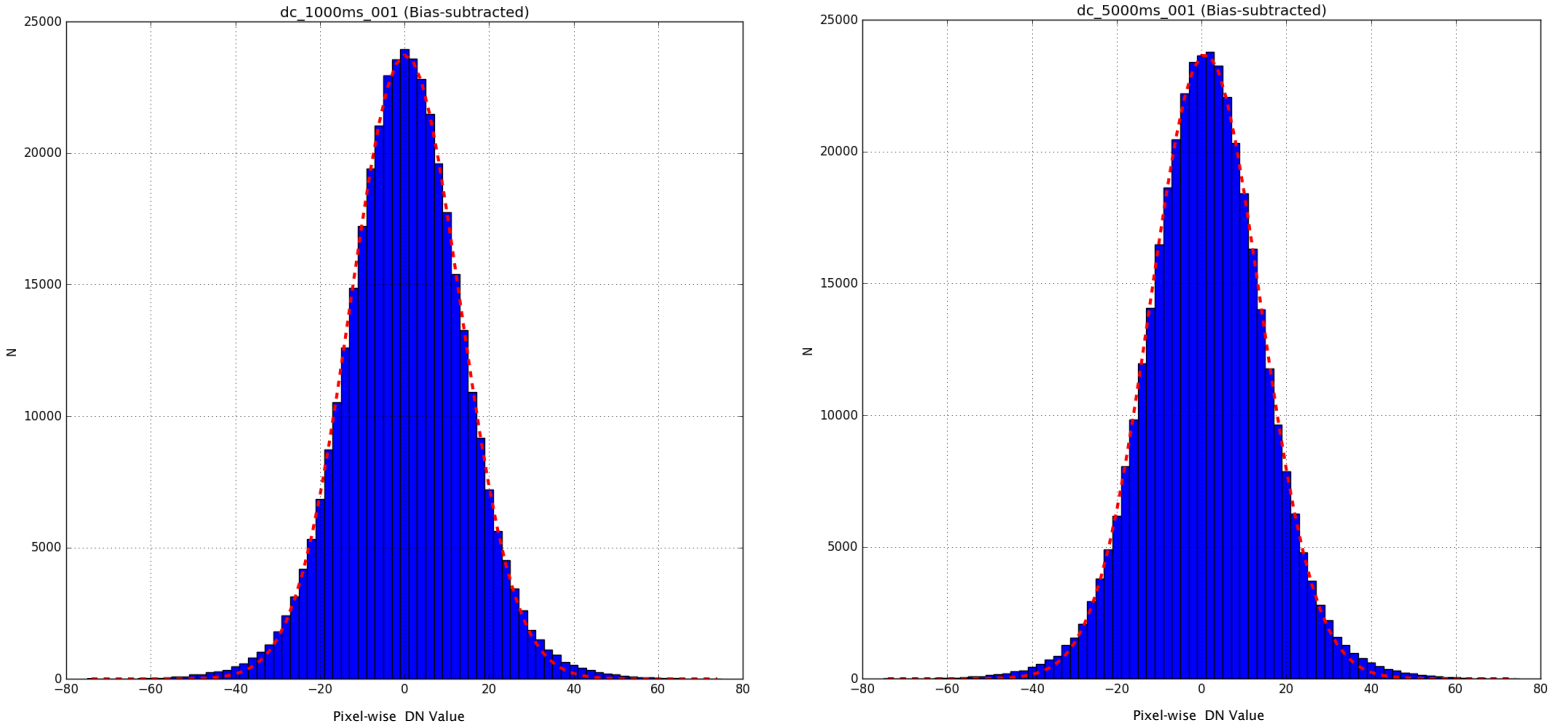
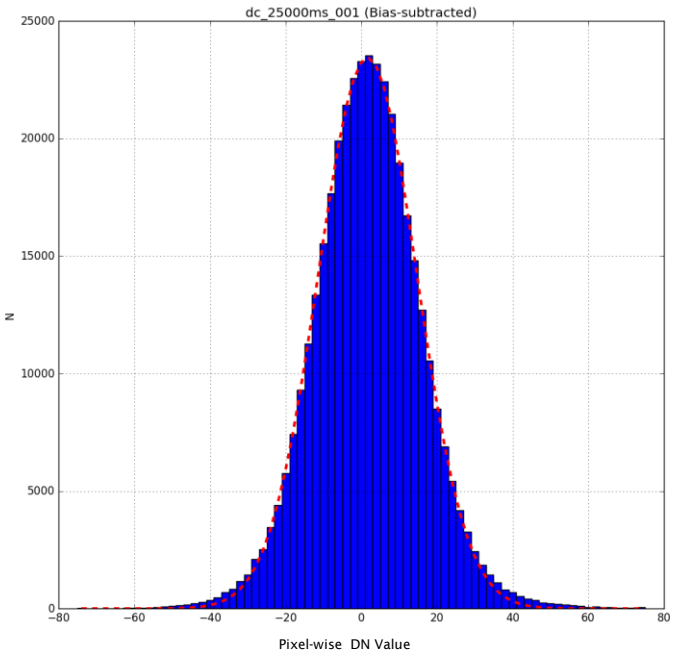
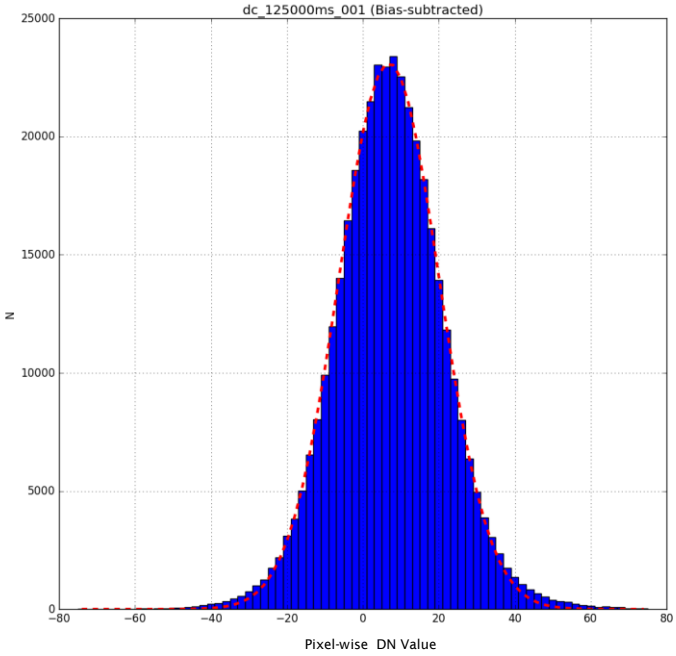


Figure continues on next page


dc_25000ms_001 (Bias-subtracted)
N
Pixel-wise DN Value


dc_125000ms_001 (Bias-subtracted)
N
Pixel-wise DN Value

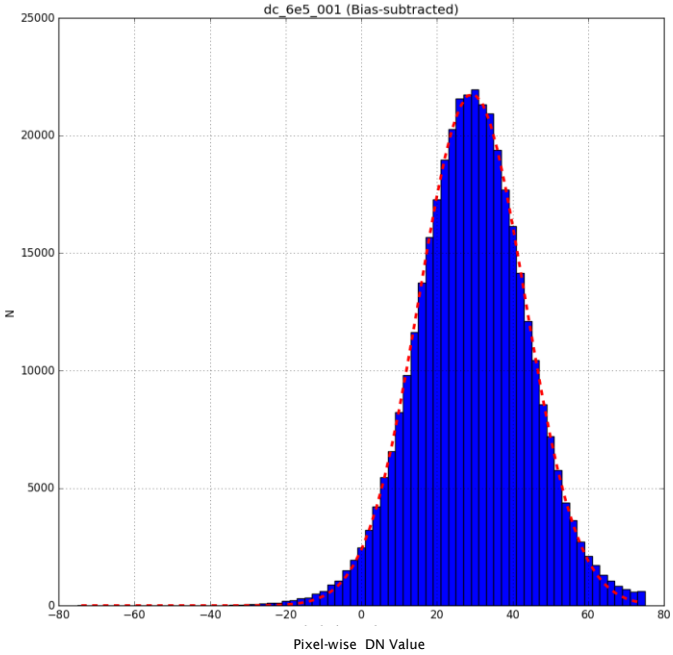
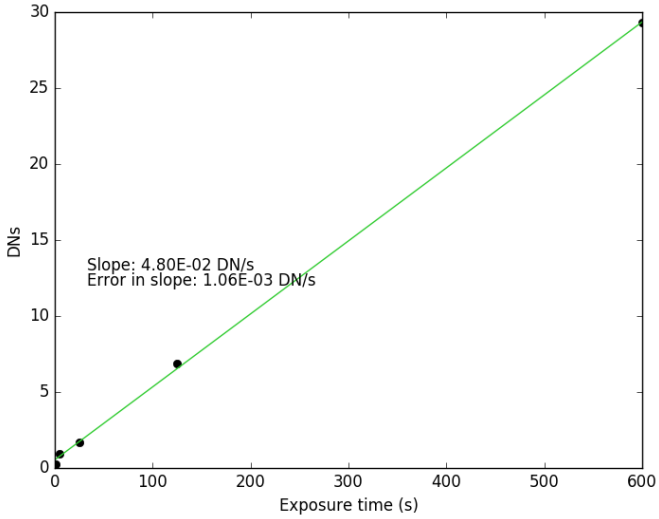

dc_6e5_001 (Bias-subtracted)
N
Pixel-wise DN Value

Figure 4. Mean DN v. Exposure Time



Note: Error bars are plotted on each data point, but they are too small to see.

The most important result from this section is certainly the slope measurement of $4.80 \times 10^{-2}\ DN/s$ and its corresponding uncertainty, $1.06 \times 10^{-3}\ DN/s$. Written a bit more intuitively, this looks like 0.048 ± .00106 DN/s. This value is a solid quantification of the dark current, dDN/dt, and can be extrapolated indefinitely to show the effect of thermal noise with increasing time.

It is important to note from our log that the temperature of the CCD was between -9.8 º C and -10.2 º C for the duration of this part of the lab. The temperature is pleasantly low, and decently stable throughout, leading to a low dark current with very good predictability.

3.4. *Summary*

The results for this section effectively measure the dark current and illustrate its linearity. Tracking the changes in the peak of the Gaussian fit creates a nice line, whose slope represents the change in thermal DNs with time. It is interesting to note that for the 1 second dark, many of the recorded DNs were negative. This occurs because, at such short exposure times, the variability in the bias level overlaps with the variability in the actual image. Thus, when the two are subtracted, some values are negative. This quickly vanishes as the exposure times increase.

With adequate cooling systems, the dark current can be reduced significantly such that it affects only very long exposures by a few DNs. Even then, it is possible to correct for the average dDN/dt, which is constant, in post-processing, but the uncertainty in dark current cannot be fixed. Dark current is typically corrected for by taking dark frames of the same length as the science image, and then subtracting them. This removes the dark current's constant linear offset, while preserving charges not generated by heat.

4. GAIN

4.1. *Objective*

The gain is an essential quantity to know for a CCD—even so, it is rarely accurately measured by CCDOps. For those occasions when I grow doubtful of the reported gain of my CCD camera, it is useful to have an empirical method for measuring it. Gain is the conversion from DNs, or counts, to number of electrons. It represents the factor by which the CCD fails in accurately counting electrons—it might register 1 electron (1 count) when actually 2 have gone into the charge well. Thus, the gain is a corrective factor that adjusts for the inability of the instrument to count properly. An accurate measure of the gain is incredibly important for performing photometry, since it's a big deal if your detector reports 0.3 or 0.5 ADU per electron when you're trying to collect a precise amount of light.

4.2. *Method*

The flat frames for this part of the lab were acquired with our setup described in section 1, with all the lights off in the room. We had our peers guard the door to ensure nobody unleashed the blaze of light from the hallway, and we vigilantly policed phones, laptops, and computer monitors inside the lab. Another group found that a single desktop monitor's screensaver could throw off their measurement for this section. We set the laptop screen to a light green color, and used the clear filter to reach DNs of ~55,000 for a 100s exposure. This, admittedly, took a very long time to get perfectly right. As soon as we had acquired all the images we needed in our controlled environment, I begin data reduction by loading in the series of flat frames. There are three images each of exposure times 0.1s, 0.5s, 1s, 3s, 10s, 30s, 60s, and 120s. Taking the mean of each of these along axis 0, the axis along which the images are stacked, yields 8 2D combined-image arrays. Taking the mean of all the pixels in a single image gives me an array of the mean signal in that image, and taking the mean of the square minus the square of the mean of all the pixels in a single image gives me an array of the variance in that image. If I take the mean value of each of these arrays for a single image, I get a mean signal and a mean variance. This comprises a single data point, and doing this for every exposure time adds data points to the Variance v. Signal plot. Their corresponding uncertainties are calculated by taking the RMS of the "variance" and "signal" arrays and dividing by the square root of the number of pixels, $\sqrt{390150}$. I then use numpy.polyfit to fit the points to a line, returning the slope and its error as in section 3.2. Since the shot noise (also RMS) of the image goes as $G\ N_e^{\frac{1}{2}}$, that means Variance = $(RMS)^2 = G^2\ N_e$. Since Signal = $G\ N_e$, the slope of the Variance v. Signal must be G, the gain of the detector. This is a useful relation to have on hand, since it enables one to find the gain using nothing but image data.

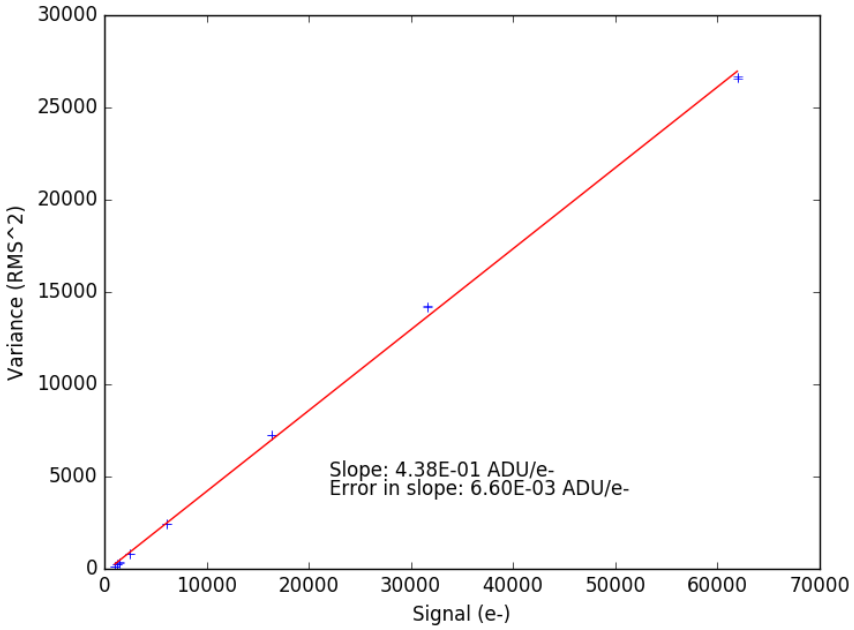
4.3. *Results*

Table 3. Gain

Exposure Time	Variance (ADU^2/e-)	Uncertainty in Variance (ADU^2/e-)	Median Signal (DN)	Uncertainty in Signal (DN)	d(Variance)/d(Signal) (ADU/e-)
0.1 s	130.895	0.237	1021.540	0.025	--
0.5 s	224.919	0.380	1217.387	0.024	0.480
1 s	343.435	0.563	1469.541	0.025	0.470
3 s	801.274	1.287	2491.064	0.045	0.448
10 s	2425.975	5.481	6071.936	0.154	0.454
30 s	7246.191	11.551	16329.596	0.480	0.470
60 s	14213.668	22.879	31561.724	0.967	0.457
120 s	26614.934	42.626	61929.823	1.889	0.408

Gain: 0.4384 ADU/e- Uncertainty in gain: 0.006602 ADU/e-

Figure 5. Median Variance v. Median Signal



Note: No data points, only error bars are plotted here. There is no error in the x-direction; any horizontal blue line you might see is a result of the "caps" on the ends of the *vertical* error bars.

It took a bit of experimentation to get the results for this section. My first question was whether or not to subtract darks from the flat images I was using to calculate my gain. Since read noise is constant, it shouldn't matter when the slope of the linear fit is calculated. Additionally, I performed a calculation with and without dark subtraction, and determined that the gain set by the first two points on the curve is identical in either case. Thus, these flats need not be dark-subtracted to calculate the slope of the line.

The overall result appears to have decent reliability, since the fit holds over a wide range of signals even when the potential wells are mostly full. I should note that I did throw out the saturated data, since it was throwing off the fit of the line. Saturated data has no variance, and thus is not useful for this plot. But, overall, I feel that there were few if any systematic errors that could have thrown off my measurement for this section. This was accomplished by having such a well-controlled environment for taking images.

4.4. *Summary*

For the duration of this section of the lab, it was imperative to keep the brightness of the flat field exactly the same for all of our images. Otherwise, the signal and variance would be changing erratically, as opposed to linearly, with increasing exposure time. The whole goal is to have a predictable relationship between the three values, and that can only be achieved with an extremely controlled, isolated environment. In fact, the majority of our time in the lab room was spent building our set-up for this part of the lab. I'm certainly glad we spent as much time as we did, because the results for this section are consistent and sensible. The reported gain of our CCD by CCDops was 1.5 e-/ADU. Thus, A gain of ~0.438 ADU/e-, or 2.28 e-/ADU, is a pretty reasonable calculation for this CCD. This number varies depending on whether I plot the mean variance against the mean signal or the median variance against the median signal, and I found that I got a larger (and closer to reported) value for ADU/e- when I used mean. Using median, I get a gain of 0.304 ADU/e-. I suppose its possible that *this* is in fact the correct value, and perhaps I shouldn't be tweaking my own calculations based on what SBIG tells me is right.

5. LINEARITY

5.1. *Objective*

If CCDs were perfect instruments, a single photon would make contact with the detector, and the electronics would read 1 count from the resulting photoelectron. Assuming you have a constant light level, then taking a series of images with increasing exposure times should yield corresponding signals, counting exactly the number of photons that went through the optics in a given period of time. Long exposures would have the highest signals, and short exposures would have the lowest. The relationship between signal and exposure time would be perfectly linear.

Unfortunately this is never the case. As the potential wells fill up, they're less and less able to pick up additional electrons. There's a threshold where the CCD can no longer count 1 DN for 1 photoelectron, and instead begins flattening out as it stops collecting. This threshold isn't a single point of no return—rather, it's a range where one can see the slow decline of the CCD's ability to report the signal. The goal of this section is to measure that decline by plotting the signal against exposure time, collecting as many data points as possible to fill out a smooth curve. Understanding linearity and how it changes with light level can be a useful way to prevent saturation and loss of information. Knowing the location of the CCD's deviation from linearity allows an observer to keep their DNs in the happy region to the left.

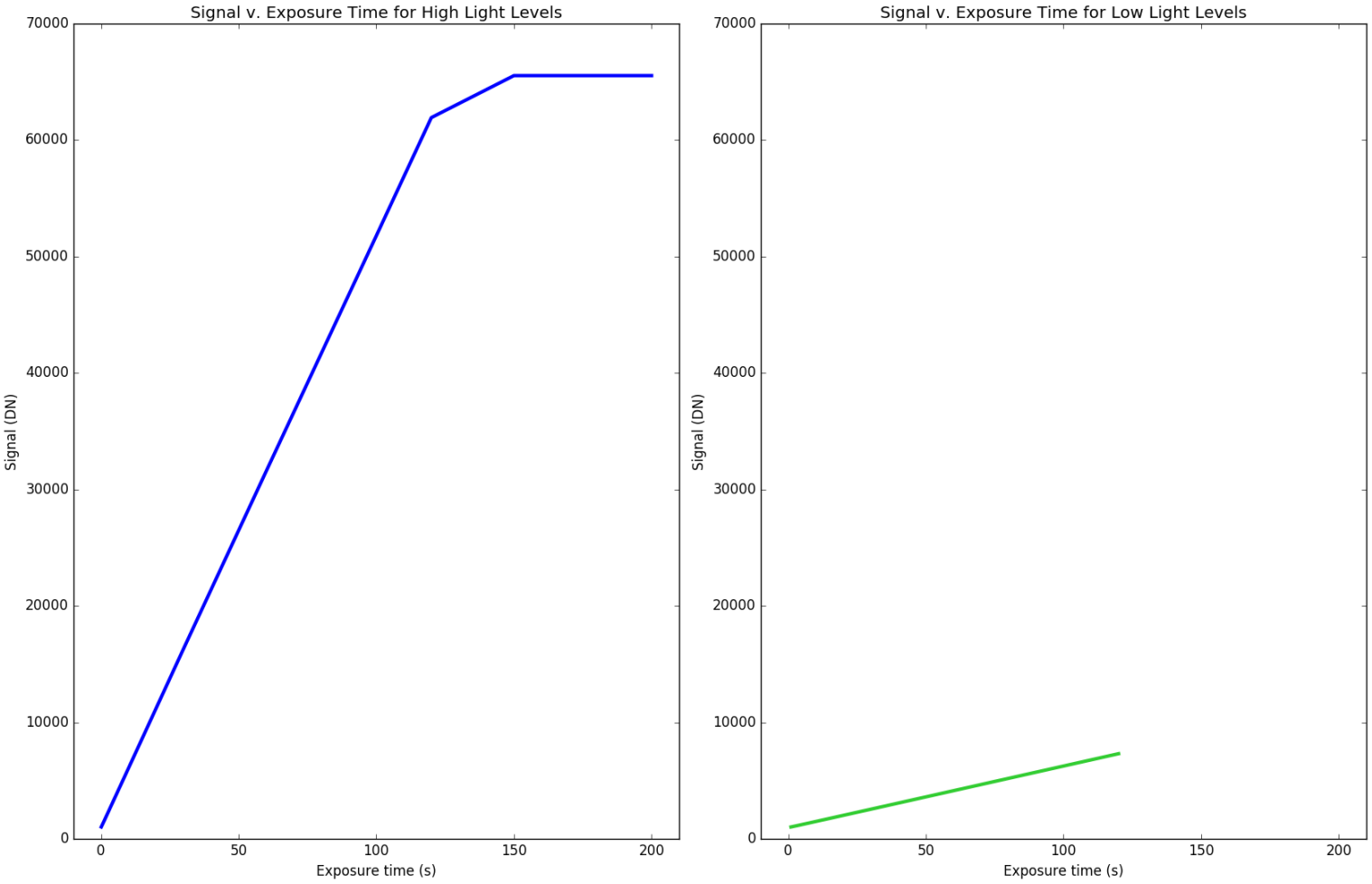
5.2. *Method*

This section requires a large number of flat images of varying exposure times, in both a high-light and low-light setting. For high-light, we use the CCD's clear filter and set the illumination laptop's screen to a dull red color, which we had experimentally determined to yield ~2,000 DN at 10s. For this section of the lab, we had to test many different combinations of screen color and image filter to get the perfect range of DN values for our chosen exposure times. We found that a dull red screen with the clear filter was the best fit, and gave us a range of values extending to just above 60,000 DN at our second-highest exposure time. With this setup, we take 12 sets of flats ranging from 0.1s to 200s in exposure time. Each set consists of three images, except for the sole 150s frame. Each set is mean combined along the image axis, and then the mean value and RMS of the resulting array are taken as the signal and uncertainty, respectively. These values are sorted into an array with their corresponding exposure times, and then plotted against each other. The resulting curve is shown in Figure 6.a. For low light levels, we used the CCD's blue filter with a forest-green colored laptop screen illuminating the sheet of white paper. With this setup, we take 7 sets of flats ranging from 1s to 120s in exposure time, each set consisting of 3 images. Using the same procedure as above, the signal and uncertainty are calculated and then plotted against the exposure time. The resulting curve for low-light levels is shown in Figure 6.b.

5.3. *Results*

Figure 6. Mean Signal v. Exposure time for High and Low Light Levels

a. **b.**



In retrospect, I very much wish we had taken more exposures in low light levels. However, it's not impossible to project what the rest of the curve would look like. Since the limit of the potential wells is nowhere near being reached in the low-light setting, signal v. exposure time should be linear. Thus, the green line above would continue in precisely the direction it's already going, until it eventually flattens out and reaches saturation way off the right side of the page. The two plots above have the same scale, so you can see relatively how long it would take to reach the value of 65535 if you were continuing along the slope of the low-light curve.

Additionally, note that these are not fits of the data points, but rather a line plot of the data points themselves. The linear sections of the graph are truly linear – if they had large errors or deviations, they would not be obscured by a fitted line.

Table 4. Light Curves

Exposure Time	High Light Levels		Low Light Levels	
	Mean Signal (DN)	Uncertainty in Mean Signal (DN)	Mean Signal (DN)	Uncertainty in Mean Signal (DN)
0.1 s	1021.540	0.02547	--	--
0.5 s	1217.387	0.02423	--	--
1 s	1469.541	0.02478	1018.861	0.02611
3 s	2491.064	0.04529	1119.586	0.02795
10 s	6071.936	0.15433	1486.798	0.03568
30 s	16329.596	0.47960	2541.535	0.06322
60 s	31561.724	0.96684	4130.139	0.10655
100 s	51777.235	1.60048	6249.362	0.16585
120 s	61929.823	1.88897	7309.297	0.19584
150 s	65535.000	0.00000	--	--
200 s	65535.000	0.00000	--	--

As expected, in the high light setting the data numbers increase linearly until they get close to the maximum saturation level. Then, the signal v. exposure time flattens out as it approaches its maximum value of 65535. In the low light setting, the data numbers increase linearly throughout, since the saturation level isn't anywhere near the highest data number in the range.

5.4. *Summary*

The linearity of the detector has been measured and plotted, confirming the shape of the theorized light curves. The response of the CCD is near-perfectly linear until the upper limit of the potential wells is reached—then things stop working well. It would be interesting to plot the exact shape of the falloff, using some kind of polynomial fit. Note that the behavior at the threshold isn't actually asymptotic, though it looks like it in the plot. For a lot of mathematical expressions, a peak value is approached but never actually reached. In the case of this experiment, the value of 65535 is certainly reached. For a future experiment, I would like to take many data points in the falloff range (since I know where it is now) to fill out that curve more precisely and find an equation that models it adequately. No doubt, a physical interpretation of whatever equation that was would point me to some interesting physics about the way electrons are repelled from an already-full potential well.

6. CONCLUSION

Property of CCD	Measured value	Uncertainty
Read Noise	10.12 DN	0.49 DN
Dark Current	0.04803 DN/s	0.00106 DN/s
Gain	0.4384 ADU/e-	0.00660 ADU/e-

These three numbers comprise the majority of the purpose of the experiment. Using just flat and dark frames, I have computed each of these values empirically using simple data reduction techniques in Python. The uncertainties for each are decently easy to calculate, and there appears to be little room for bad data to hide. If something was wrong with any of the plots used to measure these properties, it would be glaringly obvious. This, combined with the fact that I already made a few glaringly obvious errors *and corrected them*, gives me a great deal of confidence in the accuracy of these measurements. Each makes physical sense; the read noise is about 10 counts, which is a reasonable estimate for this CCD. The dark current is very small and increases with time, which makes sense because our CCD was well-cooled and in a stable environment. The gain is slightly higher than the nominal gain in the FITS header, but this is to be expected. A gain of ~2 e-/ADU is a very reasonable number.

Overall, I am satisfied with our experiment and measurements. Our environment was well controlled, and there are no detectable systematic errors present in our data. If there had been a light leak, for instance, our read noise would have ended up being time-dependent. If the light level was not well-controlled, the linear fit for our gain measurement wouldn't fit the data points very well and would have high uncertainty. Neither of these are the case, since our CCD was isolated from external light and the room's brightness was carefully monitored.

While looking at the manual calculations for dDN/dt and d(Variance)/d(Signal), I saw that the *slope* for each of these changes with time. Theoretically, both should be constants, but perhaps there is a proper gain and dark current for short exposure times that is different than the same value for longer exposure times. A future experiment could look into how the dark current and gain change with very long exposure times, compared with very short ones. No doubt, confounding variables would be encountered at either end (such as the read noise barrier at very short exposure times), and the only useful data would be the linear region in the middle. This is most likely why average dark current and gain should be constant— this is true for the good data *within the linear region*.

For a future experiment, I would like to fill out the light curves in section 5 more, and examine the exact shape of the falloff as the potential wells approach saturation. In fact, I would like to repeat the experiment with many more exposure times to add data points to every plot. This would increase the credibility of my measurements of the properties of the CCD, and ensure that they apply for a wide range of exposure times.

```
# READ NOISE DATA REDUCTION
# AUTHOR: B. CONNOR MCCLELLAN


from astropy.io import fits
import numpy as np
import matplotlib.pyplot as plt
from astropy.modeling import models, fitting


def rms(x):
    return (np.absolute(np.mean(x**2) - (np.mean(x))**2))**0.5


def construct_data_array(filename_prefix, n_images):
    da = []
    for i in range(n_images):
        if i+1 < 10:
            j = '00'+str(i+1)
        else:
            j = '0'+str(i+1)
        da.append(fits.open(filename_prefix+j+'.FIT')[0].data)
    da = np.array(da, dtype=float)
    shape = np.shape(da)
    n_pixels = shape[1] * shape[2]
    return da, n_pixels


# READ NOISE 1, 2: Calculate RMS array

data_100ms = construct_data_array('dark_100ms_', 18)
data_1000ms = construct_data_array('dark_1000ms_', 6)
data_1e4ms = construct_data_array('dark_1e4ms_', 6)
data_1e5ms = construct_data_array('dark_1e5ms_', 3)

data_list = [data_100ms, data_1000ms, data_1e4ms, data_1e5ms]
final_RMS_array = []
for i in range(4):
    average_of_square = np.mean(data_list[i][0]**2, axis=0)
    square_of_average = (np.mean(data_list[i][0], axis=0))**2
    variance = average_of_square - square_of_average
    RMS = np.absolute(variance)**0.5

    # READ NOISE 3: Find mean, median, and RMS of RMS array
    npix = data_list[i][1]
    print npix
    print "i = ", i
    print "Mean: ", np.mean(RMS)
    print "Median: ", np.median(RMS)
    print "RMS: ", rms(RMS)
    print "Read noise uncertainty: ", rms(RMS)/np.sqrt(npix), " per pixel"

    # READ NOISE 4: Make a histogram of RMS array values
    final_RMS_array.append(np.ndarray.flatten(RMS))


print "\nGauss Fit measurements:"
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(12, 12))

# PLOT #1
n, bins, patches = ax1.hist(final_RMS_array[0], 100, range=[0, 30])
bincenters = 0.5*(bins[1:]+bins[:-1])
```

```
g_init = models.Gaussian1D(amplitude=18000, mean=12, stddev=2.5)
fit_g = fitting.LevMarLSQFitter()
g = fit_g(g_init, bincenters, n)

mean1, stddev1 = g.mean[0], g.stddev[0]
print 'i = 1'
print "Mean: ", mean1
print "Stddev: ", stddev1

ax1.plot(bincenters, g(bincenters), 'r--', linewidth=3)
ax1.set_xlabel('Pixel-wise RMS of DNs')
ax1.set_ylabel('N')
ax1.set_xlim(0, 30)
ax1.set_ylim(0, 20000)
ax1.grid(True)
ax1.set_title('100ms Dark')


# PLOT #2
n, bins, patches = ax2.hist(final_RMS_array[1], 100, range=[0, 30])
bincenters = 0.5*(bins[1:]+bins[:-1])

g_init = models.Gaussian1D(amplitude=12500, mean=10.6, stddev=4)
fit_g = fitting.LevMarLSQFitter()
g = fit_g(g_init, bincenters, n)

mean2, stddev2 = g.mean[0], g.stddev[0]
print 'i = 2'
print "Mean: ", mean2
print "Stddev: ", stddev2

ax2.plot(bincenters, g(bincenters), 'r--', linewidth=3)
ax2.set_xlabel('Pixel-wise RMS of DNs')
ax2.set_ylabel('N')
ax2.set_xlim(0, 30)
ax2.set_ylim(0, 20000)
ax2.grid(True)
ax2.set_title('1s Dark')


# PLOT 3
n, bins, patches = ax3.hist(final_RMS_array[2], 100, range=[0, 30])
bincenters = 0.5*(bins[1:]+bins[:-1])

g_init = models.Gaussian1D(amplitude=12500, mean=10.6, stddev=4)
fit_g = fitting.LevMarLSQFitter()
g = fit_g(g_init, bincenters, n)

mean3, stddev3 = g.mean[0], g.stddev[0]
print 'i = 3'
print "Mean: ", mean3
print "Stddev: ", stddev3

ax3.plot(bincenters, g(bincenters), 'r--', linewidth=3)
ax3.set_xlabel('Pixel-wise RMS of DNs')
ax3.set_ylabel('N')
ax3.set_xlim(0, 30)
ax3.set_ylim(0, 20000)
ax3.grid(True)
ax3.set_title('10s Dark')
```

```
# PLOT 4
n, bins, patches = ax4.hist(final_RMS_array[3], 100, range=[0, 30])
bincenters = 0.5*(bins[1:]+bins[:-1])

g_init = models.Gaussian1D(amplitude=10000, mean=8.9, stddev=5.6)
fit_g = fitting.LevMarLSQFitter()
g = fit_g(g_init, bincenters, n)

mean4, stddev4 = g.mean[0], g.stddev[0]
print 'i = 4'
print "Mean: ", mean4
print "Stddev: ", stddev4

ax4.plot(bincenters, g(bincenters), 'r--', linewidth=3)
ax4.set_xlabel('Pixel-wise RMS of DNs')
ax4.set_ylabel('N')
ax4.set_xlim(0, 30)
ax4.set_ylim(0, 20000)
ax4.grid(True)
ax4.set_title('100s Dark')

plt.tight_layout()
plt.show()

exptime = [0.1, 1, 10, 100]
means = [mean1, mean2, mean3, mean4]
stddevs = [stddev1, stddev2, stddev3, stddev4]

err = []
for item in stddevs:
    err.append(stddevs[stddevs.index(item)]/390150.**0.5)
print err

l_init = models.Linear1D(slope=0, intercept=956)
fit_l = fitting.LinearLSQFitter()
l = fit_l(l_init, exptime, means)
slope, intercept = l.slope[0], l.intercept[0]
print "Slope, Intercept: ", slope, intercept

fig = plt.figure()
ax = fig.add_subplot(111)
ax.annotate('Slope: '+str("%.2E" % slope+' DN/s'), xy=(20, 11))
ax.annotate('Y-Intercept: '+str("%.2f" % intercept)+' DN', xy=(20, 10.5))
plt.errorbar(exptime, means, yerr=err, fmt='ko', ecolor='k', label='Mean Read Noise')
plt.plot(exptime, l(exptime), 'limegreen', label='Linear fit')
plt.xlabel('Exposure time (s)')
plt.ylabel('Mean Read Noise (DN)')
plt.xlim(-10, 110)
plt.tight_layout()
plt.legend(loc=3)
plt.grid(True)
plt.show()
```

```
# DARK CURRENT DATA REDUCTION
# AUTHOR: B. CONNOR MCCLELLAN

from astropy.io import fits
import numpy as np
import matplotlib.pyplot as plt
from astropy.modeling import models, fitting


def rms(x):
    return (np.absolute(np.mean(x**2) - (np.mean(x))**2))**0.5


def construct_data_array(filename_prefix, n_images):
    da = []
    for i in range(n_images):
        if i+1 < 10:
            j = '00'+str(i+1)
        else:
            j = '0'+str(i+1)
        da.append(fits.open(filename_prefix+j+'.FIT')[0].data)
    da = np.array(da, dtype=float)
    return da


def plot_hist_gauss(data, bins, amplitude, mean, stddev, title):
    print title
    fig, ax1 = plt.subplots(1, 1, figsize=(12, 12))
    n, bin_locations, patches = ax1.hist(data, bins, range=[-75, 75])
    bincenters = 0.5*(bin_locations[1:]+bin_locations[:-1])
    g_init = models.Gaussian1D(amplitude=amplitude, mean=mean, stddev=stddev)
    fit_g = fitting.LevMarLSQFitter()
    g = fit_g(g_init, bincenters, n)
    mean1, stddev1 = g.mean[0], g.stddev[0]
    print "Mean: ", mean1
    print "Stddev: ", stddev1
    ax1.plot(bincenters, g(bincenters), 'r--', linewidth=3)
    ax1.set_xlabel('Pixel-wise Dark Current')
    ax1.set_ylabel('N')
    ax1.grid(True)
    ax1.set_title(title)
    #plt.show()
    return mean1, stddev1


# BIAS LEVEL MEASUREMENT
bias = construct_data_array('bias_200ms_', 5)
for i in range(5):
    print "Bias: ", np.median(bias[i]), rms(bias[i])
master_bias = np.median(bias, axis=0)

prefix_list = ['dc_1000ms_', 'dc_5000ms_', 'dc_25000ms_', 'dc_125000ms_', 'dc_6e5_']
final_mean = []
final_stddev = []

for filename in prefix_list:
    i = prefix_list.index(filename)
    data = np.ndarray.flatten(construct_data_array(filename, 1)[0] - master_bias)
    result = plot_hist_gauss(data, 75, 45000, i, 13.6, filename+'001 (Bias-subtracted)')
    print "Numpy Mean: ", np.mean(data)
    final_mean.append(result[0])
    final_stddev.append(result[1]/390150.**0.5)
```

```
exptimes = [1, 5, 25, 125, 600]
print final_mean
print final_stddev

p, V = np.polyfit(exptimes, final_mean, 1, cov=True)
slope = p[0]
slope_err = np.sqrt(V[0][0])

print "Slope: ", slope
print "Error in slope: ", slope_err

yfit = []
for item in exptimes:
    y1 = p[0]*item + p[1]
    yfit.append(y1)

fig = plt.figure()
ax = plt.subplot(111)
plt.errorbar(exptimes, final_mean, yerr=final_stddev, fmt='ko', ecolor='k', elinewidth=4)
plt.plot(exptimes, yfit, 'limegreen', linewidth=1)
ax.annotate('Slope: '+str("%.2E" % slope)+' DN/s', xy=(33, 13))
ax.annotate("Error in slope: "+str("%.2E" % slope_err)+' DN/s', xy=(33, 12))
plt.xlabel('Exposure time (s)')
plt.ylabel('DNs')
plt.show()



# GAIN DATA REDUCTION
# AUTHOR: B. CONNOR MCCLELLAN

from astropy.io import fits
import numpy as np
import matplotlib.pyplot as plt


def rms(x):
    return (np.absolute(np.mean(x**2) - (np.mean(x))**2))**0.5


def construct_data_array(filename_prefix, n_images):
    da = []
    for i in range(n_images):
        if i+1 < 10:
            j = '00'+str(i+1)
        else:
            j = '0'+str(i+1)
        da.append(fits.open(filename_prefix+j+'.FIT')[0].data)
    da = np.array(da, dtype=float)
    return da

prefix_list = ['flat_100ms_', 'flat_500ms_', 'flat_1000ms_', 'flat_3000ms_', 'flat_1e4ms_',
'flat_3e4ms_',
               'flat_6e4ms_', 'flat_12e4ms_']
y = []
x = []
yerr = []
xerr = []

for prefix in prefix_list:
    # Calculate mean array
```

```
    data = construct_data_array(prefix, 3)
    mean_array = np.mean(data, axis=0)
    mean_err = rms(mean_array)/390150.**0.5

    # READ NOISE 1, 2: Calculate RMS array
    average_of_square = np.mean(data**2, axis=0)
    square_of_average = (np.mean(data, axis=0))**2
    variance = np.absolute(average_of_square - square_of_average)
    variance_err = rms(variance)/390150.**0.5

    y.append(np.mean(variance))
    yerr.append(variance_err)
    x.append(np.mean(mean_array))
    xerr.append(mean_err)

for i in range(len(prefix_list)):
    print i
    print "Variance: ", '%.3f' % y[i]
    print "Variance Error: ", '%.3f' % yerr[i]
    print "Mean Signal: ", '%.3f' % x[i]
    print "Signal error: ", '%.3f' % xerr[i]
    if i == 0:
        pass
    else:
        print "dV/dS: ", '%.3f' % ((y[i]-y[i-1])/(x[i]-x[i-1]))

p, V = np.polyfit(x, y, 1, cov=True)
print "\nSlope: ", p[0]
print "Error in slope: ", np.sqrt(V[0][0])

yfit = []
for item in x:
    y1 = p[0]*item + p[1]
    yfit.append(y1)

fig = plt.figure()
ax = fig.add_subplot(111)
plt.errorbar(x, y, xerr=xerr, yerr=yerr, fmt='none', ecolor='b')
ax.annotate('Slope: '+str("%.2E" % p[0])+' ADU/e-', xy=(22000, 5000))
ax.annotate("Error in slope: "+str("%.2E" % np.sqrt(V[0][0]))+' ADU/e-', xy=(22000, 4000))
plt.plot(x, yfit, 'r')
plt.xlabel('Signal (e-)')
plt.ylabel('Variance (RMS^2)')
plt.show()



# LINEARITY DATA REDUCTION
# AUTHOR: B. CONNOR MCCLELLAN


from astropy.io import fits
import numpy as np
import matplotlib.pyplot as plt


def rms(x):
    return (np.absolute(np.mean(x**2) - (np.mean(x))**2))**0.5


def construct_data_array(filename_prefix, n_images):
    da = []
```

```
    for i in range(n_images):
        if i+1 < 10:
            j = '00'+str(i+1)
        else:
            j = '0'+str(i+1)
        da.append(fits.open(filename_prefix+j+'.FIT')[0].data)
    da = np.array(da, dtype=float)
    return da


# CLEAR FILTER
flat_100ms = construct_data_array('flat_100ms_', 3)
flat_500ms = construct_data_array('flat_500ms_', 3)
flat_1000ms = construct_data_array('flat_1000ms_', 3)
flat_3000ms = construct_data_array('flat_3000ms_', 3)
flat_1e4ms = construct_data_array('flat_1e4ms_', 3)
flat_3e4ms = construct_data_array('flat_3e4ms_', 3)
flat_6e4ms = construct_data_array('flat_6e4ms_', 3)
flat_1e5ms = construct_data_array('flat_1e5ms_', 3)
flat_2e5ms = construct_data_array('flat_2e5ms_', 3)
flat_12e4ms = construct_data_array('flat_12e4ms_', 3)
flat_15e4ms = construct_data_array('flat_15e4ms_', 1)

data_list = [flat_100ms, flat_500ms, flat_1000ms, flat_3000ms, flat_1e4ms,
             flat_3e4ms, flat_6e4ms, flat_1e5ms, flat_12e4ms, flat_15e4ms, flat_2e5ms]

exptimes = [.1, .5, 1, 3, 10, 30, 60, 100, 120, 150, 200]
signal = []
signal_err = []
for i in range(len(data_list)):
    print i
    data = data_list[i]
    mean_signal = np.mean(np.mean(data, axis=0))
    mean_err = rms(np.mean(data, axis=0))/390150.**0.5
    print "Mean signal: ", "%.3f" % mean_signal
    print "Signal error: ", "%.5f" % mean_err
    signal.append(mean_signal)
    signal_err.append(mean_err)

print signal
print signal_err


# BLUE FILTER
flat_low_1000ms = construct_data_array('flat_low_1000ms_', 3)
flat_low_3000ms = construct_data_array('flat_low_3000ms_', 3)
flat_low_1e4ms = construct_data_array('flat_low_1e4ms_', 3)
flat_low_3e4ms = construct_data_array('flat_low_3e4ms_', 3)
flat_low_6e4ms = construct_data_array('flat_low_6e4ms_', 3)
flat_low_1e5ms = construct_data_array('flat_low_1e5ms_', 3)
flat_low_12e4ms = construct_data_array('flat_low_12e4ms_', 3)

data_list = [flat_low_1000ms, flat_low_3000ms, flat_low_1e4ms, flat_low_3e4ms,
             flat_low_6e4ms, flat_low_1e5ms, flat_low_12e4ms]

low_exptimes = [1, 3, 10, 30, 60, 100, 120]
low_signal = []
low_signal_err = []
for i in range(len(data_list)):
    print i
    data = data_list[i]
    low_mean_signal = np.mean(np.mean(data, axis=0))
```

```
    low_mean_err = rms(np.mean(data, axis=0))/390150.**0.5
    print "Low mean signal: ", "%.3f" % low_mean_signal
    print "Low signal error: ", "%.5f" % low_mean_err
    low_signal.append(low_mean_signal)
    low_signal_err.append(low_mean_err)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(17, 11))

ax1.plot(exptimes, signal, 'b', linewidth=3)
ax1.set_xlabel('Exposure time (s)')
ax1.set_xlim(-10, 210)
ax1.set_ylabel('Signal (DN)')
ax1.set_ylim(0, 70000)
ax1.set_title('Signal v. Exposure Time for High Light Levels')

ax2.plot(low_exptimes, low_signal, 'limegreen', linewidth=3)
ax2.set_xlabel('Exposure time (s)')
ax2.set_xlim(-10, 210)
ax2.set_ylabel('Signal (DN)')
ax2.set_ylim(0, 70000)
ax2.set_title('Signal v. Exposure Time for Low Light Levels')

plt.tight_layout()
plt.show()
```

OBSERVING LOG:

A| 16x 0.1 sec darks
T = -10°C , lens cap on in main room
dark_100ms_[i]
-6x 1 sec darks
T = -10°C "
dark_1000ms_[i]
-6x 10 sec
T = -10°C
dark_1e4ms_[i]
-3x 100 sec T = -10°C
dark_1e5ms_[i]

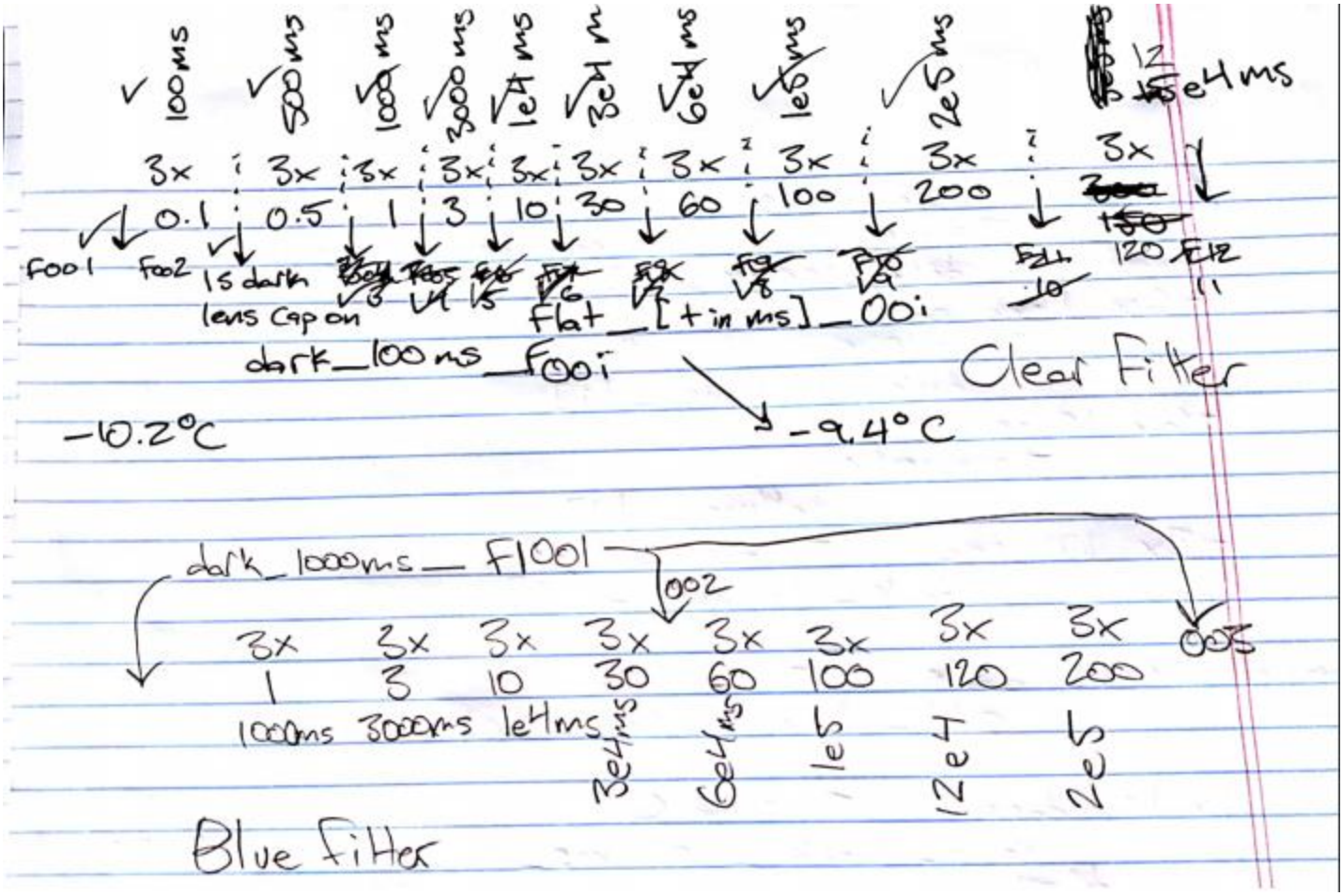
Calibration: 1x 1s dark T = -10°C
calib_dark_1000ms_001 CAP ON!

1x 1s dark T = -10°C
calib_dark_1000ms_002 CAP OFF!
* This is to check cap on/cap off light leak.

B| 1x 1s dark T = -10°C
calib_dark_1000ms_003

bias 1 [1x 200ms dark T = -10.2°C
bias_200ms_001

1x 1s dark T = -9.8°C
bias 2 → dc_1000ms_001
1x 5s dark T = -10.2°C
bias 3 → dc_5000ms_001
1x 25s dark t = -10.2°
bias 4 → dc 25000ms_001
1x 125s dark t = -9.8°C
bias 5 → dc_125000ms_001
1x 600s dark t = -10.2°C
bias 6 → dc_6e5_001


100ms
500ms
1000ms
3000ms
1e4ms
3e4ms
6e4ms
1e5ms
2e5ms
12
e4ms
3x
0.1
0.5
1
3
10
30
60
100
200
150
120
F001
F002
F012
F011
F010
F009
F008
F007
F006
F005
F004
F003
F002
F001
1s dark
lens cap on
flat_[t in ms]_00i
dark_100ms_F001
Clear Filter
-10.2°C
-9.4°C
dark_1000ms_F1001
002
005
1
3
10
30
60
100
120
200
1000ms
3000ms
1e4ms
3e4ms
6e4ms
1e5
12e4
2e5
Blue filter