Lab 1 Report

Jack Colvin, Along with my group, Larrence Xing, Kevin Wu, Annabelle Yarborough and Olivia Lucal

Part One: Analysis of Archival Data

In this part of the lab, we used an archivale fits data of a past observation using SEO, conducted basic analyses of the raw and calibrated images, as well as the bias, flat, and dark frame images used to convert the raw image into the calibrated image. The specific observation we used was done by the observer kadrlica on 03/25/2025, and can be found on the stars server at https://stars.uchicago.edu/fitsview25/data/2025-03-25/kadrlica (https://stars.uchicago.edu/fitsview25/data/2025-03-25/kadrlica), and found the flat, bias, and dark frame in the MasterCCD Directory: https://stars.uchicago.edu/images/StoneEdge/0.5meter/2025/MastersCCD/ (https://stars.uchicago.edu/images/StoneEdge/0.5meter/2025/MastersCCD/). Specifically, the object we are looking at is a galaxy, and the image was taken in the g-band with a 64 second exposure.

The Following are the file names for all the files we used:

Raw: 12h18m57s+47d18m13s_g-band_60.0s_bin1_250325_053735_kadrlica_seo_0_RAW.fits

Cal: 12h18m57s+47d18m13s_g-band_60.0s_bin1_250325_053735_kadrlica_seo_0_FCAL.fits

Bias: /meanbias_bias_120s_bin1_2025-02-08_seo-seo_lanabv_bias_MBIAS.fits

Dark: /meandark_h-alpha_64.0s_bin1_20250208_012559-013605_chultun_seo_MDARK.fits

Flat: /mflat_g-band_bin1DR_20250208_020015-020219_chultun_seo_MFLAT.fits

For each of the frames mentioned above, we calculated basic summary statistics, such as mean, standard deviation, min, and max pixel values and produced histograms of the pixel distributions, as well as conducted specific analyses for each frame which included trying to identify hot and dark pizels, mapping the point spread function of a specific light source, and identifying and calculate the dark current from the dark and bias frames. Finally, for each image, we used plt.imshow, with origin='lower' and cmap='gray' to plot a correctly oriented image of each frame to compare it to the statistics we will discuss in section 1.1.

1.1: Basic Summary Stastics

1.1 Introduction

We started by importing astropy, matplotlib, and numpy to allow us to import our fits files into python as 2D numpy arrays, and calculated their mean, standard deviation, min, max, and median simply by using numpy operations.

1.1 Data

Stat	Raw	Bias	Dark	Flat	Cal
mean	991	808	883	1.00	6.77e-5
median	988	808	880	1.00	6.76e-5
std	46.1	3.85	44.9	.021	1.03e-5
min	919	777	848	.251	-.00022
max	20322	879	24381	1.33	.0028

Conclusions/Analysis

- What first stands out is how different the values are for the calibrated image than any of the other images. This is because the calibrated image is put into new units, Janskys. This is because the measurements of the other frames are based on the amount of electrons produced by photons hitting each pixel, which is difficult to work and make calculations with, whereas Jansky's are a measurement of flux density composed of standard units (W/m^2*hz), and thus are much easier to use in calculating quantities such as Luminosity or Energy.
- Aditionally, the Flat field also has much lower pixel values than the other frames. This also makes sense, as the flat field doesn't measure pixel values in relation to the incoming light source in terms of electrons produced, as the flat field is taken against a constant light source, but rather the values represent the relative brightness, and thus sensitivity, of pixels in relation to each other. This is also why the mean is almost exactly 1.
- The standard deviaiton of the bias frame is extremely low, which makes sense because the bias frame is taken with a 0 second exposure, and thus doesn't measure deviation in light from the source, but rather inherent noise in the CCD itself, which is much more minimal.
- The max value of the dark frame is very high, which makes sense, as it is used to identify hot pixels, which would have extremely high readings. These hot pixels can then be cut out in the process of producing the calibrated image, which we will carry out as part of our analysis in the next step of this lab.
- The raw image has a mean slightly higher than both the dark and bias frame, but with a similar min and max and standard deviation (at least to the dark frame). This makes sense, as the raw frame is pointing towards actual objects, and thus will naturally be brighter than those frames, but have similar statistics since all of the effects, such as

dark current and bias between pixels, will be present in the raw image as those frames have not yet been removed.

1.2: Bias Frame Analysis

1.2 introduction

We started by conducting an analysis of the Bias Frame, which included plotting a histogram of the counts, and evaluating what it means in the context of what the Bias Frame is trying to measure. Specifically, we wanted to evaluate an estimation for the inherent noise of the CCDs. Visually, this noise will be related to the shape (and specifically the width) of the distribution of pixel values, as a wider histogram means that the pixel values vary by larger amounts, and thus there is more noise, and vice versa. Through our analysis, we obtained a couple ideas of how one might go about measuring this noise. Finally, we created an image of the bias frame to get an idea of what it looks like, and to compare what we see in the image to our calculated statistics.

1.2 analysis (code)

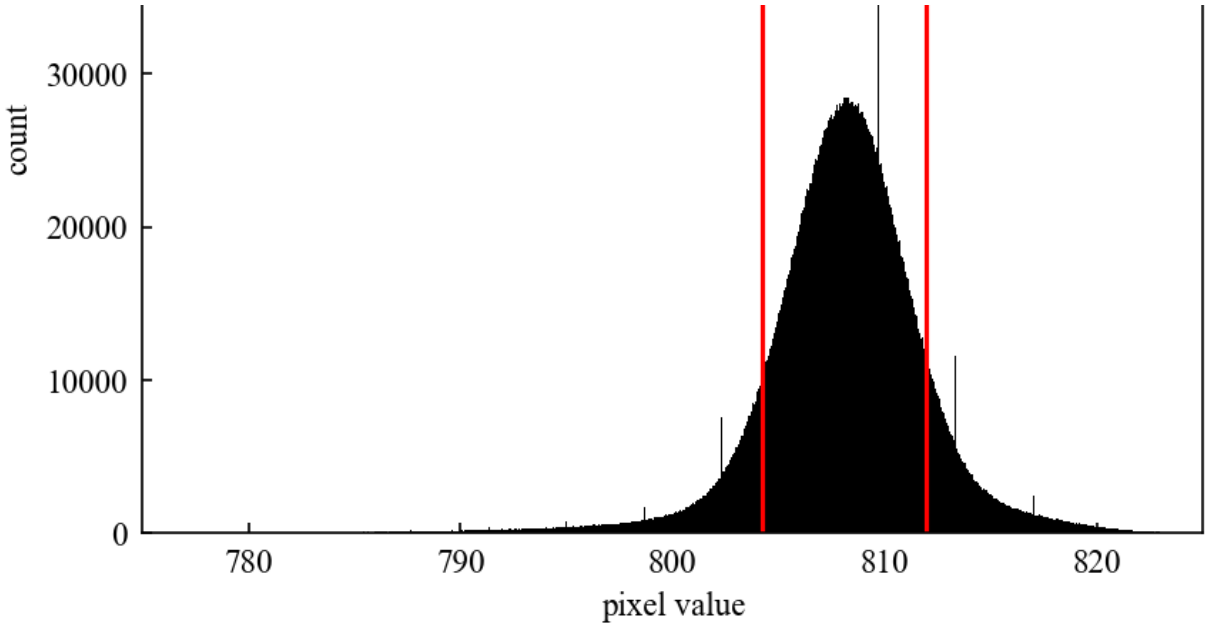
In [1]:
```
import numpy as np
from astropy.io import fits
import matplotlib.pyplot as plt
from astropy.visualization import ZScaleInterval
from matplotlib.colors import LogNorm
from astropy.nddata import Cutout2D
from astropy.modeling import models, fitting

def plot_prettier(dpi=150, fontsize=11, usetex=False):
    '''
    Make plots look nicer compared to Matplotlib defaults
    Parameters:
        dpi - int, "dots per inch" - controls resolution of PNG images
                by Matplotlib
        fontsize - int, font size to use overall
        usetex - bool, whether to use LaTeX to render fonds of axes la
                use False if you don't have LaTeX installed on your sy
    '''
    plt.rcParams['figure.dpi']= dpi
    plt.rc("savefig", dpi=dpi)
    plt.rc('font', size=fontsize)
    plt.rc('xtick', direction='in')
    plt.rc('ytick', direction='in')
    plt.rc('xtick.major', pad=5)
    plt.rc('xtick.minor', pad=5)
    plt.rc('ytick.major', pad=5)
    plt.rc('ytick.minor', pad=5)
    plt.rc('lines', dotted_pattern = [2., 2.])
    if usetex:
        plt.rc('text', usetex=usetex)
    else:
        plt.rcParams['mathtext.fontset'] = 'cm'
        plt.rcParams['font.family'] = 'serif'
        plt.rcParams['font.serif'] = ['Times New Roman'] + plt.rcParam


plot_prettier(dpi=150, fontsize=11)
```

In [2]:
```
bias = fits.open("/Users/jackcolvin//meanbias_bias_120s_bin1_2025-02-0
```

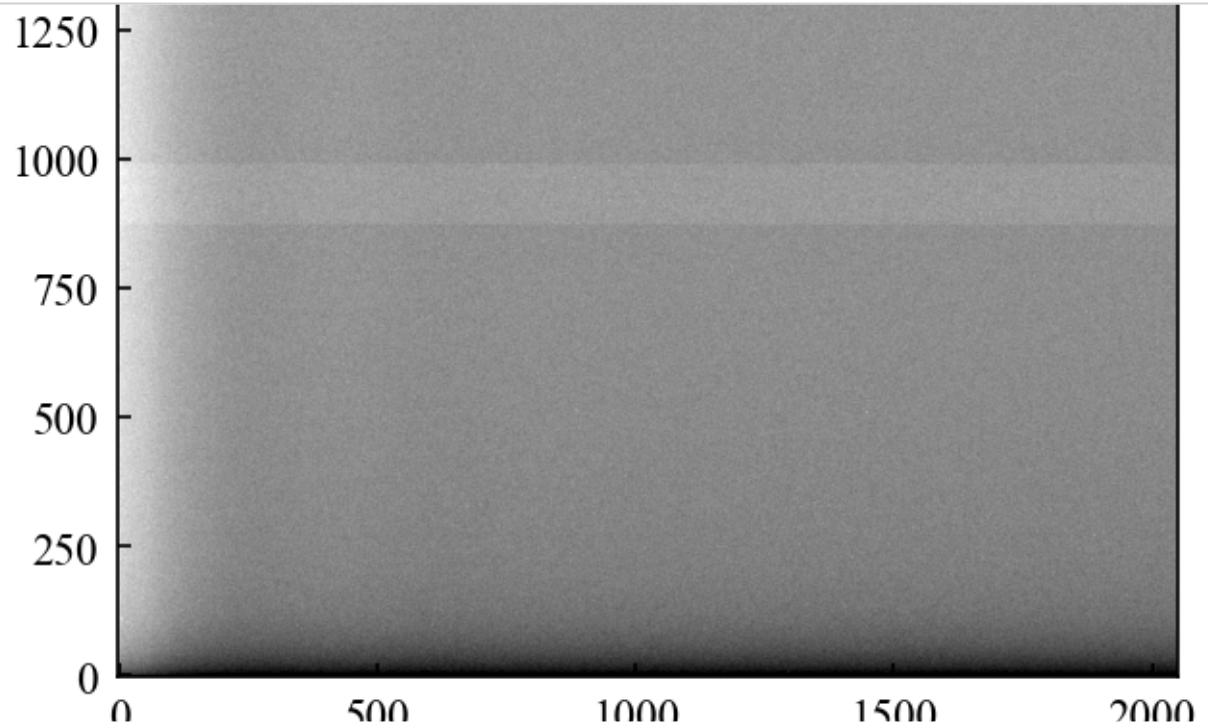
In [3]:
```
data2 = bias[0].data
#print(data.columns.names)
header = bias[0].header

print(f' The mean pixel value of the Bias Frame is {np.mean(data2)}')
print(f' The median pixel value of the Bias Frame is {np.median(data2)
print(f' The minimum pixel value of the Bias Frame is {data2.min()}')
print(f' The maximum pixel value of the Bias Frame is {data2.max()}')
print(f' The standard deviation of the Bias Frame is {np.std(data2)}')


plt.hist(data2.flatten(), bins = "auto", color = "black")
plt.axvline(np.mean(data2) - np.std(data2), color = 'red')
plt.axvline(np.mean(data2) + np.std(data2), color = 'red')
plt.legend(['1 sigma threshold'])
plt.xlim(775, 825)
plt.xlabel("pixel value")
plt.ylabel("count")

#The standard deviation actually seems to match the width of the distr
#this may not be perfect however, because this standard deviation can
#so a better strategy might be to fit a gaussian to the histogram
#which will ignore any outlier values and just capture the core of the
#and use the standard deviation of that gaussian as a measure of stati
#as this distribution is clearly not perfectly gaussian
#with tails that are far too long compared to its width
```

```
 The mean pixel value of the Bias Frame is 808.0859610269226
 The median pixel value of the Bias Frame is 808.2
 The minimum pixel value of the Bias Frame is 776.65
 The maximum pixel value of the Bias Frame is 879.4
 The standard deviation of the Bias Frame is 3.8528679069930583
```

Out[3]: Text(0, 0.5, 'count')



In [4]:
```
intervalb = ZScaleInterval()

vminb, vmaxb = intervalb.get_limits(data2)
plt.imshow(data2, cmap='gray', origin='lower', vmin = vminb, vmax = vm
```



In [5]:
```
bias.close()
```

1.2 Conclusions

- The Histogram produced by the bias frame looks generally gaussian, so we first looked to estimate the histogram width, and thus the read noise as well, by standard deviation (1 sigma lines are shown above in red)
- This gives a pretty good estimate of distribution width, and thus read noise in pixels, but the tails of the distribution seem a bit longer than a normal Gaussian, so the standard deviation doesn't perfectly reflect the correct width.
- A potential way to fix this would be fitting the data to a Guassian, and then measuring the standard deviation of that histogram as a measure of read noise. That would remove outliers, but still consider the tails of the distribution to an extent, and thus might provide a better measure of the width.
- The image of the bias frame matches our analysis, with not very much deviation from pixel to pixel when compared with some of the other images, which is reflective of the low value for the standard deviation. Furthermore, there are no noticable dead or hot pixels within this frame, which is reflected in the minimum and maximum pixel values for this frame being relatively close to the mean.

1.3: Dark Frame Analysis

1.3 Introduction

The next frame we analyzed was the dark frame, which is when an exposure is taken with a closed shutter. For this frame, we focused on analyzing and cutting out hot pixels, and estimating the dark current, using a histogram of the pixel values. We are expecting to see a bimodal distribution, with a main peak and a lower peak of higher pixel values corresponding to the hot pixels, which we can subsequently cut out simply from examining our histogram. Finally, we again plotted an image of the pixel values to visualize what the dark frame looks like, and to compare it to our calculated statistics. To calculate the dark current, I used the formula $DC = \frac{<D>-<B>}{exptime} * gain$ where the gain is assumed to be 1, and the exposure time is retrieved from the header file of the Dark Frame.

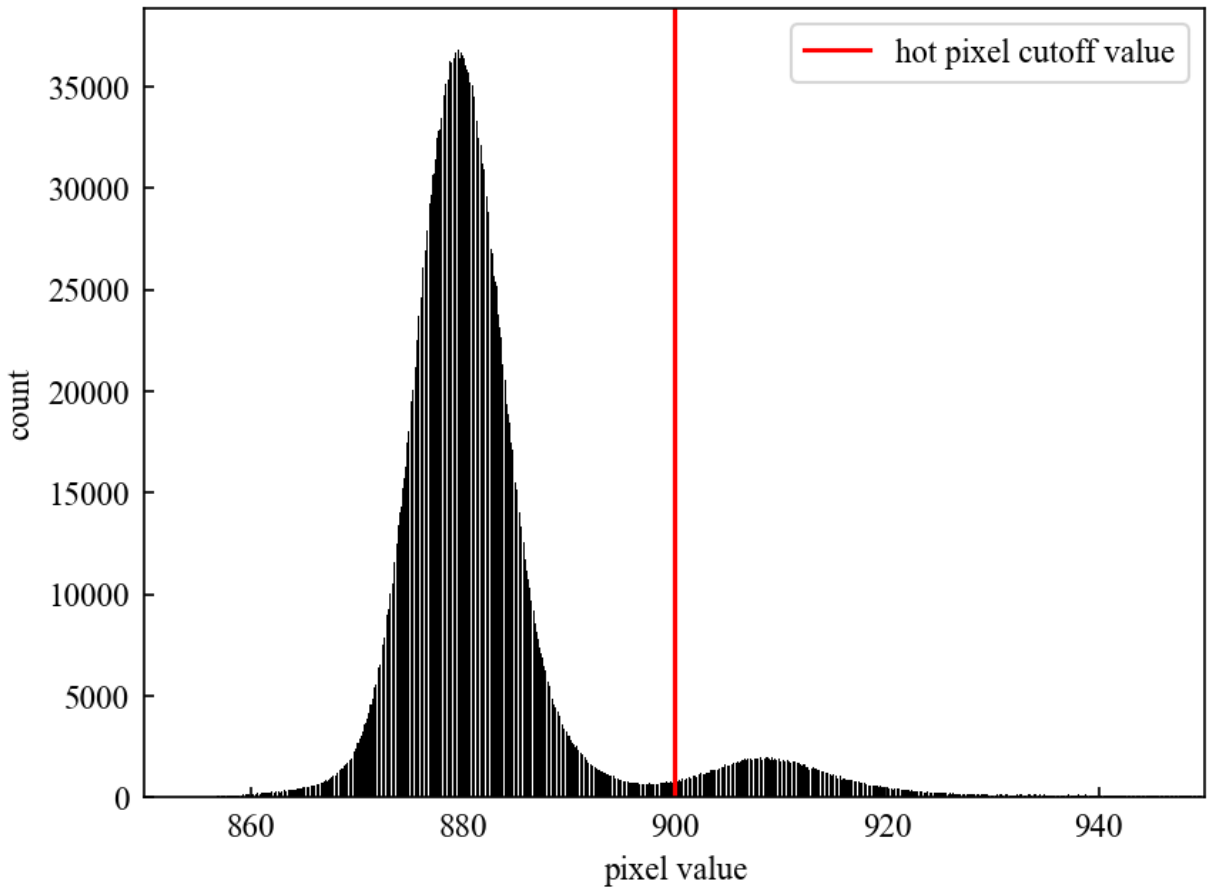
In [6]:
```
dark = fits.open("/Users/jackcolvin//meandark_h-alpha_64.0s_bin1_20250
```

In [7]:
```

```

```
data3 = dark[0].data
#print(data.columns.names)

header = dark[0].header


#gain seems to be 0 according to this header
#which does not make sense
#as the gain should not have a value of 0.

print(f' The mean pixel value of the Dark Frame is {np.mean(data3)}')
print(f' The median pixel value of the Dark Frame is {np.median(data3)
print(f' The minimum pixel value of the Dark Frame is {data3.min()}')
print(f' The maximum pixel value of the Dark Frame is {data3.max()}')
print(f' The standard deviation of the Dark Frame is {np.std(data3)}')

exp_time = header['EXPTIME']#this is the exposure time
print(f' the exposure time is {exp_time}, to be used to calculate dark
plt.hist(data3.flatten(), bins = 'auto', color = 'black')
plt.xlabel('pixel value')
plt.ylabel('count')
plt.axvline(900, color ='red')
plt.legend(['hot pixel cutoff value'])
plt.xlim(850, 950)

#Bimodal distribution, hot pixels represent the second peak
#so cuts should be done on this image at a value of around 900.
#check fraction of pixels that would get rejected:

cut = data3.flatten() > 900

rejected_dark = data3.flatten()[cut]

rej_fraction = len(rejected_dark.flatten())/len(data3.flatten())
print(f'the fraction of pixels rejected by our cut is {100*rej_fractio

#about 7.7% of pixels would get rejected by this cut.
```

```
The mean pixel value of the Dark Frame is 883.5531473085869
The median pixel value of the Dark Frame is 880.3
The minimum pixel value of the Dark Frame is 847.9
The maximum pixel value of the Dark Frame is 24380.5
The standard deviation of the Dark Frame is 44.897683416718685
the exposure time is 64 , to be used to calculate dark current
```

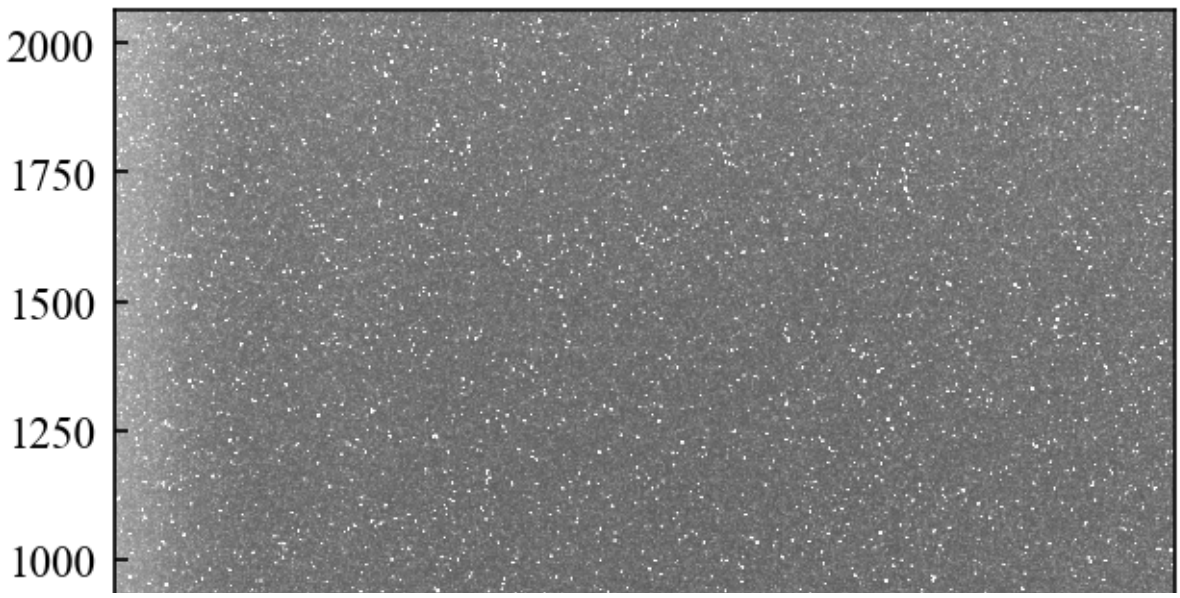
```
the exposure time is 64, to be used to calculate dark current
the fraction of pixels rejected by our cut is 7.687851070433624%
```

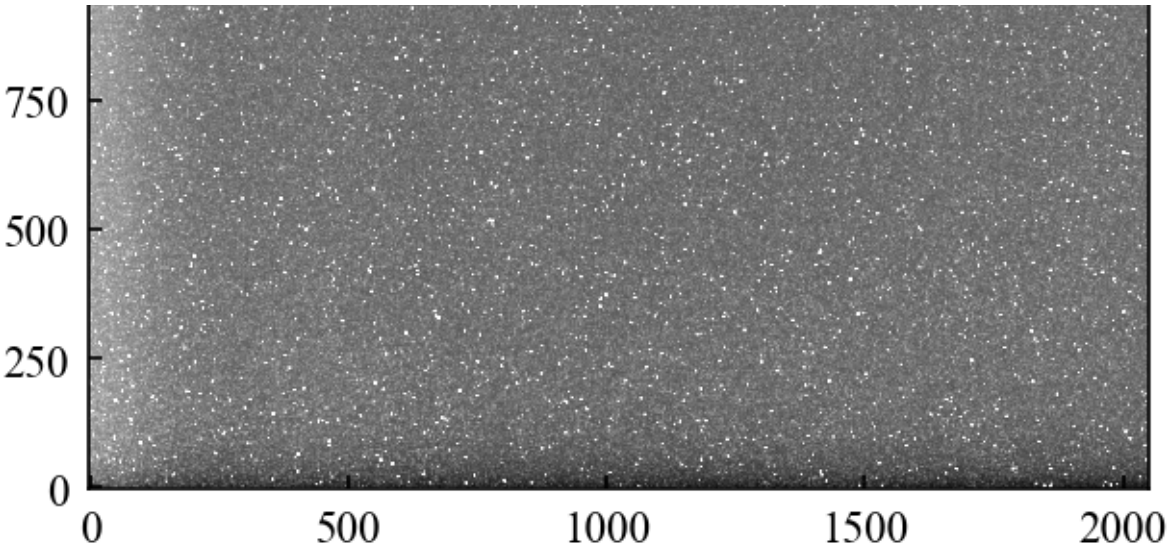


In [8]:

```
interval1 = ZScaleInterval()

vmin1, vmax1 = interval1.get_limits(data3)

plt.imshow(data3, cmap='gray', origin='lower', vmin = vmin1, vmax = vm

#gain = 1
#exposure time = 64
#average dark frame value = 883.5
#average bias frame = 808.1

#dark current = <D> - <B> / Exposure Time * gain

dark_current = ((883.5 - 808.1)/ 64)*1

print(f'the dark current is {dark_current} e/p/s')

#the dark frame is taken at -20C
#the bias frame is taken at -20C as well
#this can affect the measurement since if these two frames are taken a
#then they can't just be subtracted like this to find the dark current
#as the bias at the temperature the dark frame is taken at may be diff
#thus leading to a different calculation for dark current.
#in this case however, our calculation is valid
#as the temperature is the same for both dark current and bias frame.
```

```
the dark current is 1.1781249999999996 e/p/s
```





In [9]:
```
dark.close()
```

1.3 Conclusions

- As Expected, the histogram shows a bimodal distribution, with one peak at around a value of 875, and a second around a value of 910. Visually, we chose a cutoff threshold of 900 or above to eliminate hot pixels, which eliminates 7.7% of pixels
- We also calculated a dark current of 1.18 e/p/s, which seemed quite high compared to some basic research, but still within a reasonable range if conditions were not ideal.
- The Dark Frame was taken at -20 degrees celcius, the same temperature as the bias frame, which allows us to get an accurate calculation of the dark current at that temperature. If that was not the case, this measurement could be skewed, as the bias frame and dark frame both change at different temperatures, and thus the calculated dark current, which depends on the bias frame, would be different.
- The image again reflects the statistics. Most of the pixels are black or grey, but there are a select few hot pixels that are white, which reflects the bimodal nature of our histogram.

1.4: Flat Field Analysis

1.4 Introduction

The final frame we analyzed before looking at the calibrated image was the Flat Field. As with the Bias and Dark frames, we plotted a histogram of the pixel values of the flat field, and analyzed it relative to the image produced. We also described how its statistical properties relate to the purpose of the Flat Field In quantifying the variation of pixel sensativities across the CCD cameras, as the Flat Field is taken against a uniform background, and thus any pixel variations are due to intrinsic differences in the pixel's sensativities to light, and nothing else. Interestingly, unlike the other frames, the flat field included many nan values, so my analysis included removing those values and plotting an image both with and within those values to compare them.

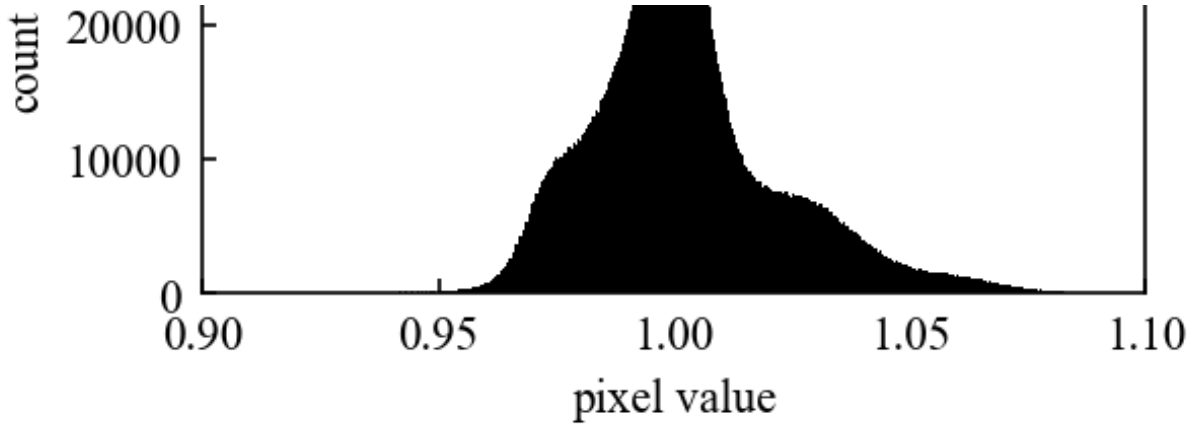
1.4 Code

In [10]:
```
flat = fits.open("/Users/jackcolvin//mflat_g-band_bin1DR_20250208_0200
```

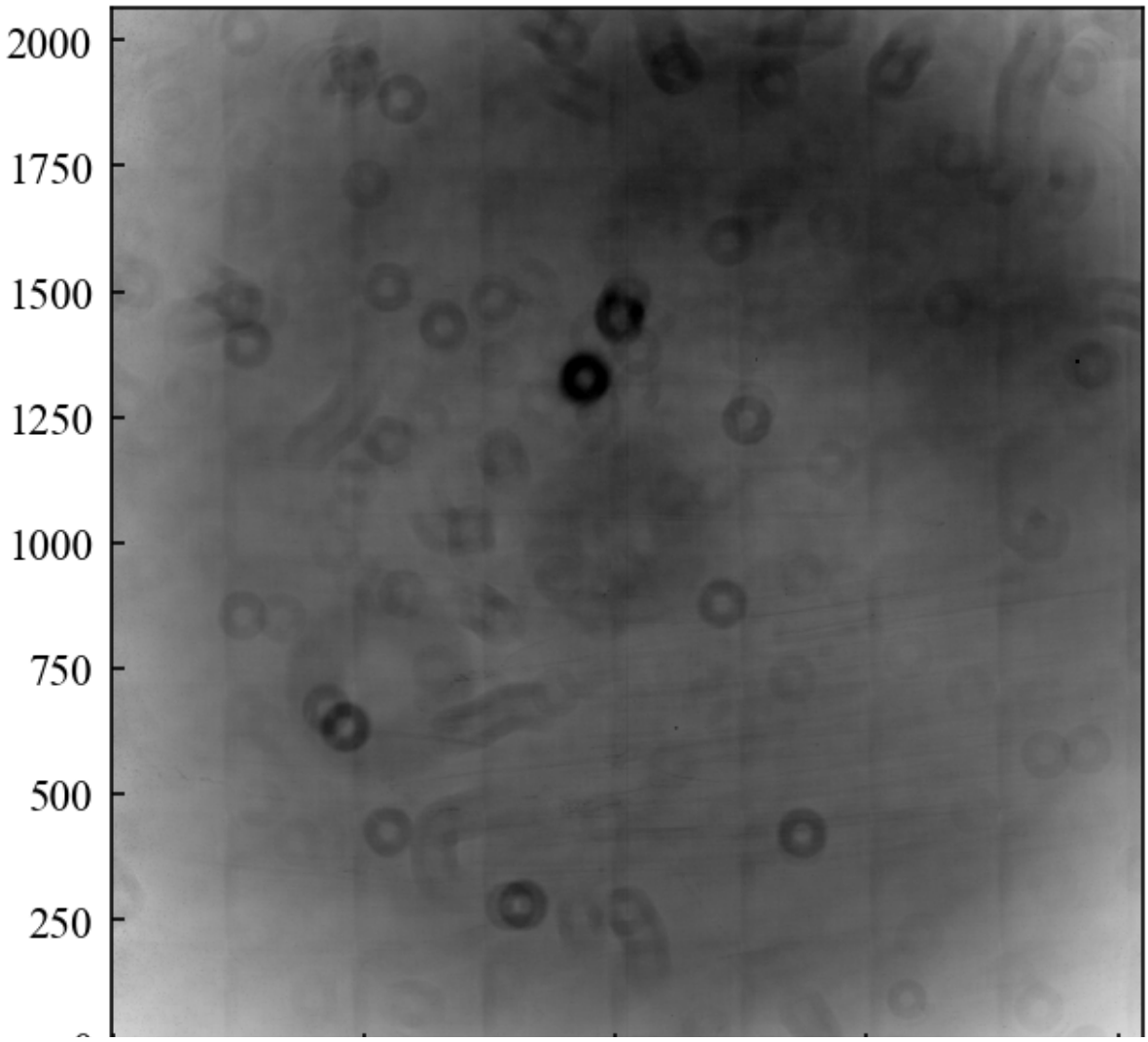
In [11]:
```

```

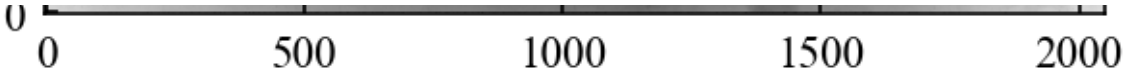
```
data4 = flat[0].data

cut = ~np.isnan(data4) #cut all values that are not nan values
data5 = data4[cut]


print(f' The mean pixel value of the Flat Field is {np.mean(data5)}')
print(f' The median pixel value of the Flat Field is {np.median(data5)
print(f' The minimum pixel value of the Flat Field is {data5.min()}')
print(f' The maximum pixel value of the Flat Field is {data5.max()}')
print(f' The standard deviation of the Flat Field is {np.std(data5)}')
plt.figure(figsize = (4,2))
#For some reason, I ha dto make this figure much smaller than the othe
plt.hist(data5.flatten(), bins = 'auto', color = 'black')
plt.xlim(.9, 1.1)
plt.xlabel('pixel value')
plt.ylabel('count')
plt.show()

#the histogram is not approximately gaussian
#but is rather an asymetric distribution
#the flat field is taken at a constant illimination
#and thus the distribution of pixels in the flat field represent the r
#sensativities of the pixels to each other
#specifically, the mean of the pixels is almost exactly 1, and the med
#so all pixels will likely be scaled by their relative values in the c
#with pixels of value 1 being unweighted.

data4_orig = data4.copy()
cut2 = np.isnan(data4)
data4[cut2] = np.mean(data5)
```

```
The mean pixel value of the Flat Field is 1.002423749004671
The median pixel value of the Flat Field is 1.0
The minimum pixel value of the Flat Field is 0.2519482896898261
The maximum pixel value of the Flat Field is 1.3306420020076193
The standard deviation of the Flat Field is 0.020901001915530425
```

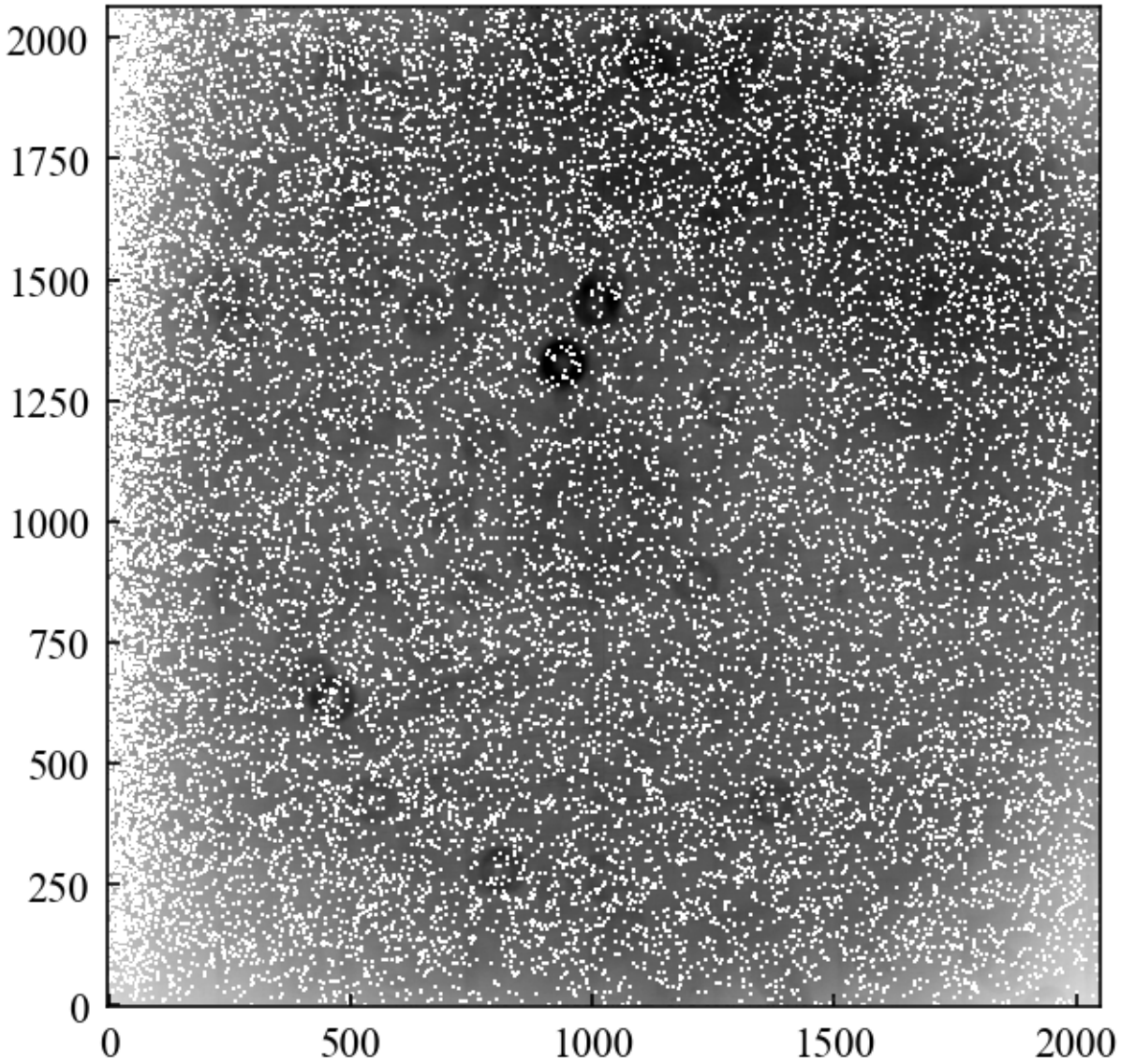


In [12]:
```
interval2 = ZScaleInterval()
vmin2, vmax2 = interval2.get_limits(data4)
plt.imshow(data4, cmap='gray', origin='lower',vmin = vmin2, vmax = vma
plt.show()
plt.imshow(data4_orig, cmap='gray', origin='lower',vmin = vmin2, vmax
```





Out[12]: <matplotlib.image.AxesImage at 0x142fa82d0>



In [13]:
```
flat.close()
```

1.4 Analysis/Conclusions

- The Flat Field quantitatively represents how sensative pixels are to incoming light relative to the average pixel (which has a value 1), and thus can be used to weight pixels based on their sensativities in the calibrated image.
- Pixels with higher values gathered more light when looking at a source of constant flux and thus are more sensitive, and should be dimmed by a factor corresponding to their value in the flat field, whereas pixels with values below one are darker despite recieving the same amount of lught and thus should be brightened by a factor corresponding to their pixel value
- This meaning of the flat field is represented by how the average of the pixel values is almost exactly one, which is stastically necessary as 1 corresponds to a pixel that is of averrgae sensativity, and thus does not need to be scaled in the final calibrated image.
- These varying sensativities are likely due to the quantum efficiency of the CCD itself varying across the camera senser, with brighter pixels having higher quantum efficiencies, and dimmer ones have lowe quantum efficiencies.
- I plotted both an image of the original flat field, and the flat field where the nan values were cut and replaced with the mean value of the image. We can see that the nan values are responsible for hot pixels within the image, and thus those pixels will likely also need to be removed or edited out from the raw image in the calibration process. You can still see that the two images have the same general structure, even if one has many nan valued pixels.
- It is also intresting that this image shows the fingerprints, but the raw image does not, which seems like it should effect the calibration process in some way. I would be interested to know why this is the case.

1.5: Calibrated Image Analysis

1.5 Introduction

The Final Step of the analysis process was conducting an analysis of the final calibrated image produced by removing the bias, dark, and flat frames from our raw image using a pipeline (done automatically in the stars server). Specifically, we looked at trying to estimate the point spread function (PSF) of the image, which corresponds to how light is spread around the astronomical objects in the image. Specifically, we picked one of the stars within the image (we chose a star rather than a galaxy so we could estimate it as a point source), and fit a gaussian to the pixel values in a small regoin around that source, which we used as an approximation of the PSF. From there, we calculated the full-width at half maximum (FWHM), the distance in pixels between the brightest point in the image to a point with half the intensity. This measurement represents a good estimation for the angular resolution of the camera on the specific day and time our image was taken, as it describes how well the image is able to pinpoint a point source of light (like a star). Specifically, a smaller FWHM represents a sharper image, as the light from the star bleeds less into neigboring pixels and thus drops off more sharply, wheras a larger FWHM means that the light is blurrier and extends into a larger number of pixels. To do this, I used the models.Gaussian2D function to approximate a Gaussian given an estimated standard deviation and mean (which I estimated from the image and tried multiple nearby values to ensure that my results were stable). I then fed my specific cutout section of my image into this model to fit my approximated gaussian specifically to the pixel values of my image. Finally, I plotted both my cutout and the Guassian approximation to compare them and ensure my fit was reasonable.

As with all of the other frames, I also plotted an picture of the entire calibrated image to compare with the summary statistics I did to analyze it.

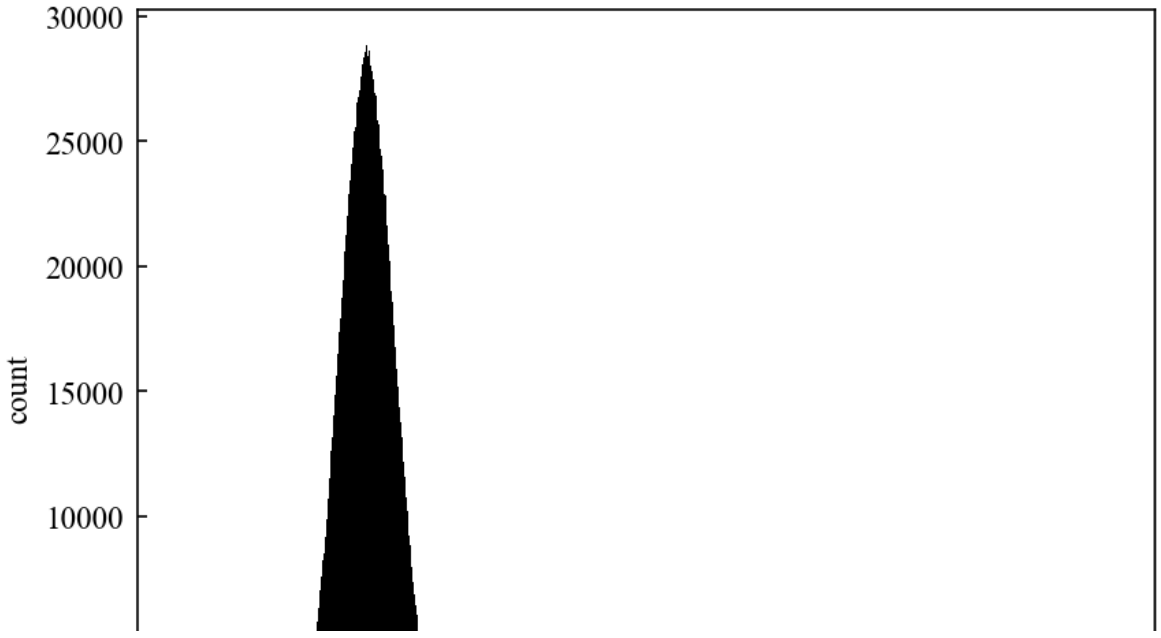
1.5 Code

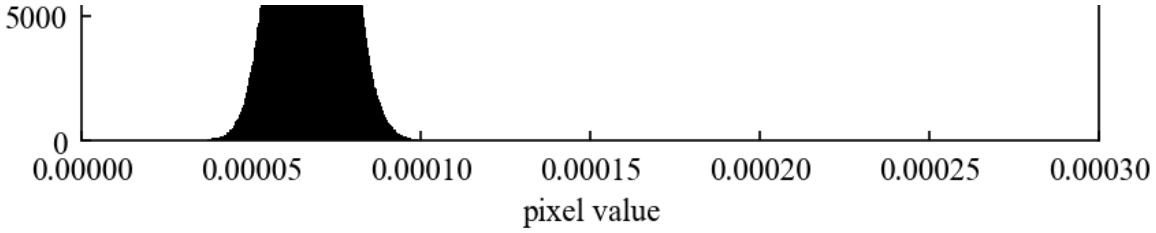
In [14]:

```
cal = fits.open("/Users/jackcolvin/12h18m57s+47d18m13s_g-band_60.0s_bi

data = cal[0].data
#print(data.columns.names)

print(f' The mean pixel value of the Calibrated Image is {np.mean(data
print(f' The median pixel value of the Calibrated Image is {np.median(
print(f' The minimum pixel value of the Calibrated Image is {data.min(
print(f' The maximum pixel value of the Calibrated Image is {data.max(
print(f'The standard deviation of the Calibrated Image is {np.std(data


plt.hist(data.flatten(), bins = 'auto', color = 'black')
plt.xlabel('pixel value')
plt.ylabel('count')
plt.xlim(0, .0003)
```

```
 The mean pixel value of the Calibrated Image is 6.772128108423203e-0
5
 The median pixel value of the Calibrated Image is 6.760513497283682e
-05
 The minimum pixel value of the Calibrated Image is -0.00022435720893
55439
 The maximum pixel value of the Calibrated Image is 0.002798604313284
1587
The standard deviation of the Calibrated Image is 1.0274275155097712e
-05
```

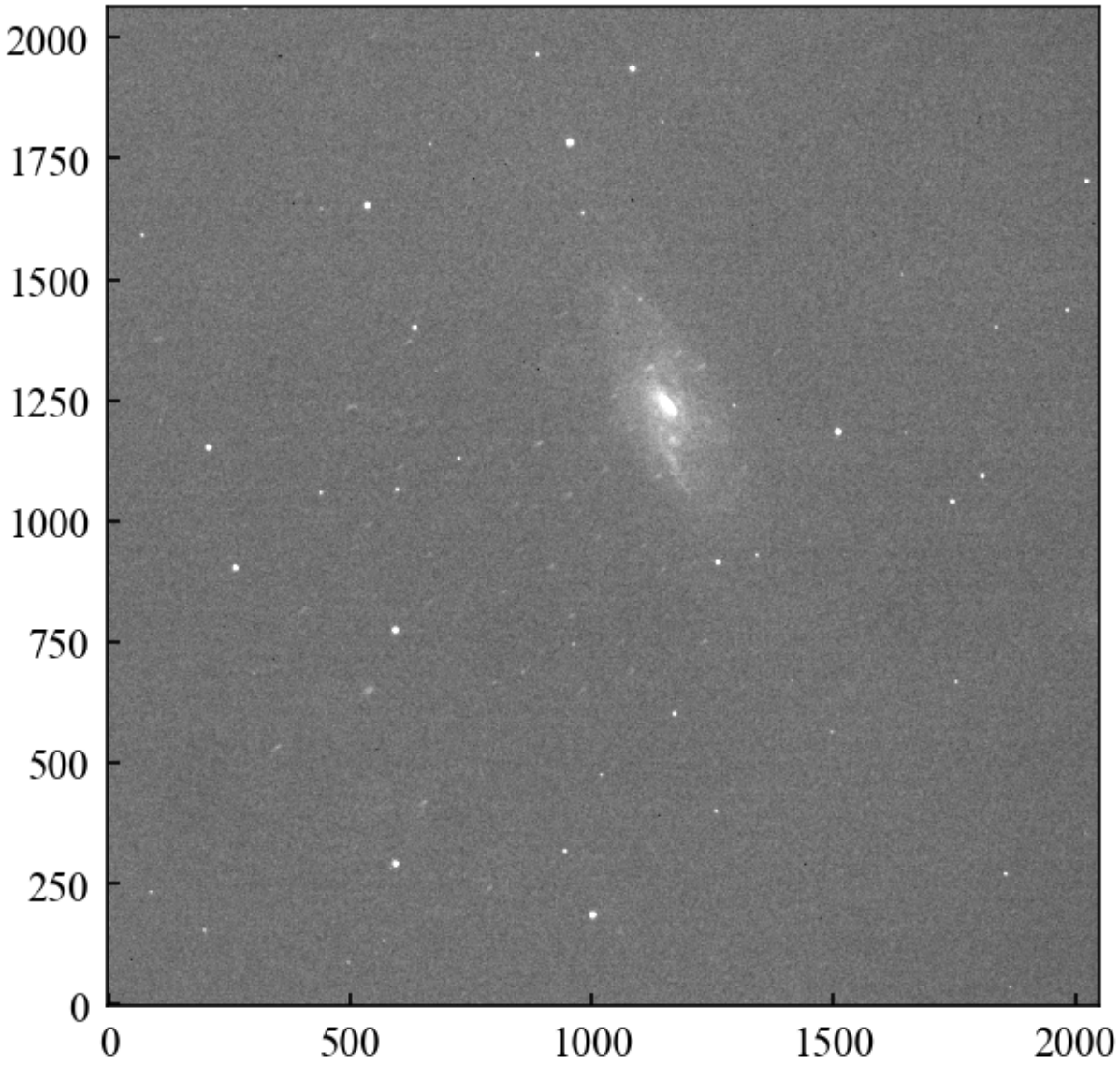
Out[14]: (0.0, 0.0003)





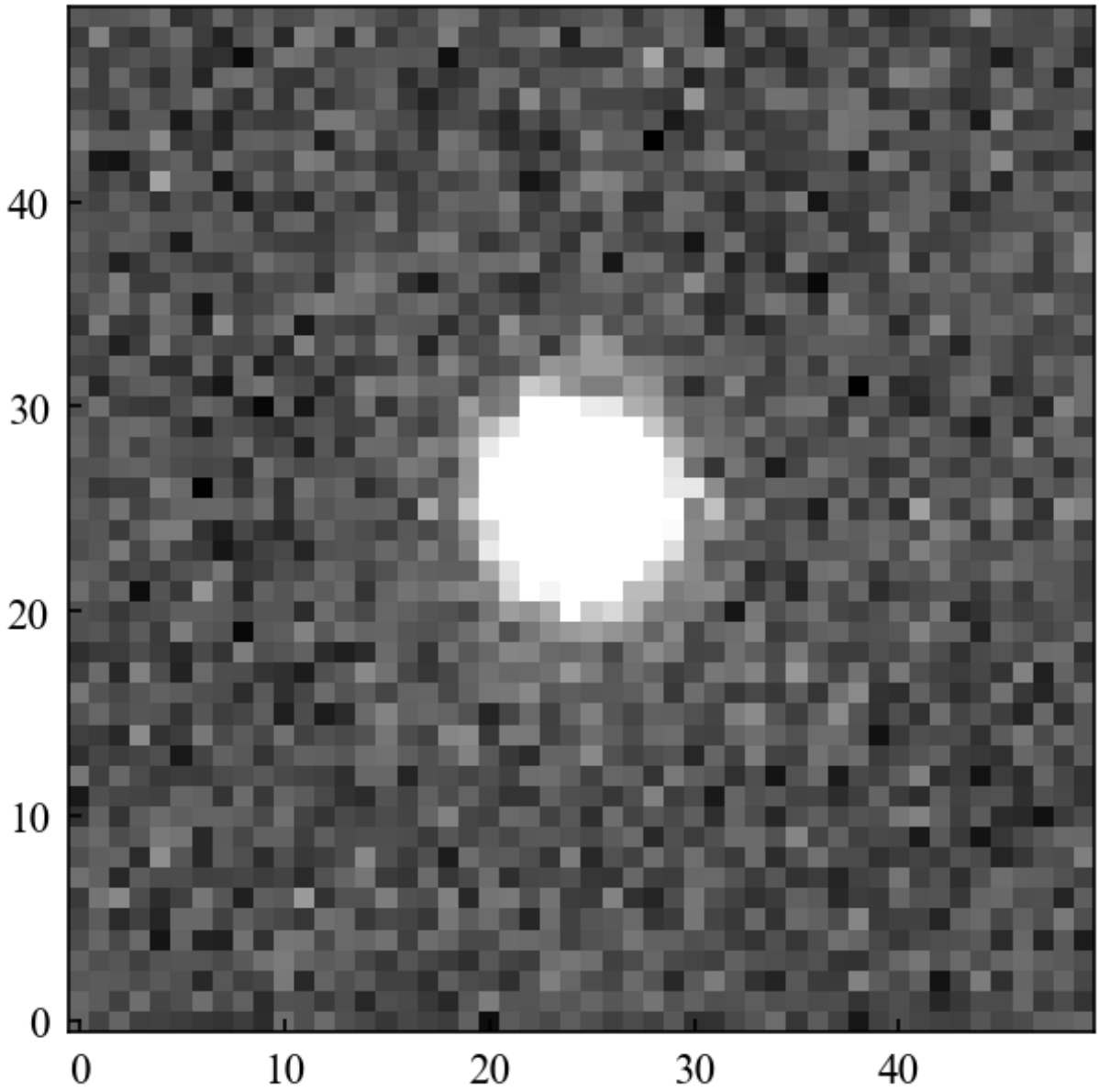
In [15]:
```
interval = ZScaleInterval()
vmin, vmax = interval.get_limits(data)

plt.imshow(data, cmap='gray', origin='lower', vmin = vmin, vmax = vmax
```

Out[15]: <matplotlib.image.AxesImage at 0x34a2ef150>



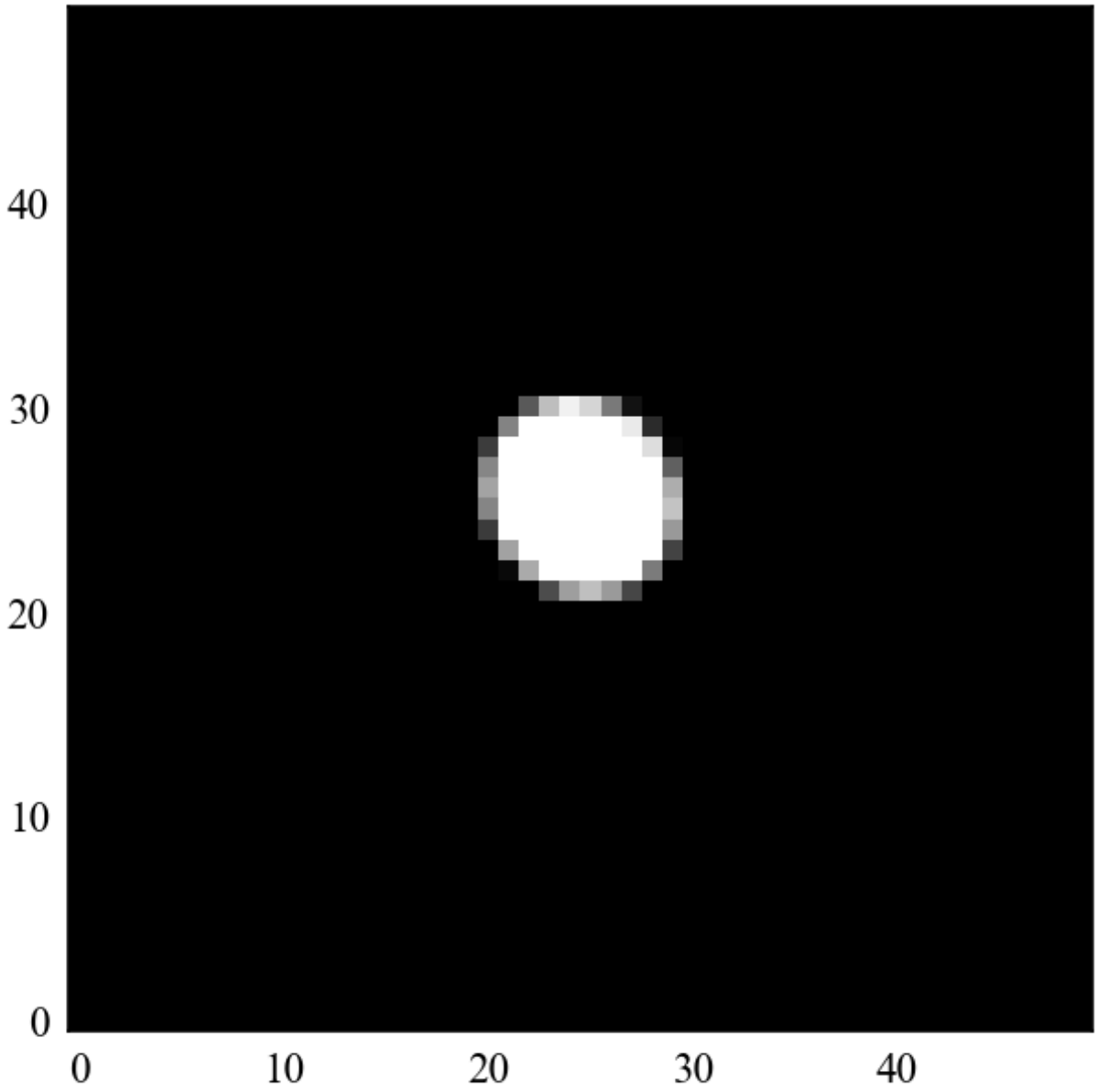
In [16]:
```
#Fitting A gaussian to the PSF
```

```
#coordinates of the galaxy seem to be about (1100, 1250)
#cut out that region than fit a 2d gaussian pdf.
#I just chose a random star to use as an approximation for a point sou

cutout = Cutout2D(data, (595, 775), (50,50))

data_cutout = cutout.data

intervals = ZScaleInterval()
vmins, vmaxs = intervals.get_limits(data_cutout)


plt.imshow(data_cutout, cmap = 'gray', origin = 'lower', vmin = vmins,
plt.show()
#now we have our cutout data, so we need to fit a gaussian to it
#and use the formula FWHM = 2.355 * sigma

xcenter = ycenter = 25
xstd = ystd = 2
maxval = data_cutout.max()

#my fitting model requires a guess for standard deviation and center p
#we also need arrays of the coordinate values of each pixel in x and i
#which are built up below

y = np.zeros(shape = data_cutout.shape, dtype = int)
x = np.zeros(shape = data_cutout.shape, dtype = int)
for i in range(len(y[0])):
    y[i] += len(y[0]) - (i+1)
    x[:,i] += i


gauss_init = models.Gaussian2D(maxval, xcenter, ycenter, xstd, ystd)

fit_p = fitting.LevMarLSQFitter()
gaussian_fit = fit_p(gauss_init, x, y, data_cutout)

pixel_std = gaussian_fit.x_stddev.value

FWHM = 2.355 * pixel_std

print(f'the FWHM is {FWHM} pixels, which corresponds to 3.85 arcsecond


fit_pdf = gaussian_fit(x, y)

plt.imshow(fit_pdf, cmap = 'gray', origin = 'lower', vmin = vmins, vma

#the gaussian model seems like a pretty good representation for our im
#so it seems like it will be a good model to use.
```




```
the FWHM is 4.711431739350338 pixels, which corresponds to 3.85 arcse
conds
```

Out[16]: `<matplotlib.image.AxesImage at 0x34a34f150>`



In [17]:
```
cal.close()
```

1.5 Analysis/Conclusions

- The calculated standard deviation of our fitted Gaussian was 2 pixels. Using the approximation FWHM = sigma * 2.355, we obtain a value for FWHM of 4.7 pixels, which seems relatively accurate looking at our image and our plot of our 2d fitted Gaussian (where the star seems to span about 10 pixels, making 5 pixels a good estimate for the FWHM).
- Using the fact that the camera's field of view is 26 by 26 arcmins = 1560 by 1560 arcsecsm and given that the image is 2048 by 2048 in pixels, we obtain a conversion of 1 pixel = .72 arcsecs, giving a value for FWHM in arcsecs of 3.85 (and thus as an estimation of angular resolution as well).
- This is a quite poor angular resolution, but is reasonable for an observatory like SEO if the conditions were sub-optimal for this specific observation.
- Looking at the Image, it also does look like a lot of the noisiness we see in the raw image (pictured next) has been removed, so we can more clearly see what spots are stars vs hot pixels. There is of course some noisiness still present however, which is clear when looking at a zoomed in image of our cutout.

Bonus: Raw Image Code and Basic Thoughts

There were nio specific questions we were tasked with answering for the raw image, so I just plotted a histogram, calculated summary statistics, and compared those results with an image of the raw image, similarly to what has been done as a part of analyzing all of the other frames.

Code:

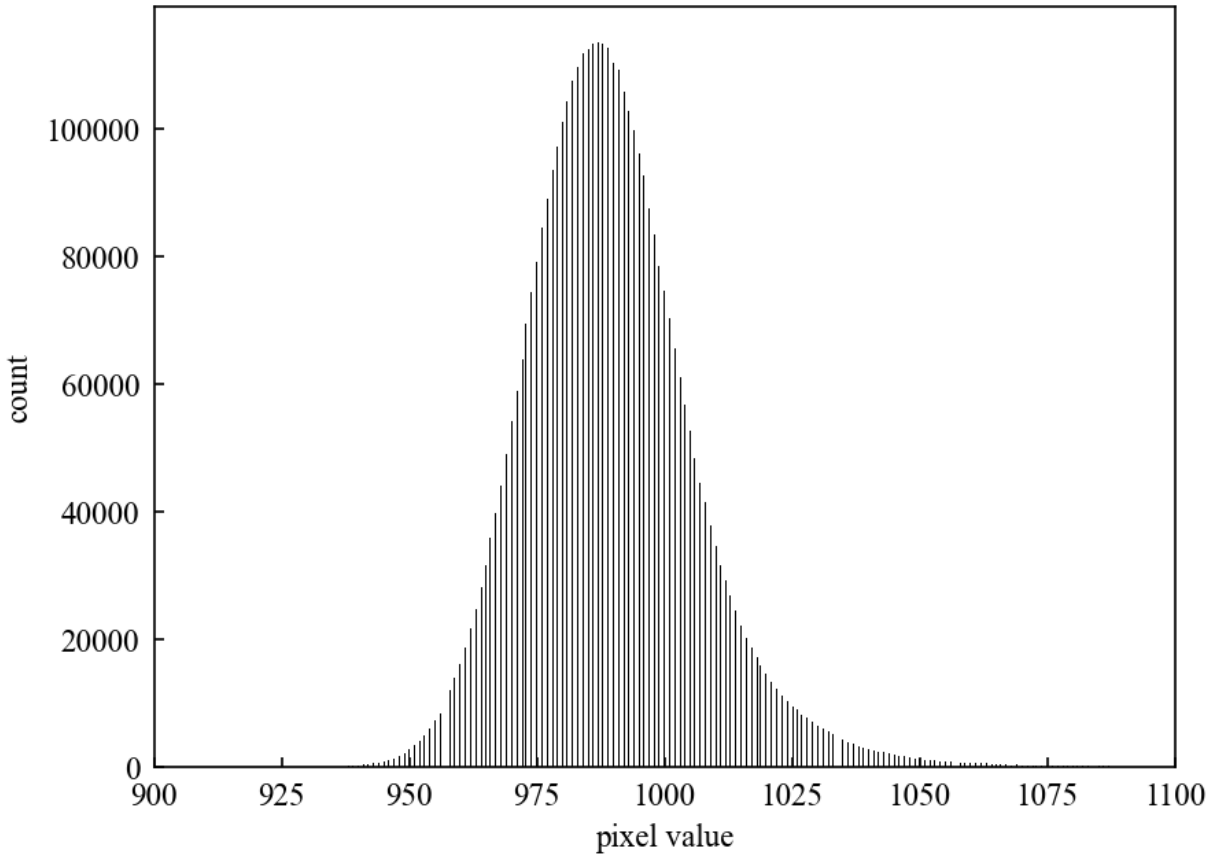
In [18]:
```
raw = fits.open("/Users/jackcolvin/12h18m57s+47d18m13s_g-band_60.0s_bi
```

In [19]:
```

```

```
data1 = raw[0].data
#print(data.columns.names)


print(f' The mean pixel value of the Raw Image is {np.mean(data1)}')
print(f' The median pixel value of the Raw Image is {np.median(data1)}
print(f' The minimum pixel value of the Raw Image is {data1.min()}')
print(f' The maximum pixel value of the Raw Image is {data1.max()}')
print(f' The standard deviation of the Raw Image is {np.std(data1)}')

plt.hist(data1.flatten(), bins = "auto", color = 'black')
plt.xlabel('pixel value')
plt.ylabel('count')
plt.xlim(900, 1100)
```

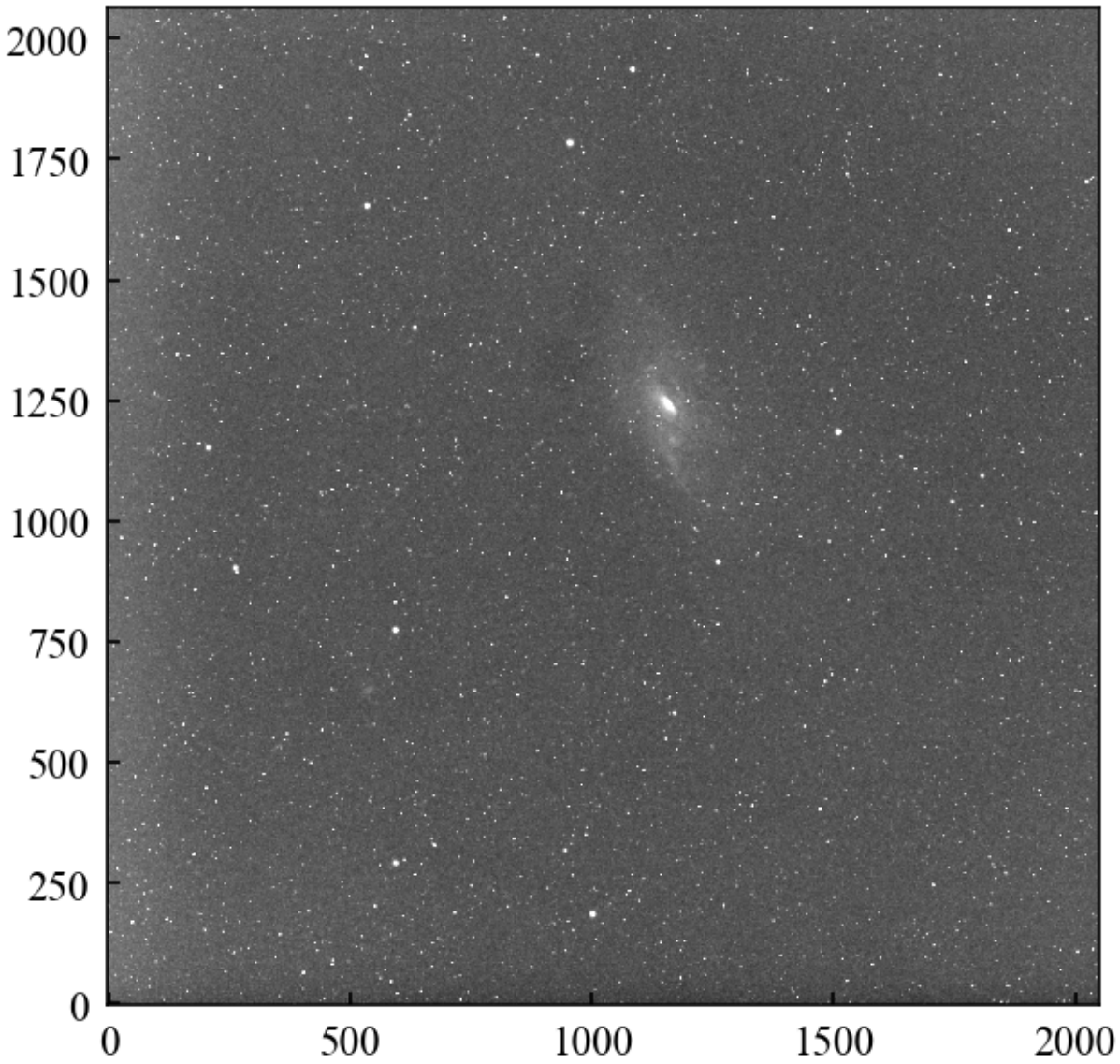
```
 The mean pixel value of the Raw Image is 990.5645820558533
 The median pixel value of the Raw Image is 988.0
 The minimum pixel value of the Raw Image is 919
 The maximum pixel value of the Raw Image is 20322
 The standard deviation of the Raw Image is 46.0902411022738
```

Out[19]: (900.0, 1100.0)



In [20]:
```
interval1 = ZScaleInterval()
vmin1, vmax1 = interval1.get_limits(data1)

plt.imshow(data1, cmap='gray', origin='lower', vmin = vmin1, vmax = vm
```

Out[20]: <matplotlib.image.AxesImage at 0x28ea22590>



In [21]:
```
raw.close()
```

Type *Markdown* and LaTeX: α^2

Raw Image Thoughts

- You can certainly see how the raw image corresponds to the bias, dark, and flat frames much more than the calibrated image. It has a lot more noisiness and points where it is unclear if we are looking at a star, or simply a hot pixel.
- The histogram has a lot of breaks in it compared to the other histograms, furtehr indicating this sort of noisiness rather than the pixels having a continuous, near-gaussian distribution like in the calibrated image.
- Overall it is simply a lot harder to find and focus on what objects might be interesting to observe, althought the main galaxy at the center of the image is clear even in the raw image.

Part Two: Taking Our Own SEO Observa

After Analyzing archival SEO images, we next scheduled observation time to create images of an object of our choosing. We decided to observe M97, because it is very observable from SEO's location. M97 is a Nebula located at Ra = 11h14m16s Dec = 55d01m08s. We took 3 exposures of 120 seconds each, one in the i band, one in the g band, and one in the r band. They were taken at around 11pm Chicago time, so 9pm at the observatory. We had a lot of problems with humidity, with us deciding to take our exposures that early in the night because the humidity later that night and on other nights throughout this past week has been well above the threshold value of 90%. Finally, after taking our three exposures, we combined them using the astropy function make_lupton_rgb into an rgb image, and used the parameters Q and stretch to make our image and object visible. Our observation raw image can be found at https://stars.uchicago.edu/fitsview25/data/2025-04-20/ayarborough (https://stars.uchicago.edu/fitsview25/data/2025-04-20/ayarborough). The files used for our rband, gband and iband are called:

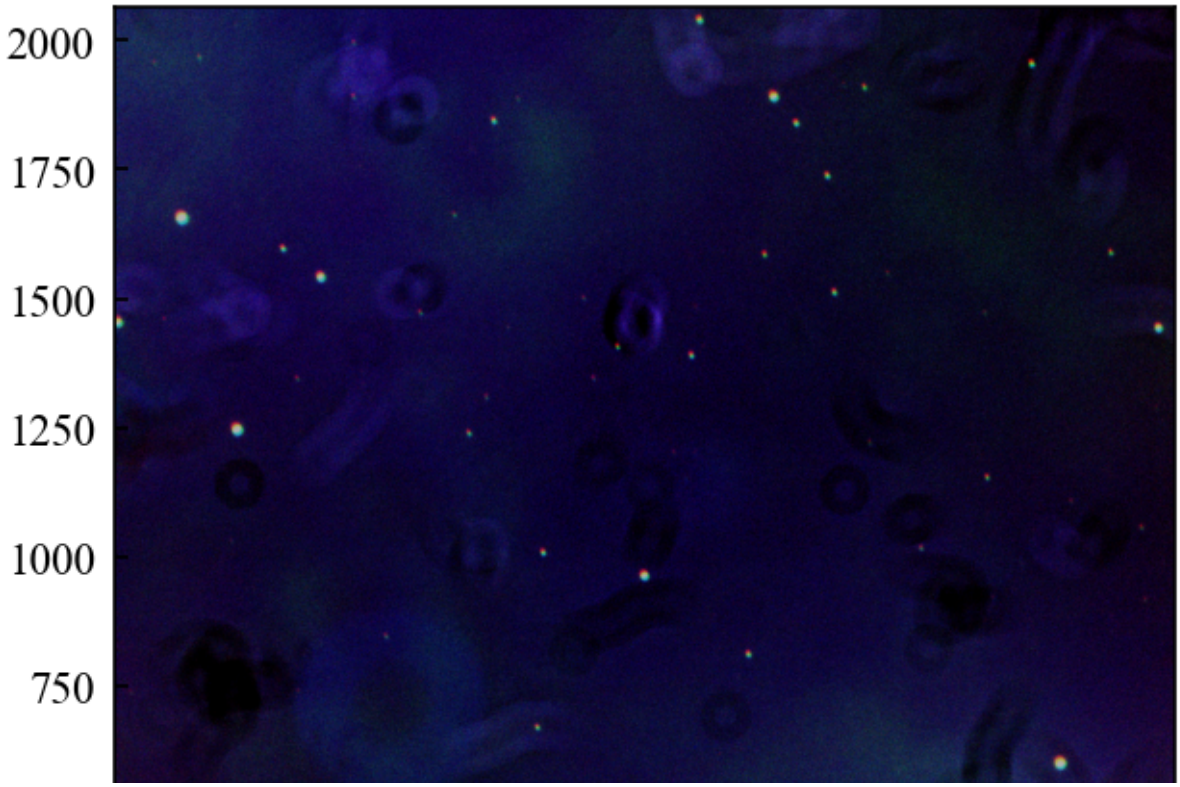
rband: /unknown_r-band_120.0s_bin1_250420_033427_ayarborough_seo_0_FCAL.fits
gband: /unknown_g-band_120.0s_bin1_250420_033056_ayarborough_seo_0_FCAL.fits
iband: /unknown_i-band_120.0s_bin1_250420_033758_ayarborough_seo_0_FCAL.fits

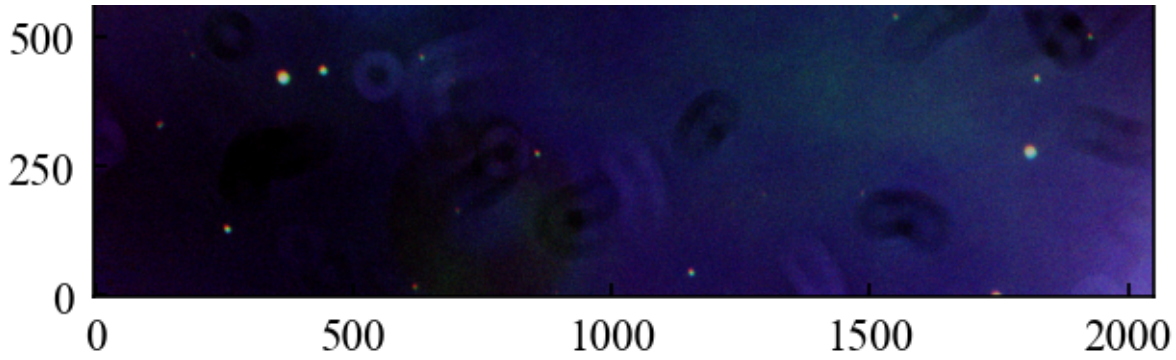
Code To Generate RGB Image

In [22]:
```
rband = fits.open("/Users/jackcolvin//unknown_r-band_120.0s_bin1_25042
gband = fits.open("/Users/jackcolvin//unknown_g-band_120.0s_bin1_25042
iband = fits.open("/Users/jackcolvin//unknown_i-band_120.0s_bin1_25042
```

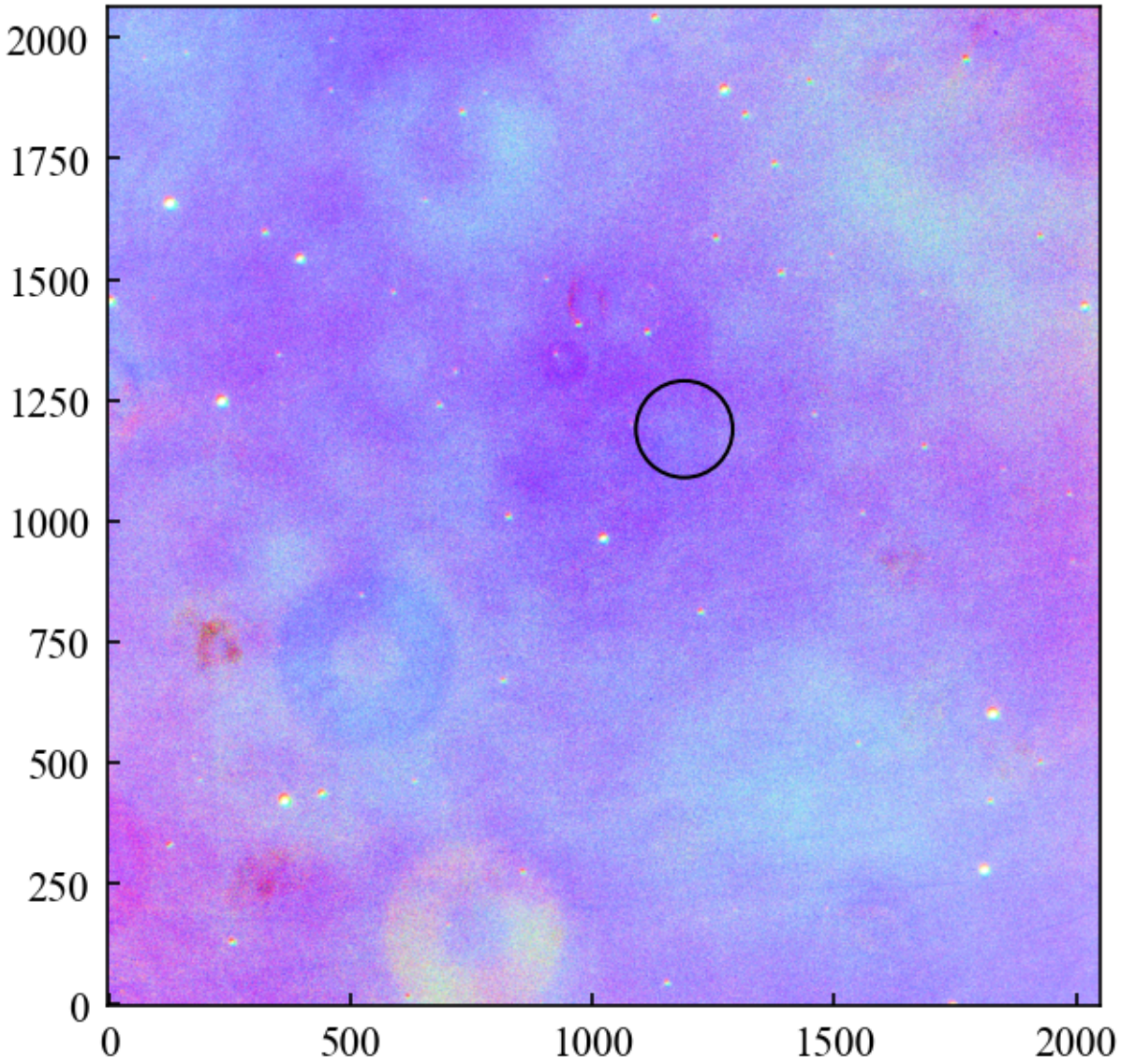
In [23]:
```
```

```
from astropy.visualization.lupton_rgb import make_lupton_rgb
from astropy.visualization import LogStretch
r = iband[0].data
g = rband[0].data
b = gband[0].data

#lupton_rgb doesn't support scaling
#so that must be done manually
zscale = ZScaleInterval()
r_scaled = zscale(r)
g_scaled = zscale(g)
b_scaled = zscale(b)



image = make_lupton_rgb(r_scaled, g_scaled, b_scaled, Q = 0, stretch =
plt.imshow(image, origin = 'lower')
plt.show()
image = make_lupton_rgb(r_scaled, g_scaled, b_scaled, Q = 0, stretch =
circle = plt.Circle((1190, 1190), 100, fill = False)
fig, ax = plt.subplots()
ax.add_patch(circle)
plt.imshow(image, origin = 'lower')
```





Out[23]: <matplotlib.image.AxesImage at 0x28fab1e50>



In [24]:
```
rband.close()
gband.close()
iband.close()
```

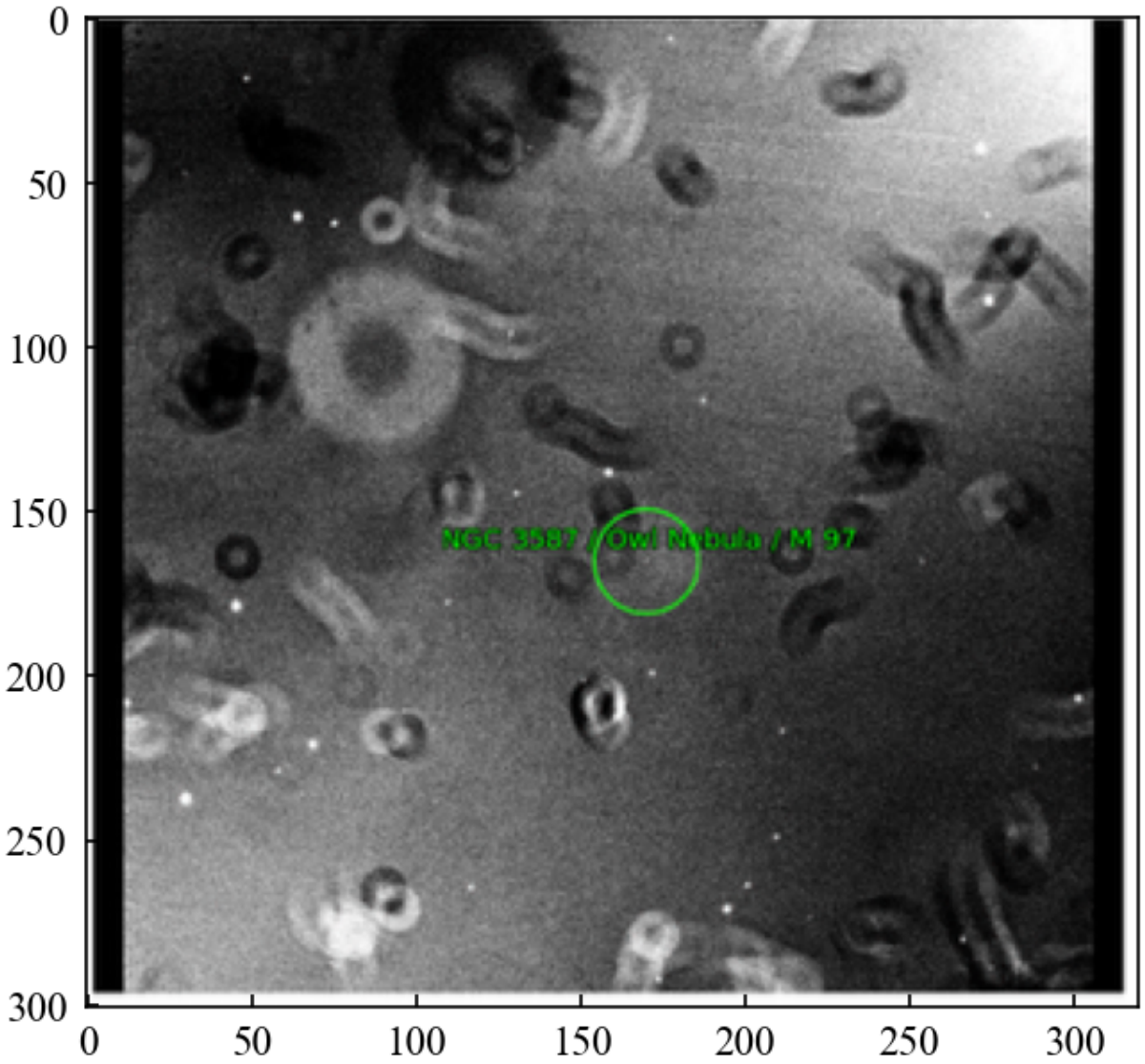
Concluding Thoughts

The biggest challenge I had when generating my RGB image was figuring out the scaling. When just using Zscale, the sky appeared really bright, but when darkening the sky, the stars themselves were also dimmed, so I tried to find a balance between those two extremes. Additionally, although when taking the image, the humidity wasn't above 90%, it was still high (around 85%), which hs clearly affected the image as well, making it appear hazy, especially in the r-band (which is contributing the green color to the image). Finally, we also had trouble discerning exactly which object is M97. In order to visualize M97, I created a second image where the sky is unnaturally bright in which M97 is very slightly visible as the hazy blue dot just to the upper right of the central star (and is circled), but it is still very minimally visible. This makes sense, as it is a nebula, and thus is a very diffuse cloud of dust and light, and given the fact that the humidity was also on the high end of the spectrum, it makes sense that it is hard to discern. We used Astronomy.net to verify that we were indeed seeing M97 (note that the image uploaded to astonomy.net is flipped), and even despite the haziness of our image, Astronomy.net was able to identify M97 within our image.

In [25]:
```
from PIL import Image

img = Image.open("/Users/jackcolvin//thumbnail_Screenshot 2025-04-21 a
```

In [26]:
```
plt.imshow(img)
```

Out[26]: <matplotlib.image.AxesImage at 0x291ef4510>



In [27]:
```
img.close()
```