Imaging Instrumentation Laboratory #2

Spatial Resolution and Noise (MTF and NPS)

Katie Ceraso, Xihan Liu

Introduction

One of the greatest challenges in imaging is identifying and minimizing noise in the imaging system. There are several performance metrics those using and designing imaging systems use to assess the performance of an optical system. One such metric is the Modulation Transfer Function, or MTF, which measures an imaging system's ability to to transfer contrast at a given resolution, taking into account these important imaging factors [1]. Another metric is the Noise Power Spectrum, or NPS, which measures the magnitude of different frequencies of noise that propagate through a system [2]. The key feature of both of these metrics is they take into account spatial resolution, when assessing imaging quality and noise, a factor that was missing in our previous analysis. This led to the following experiment, which consists of three parts. Part 1 involves computing the spatial covariance function and the NPS of averaged no-signal images, in focus signal images, and out-of-focus signal images. The goal of this section is to assess the noise that is intrinsic to the system, and its relationship with focus. Part 2 involves measuring the spatial resolution properties of the system, through collecting images of various spatial resolution targets and calculating the system's MTF using the Edge Spread Function (ESF) method. Lastly, the third part requires taking the highest quality image of a target to determine the limitations of our imaging system.

Methods

The materials used for this lab are as follows:

- 1 CMOS Camera and Lens (mounted on optical post)
- 2 Forked base clamp (plus tie down screws)
- 1 Manual XY Stage (mounted on optical post)
- 1 Lighted Imaging Plate and non-lit backboard
- 2 Vertical supports (plus tie down screws and washers)
- 1 Star pattern target
- 1 Line pair target
- 1 Uniform patch target
- 1 “Knife” edge for obtaining an edge response

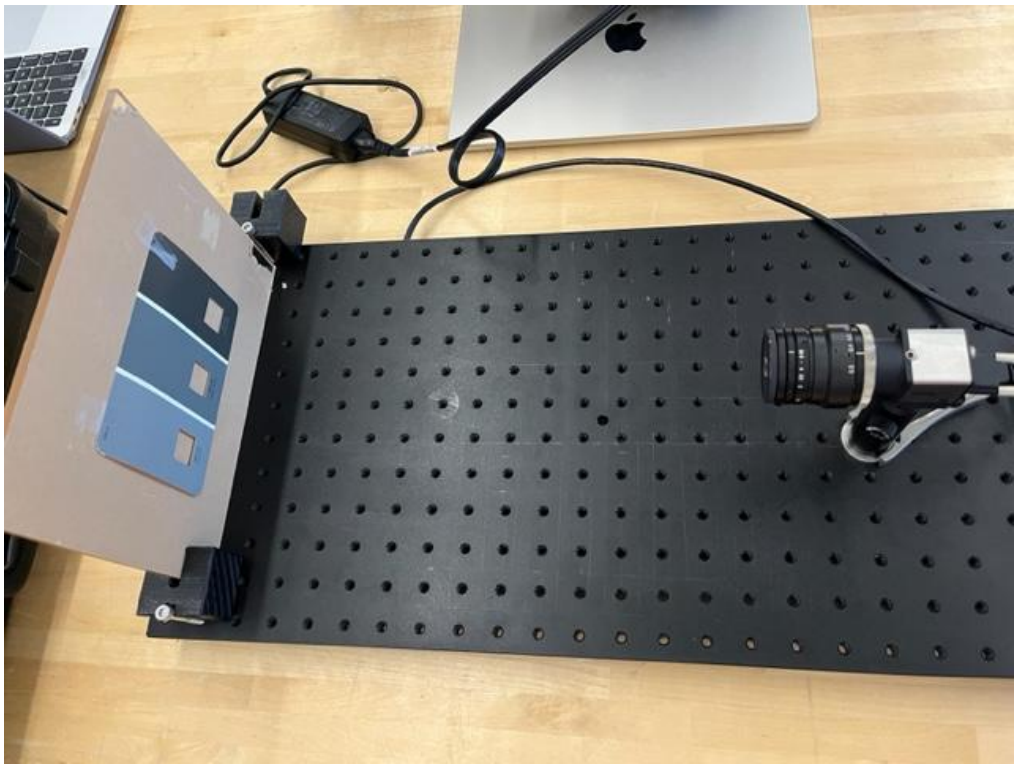


Figure 1: Optical Bench Set-Up for Experiment 1

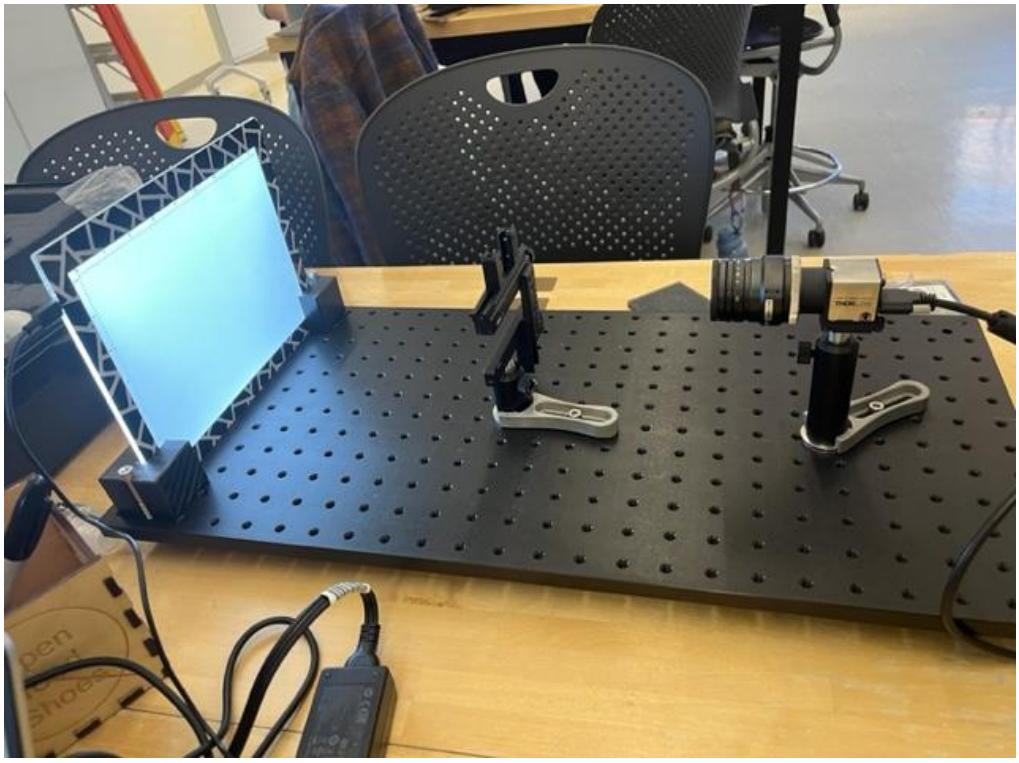


Figure 2: Optical Bench Set-Up for Experiments 2-3

For part 1 of this lab, the lab set up in Figure 1 was utilized. For parts 2 and 3, the lab set up in Figure 2 was used.

Experiment 1: Measuring Spatial Covariance in the Imaging System

In experiment 1, we are going to investigate the covariance in an imaging system with a uniform scene. Figure 1 shows our set up for this experiment. To perform the experiment, we test our imaging system with two conditions: when the lens cap is on and when the lens cap is off. We collect 300 frames in a 60x60 and 100x100 area of interest for each of these two conditions. When we evaluate our imaging system using a uniform patch target (when lens cap off), we collect both defocused and in-focus images to assess the relationship between spatial covariance and focus. The settings for data collection: exposure rate = 25 ms, frame rate = 7 Hz, pixel clock = 7 MHz, focus = 0.4 mm and aperture = 6.

With the collected data, we compute the spatial covariance function and the noise power spectrum (the Fourier transform of the covariance) as our image quality metrics.

Experiment 2: Measuring Spatial Resolution Properties of the Imaging System by Calculating the Modulation Transfer Function

In experiment 2, we used three different imaging targets for observing and measuring the resolution properties of the system. Each of these imaging targets are imaged at three different focus levels. As Figure 2 shows, we will use an XY Positioner to hold the imaging targets between the camera and the backlight board.

The settings of this experiment are: aperture = 16, pixel clock = 7MHz, and distance between imaging targets and front of the camera = 5 inches.

For the first round of imaging, we first focus on the portion of line pair target that covers 2.0 lp/mm down to 10 lp/mm to get the sharpness images. We replace the line pair target with the star pattern target and “knife” edge target sequentially, without repositioning the XY stage or changing the focus or any other imaging conditions. When we use the “knife” target, we vary the angulation of the edge by tilting it a few degrees to ensure good oversampling.

For the second round of imaging, we would perform the same procedure but set the focus on the line pair target until the 4 lp/mm target is just blurred beyond recognizing the different lines. We use the same focus to image the star target and “knife” edge targets.

For the last round of imaging, we perform the same procedure, but set the focus on the line pair target until the 8 lp/mm target is just blurred enough so the separate lines are indistinguishable. Similarly, we would replace the line target with the star and “knife” target for image collection without changing the focus or other imaging conditions.

In this experiment, we would use MTF function to evaluate our imaging system's ability to transfer contrast at a particular resolution from the object to the image.

Experiment 3: Obtaining the Sharpest Possible Image of the Star Target and line part target

In the experiment, our main goal is to obtain the sharpest possible image for the star target, and collect images of the line pair pattern target at these conditions. We collect an area of interest image for each of them based on these settings: Aperture = 8 , Focus = N/A (Hardware is not able to shown) , exposure = 25 ms, pixel_clock = 7MHz, and distance between imaging targets and front of the camera = 5".

Results

Detailed below are the findings from the experiments outlined above. Some key findings from our experiments include finding the dark noise of the system in the form of covariance, and that the Noise Power Spectrum (NPS) decreases at low-frequency as the blur of the system increases. The MTF of the system was calculated with different amounts of blur, and it dropped to 0 at lower spatial frequencies as the blur increases.

Experiment 1: Measuring Spatial Covariance in the Imaging System

In this experiment, a 100x100 pixel AOI was taken from full field dark and light images. The results from this AOI were not entirely uniform, so a second AOI of 60x60 pixels was also analyzed. Both AOIs, their covariance functions, and their Noise Power Spectrums are included below. Their fairly consistent trends, and any inconsistencies, are detailed in the discussion section.

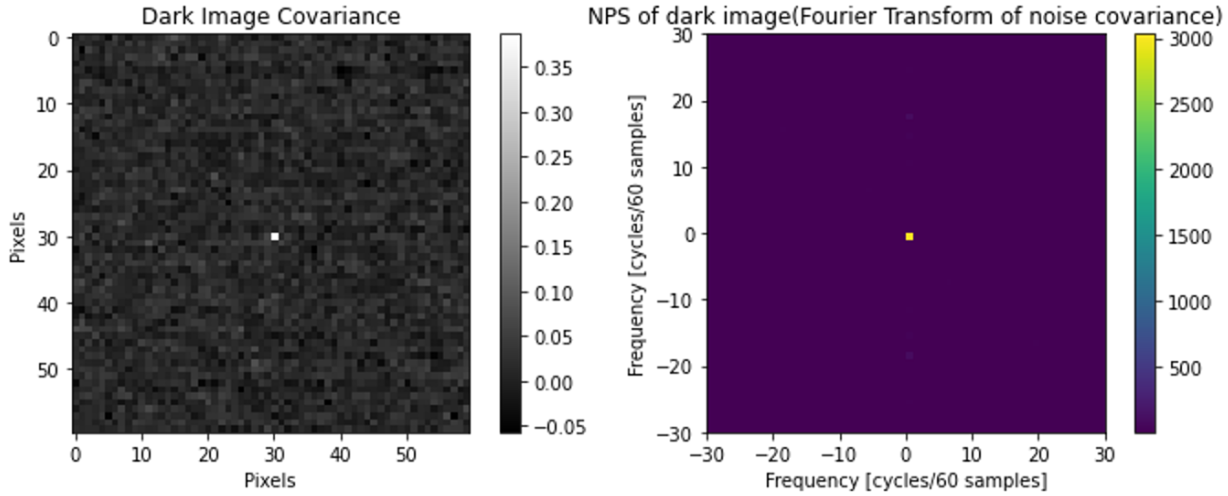


Figure 1: Covariance and NPS of 60x60 AOIs of Dark Image

Left: Covariance of 60x60 AOIs of Dark Image. Right: NPS of 60x60 AOIs of Dark Image.

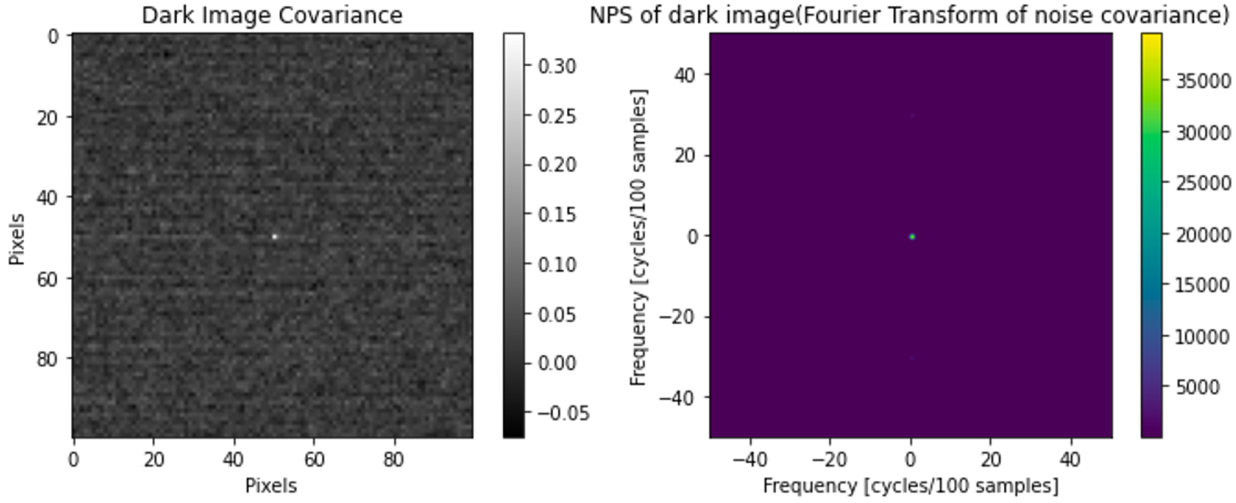


Figure 2: Covariance and NPS of 100x100 AOIs of Dark Image

Left: Covariance of 100x100 AOIs of Dark Image. Right: NPS of 100x100 AOIs of Dark Image.

From Figure 1 and Figure 2, we took dark images with the cap on, and calculated the covariance. One can see that the covariance ranges from -0.05 to 0.40 and -0.05 to 0.35 for 60x60 pixel and 100x100 pixel AOIs, respectively. However, the Noise Power Spectrum for the 60x60 pixel AOI and the 100x100 pixel AOIs have a more significant difference: the center pixel in the NPS of the 60x60 pixel AOI has a value of 3000, while the center pixel in the NPS of the 100x100 pixel AOI is 25000.

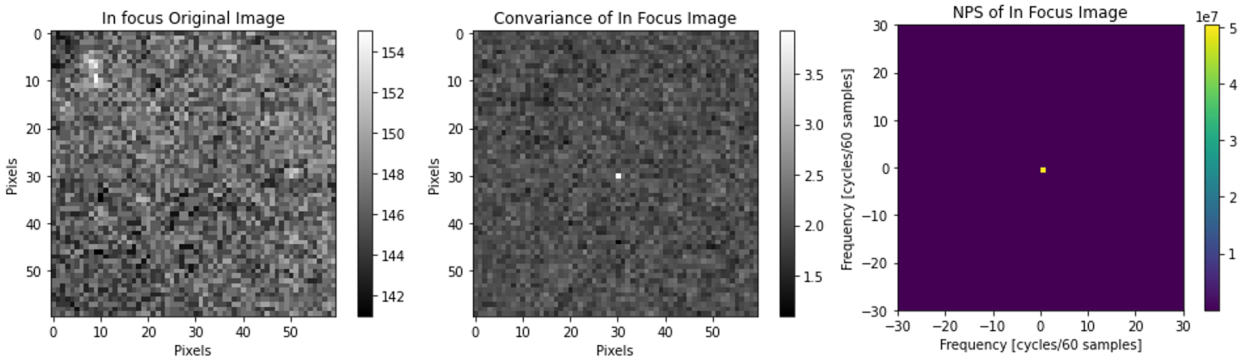


Figure 3: Original, Covariance, and NPS of 60x60 AOIs of In Focus Image

Left: 60x60 AOIs of the original in-focus image. Middle: Covariance of 60x60 AOIs of in-focus image. Right: NPS of 60x60 AOIs of in-focus image.

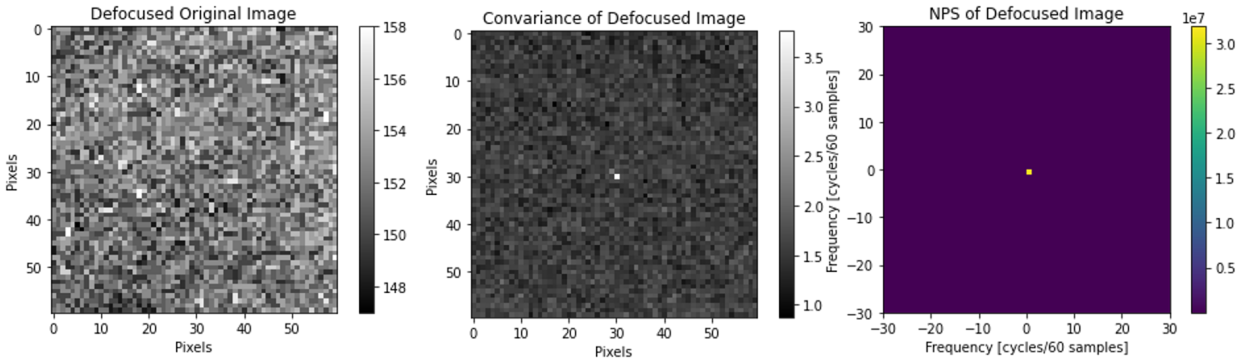


Figure 4: Original, Covariance, and NPS of 60x60 AOIs of Defocused Image

Left: 60x60 AOIs of the original defocused image. Middle: Covariance of 60x60 AOIs of defocused image. Right: NPS of 60x60 AOIs of defocused image.

Additionally, Figures 3 and 4 show the 60x60 AOIs of captured images in focused and defocused conditions with their covariance and noise power spectrum for the center pixel. The intensity of the defocused image is higher than the intensity of the in-focused image. The covariances of the center pixel to the rest of the in-focused and defocused images range from 1.0 to 4.0. The NPS of the center pixel of the in-focused image reaches 5×10^7 and defocused image reaches 3×10^7.

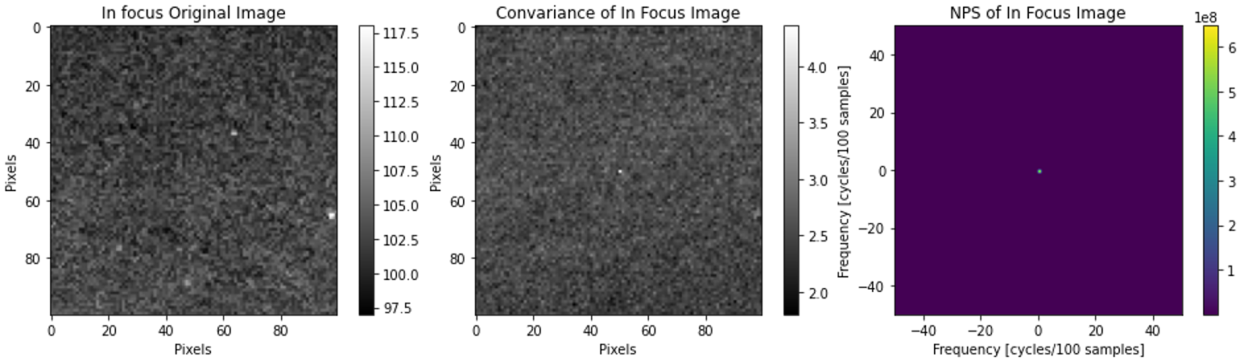


Figure 5: Original, Covariance, and NPS of 100x100 AOIs of In-Focus Image

Left: 100x100 AOIs of the original in-focus image. Middle: Covariance of 100x100 AOIs of in-focus image. Right: NPS of 100x100 AOIs of in-focus image.

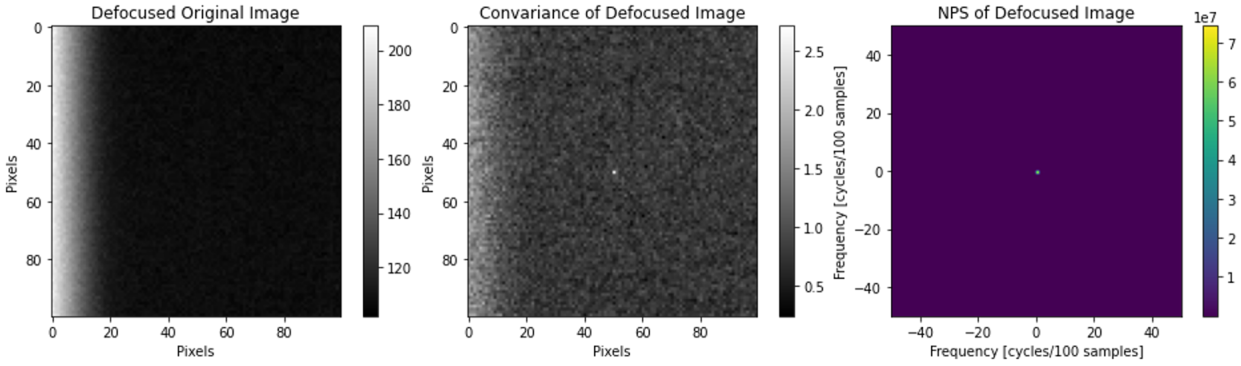


Figure 6: Original, Covariance, and NPS of 100x100 AOIs of Defocused Image

Left: 100x100 AOIs of the original defocused image. Middle: Covariance of 100x100 AOIs of defocused image. Right: NPS of 100x100 AOIs of defocused image.

Similarly, Figures 5 and 6 show the 100x100 AOIs of captured images in focused and defocused conditions with their covariance and noise power spectrum for the center pixel. The intensity of the defocused image is higher than the intensity of the in-focused image. We could notice a white gradient region on the left of the defocused image. We thought this happened because when we picked the same region, the camera was slightly tilted when we changed the focus physically. The covariances of the center pixel to the rest of the in-focused and defocused images range from 1.5 to 4.5 and 0 to 3. The NPS of the center pixel of the in focus image reaches 4.5×10^8 and defocused image reaches 4.5×10^7.

Experiment 2: Measuring Spatial Resolution Properties of the Imaging System by Calculating the Modulation Transfer Function

The three different image targets, the line pair target in Figure 7 A-C, the star target in Figure 7 D-F, and AOIs of the knife edge target in Figure 7 G-I, were imaged at different levels of focus. Each image target was taken as focused images, where the 2 lp/mm and 10 lp/mm were distinguishable, defocused where the 8 lp/mm line pairs were blurred, and defocused where the 4 lp/mm line pairs were blurred. With increasing blur across all the targets, the spatial resolution decreased, as the edges between light and dark objects became blurred, and the difference between high spatial frequency patterns became harder to distinguish.

It's important to point out that in spite of using the same imaging conditions, there is significantly less light in the focused line pair target (Figure 7A) and the focused star target (Figure 7D). We believe this was due to changing light conditions in the room, possibly our team shifting positions around the imaging set up. We are confident this change does not significantly affect our MTF Calculations or the rest of our analysis.

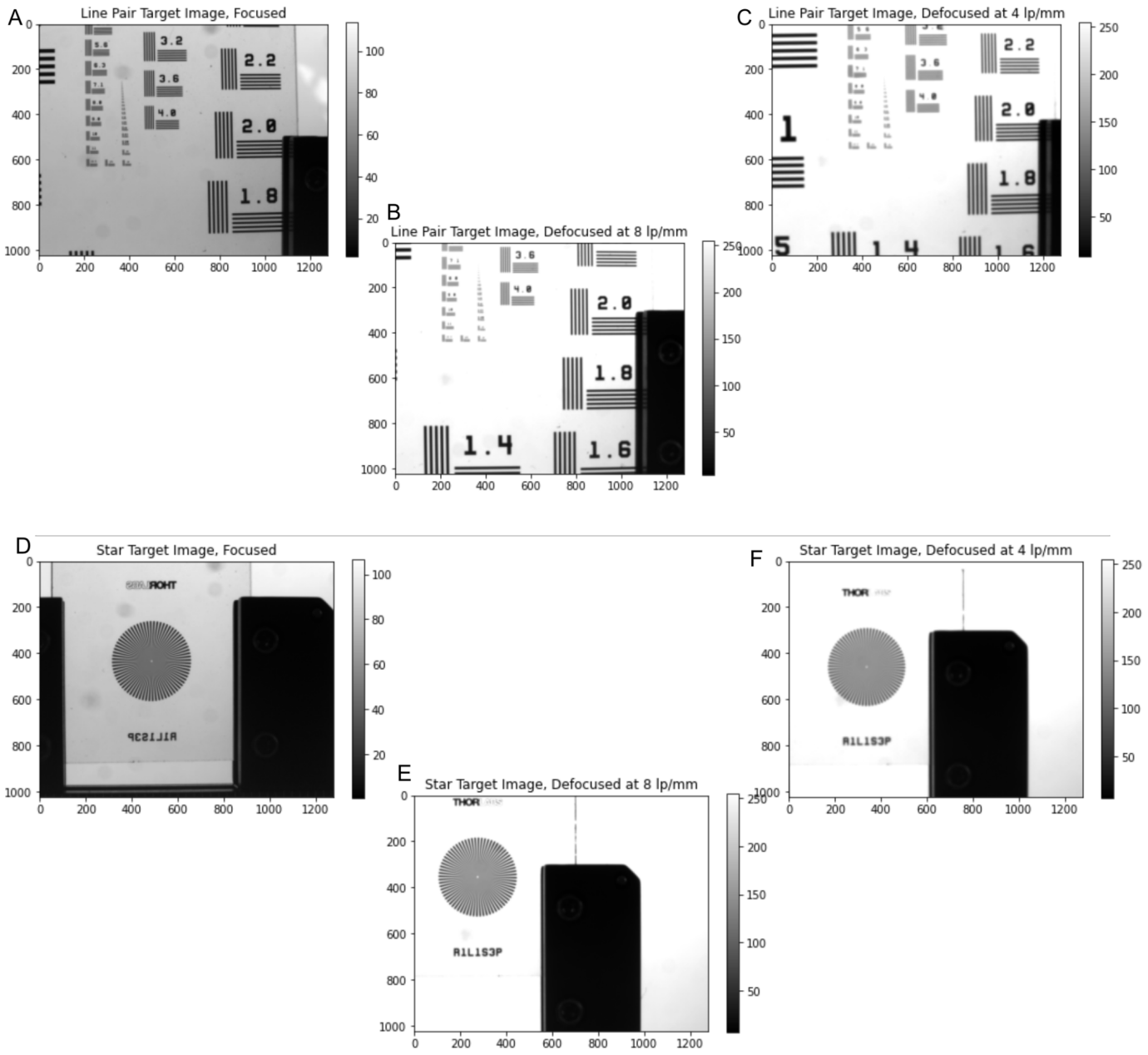


Figure 7: Line Pair Target, Star Target, and Knife Edge Target Imaged at Different Focuses

A) Line Pair target imaged at optimal focus, B) defocused so the 8 lp/mm line pairs were blurry, and C) defocused so the 4 lp/mm line pairs were blurry. D) Star target imaged at optimal focus, E) defocused so the 8 lp/mm line pairs were blurry, and F) defocused so the 4 lp/mm line pairs were blurry. G) Knife Edge target imaged at optimal focus, H) defocused so the 8 lp/mm line pairs were blurry, and I) defocused so the 4 lp/mm line pairs were blurry.

In Figure 8, the steps of the MTF calculation are shown. This includes the initial knife edge image, the Edge Spread Function (ESF) with the data unbinned and binned, the Line Spread Function (LSF) zoomed in to a smaller range on the x-axis, and finally the Modulation Transfer Function (MTF) of the system under these imaging conditions. The ESF shows the pixels to the left of the edge are almost all highest intensity white pixels, and almost all the pixels to the right are lowest intensity dark pixels, with a steep transition at the edge. The LSF is the derivative of the ESF, and it represents relatively constant pixel intensity on either side of the edge, with a sharp transition between directly left and right of the edge.

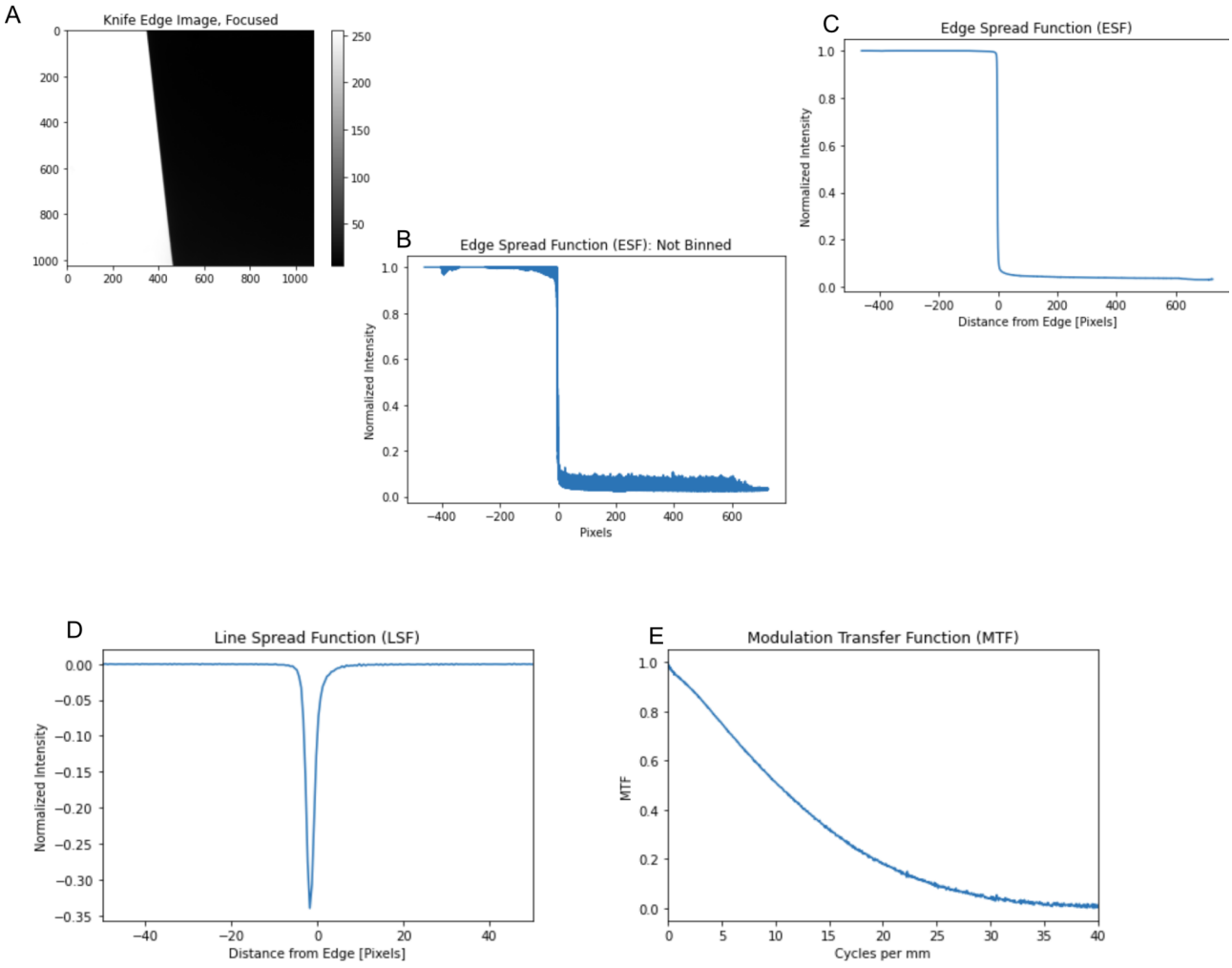
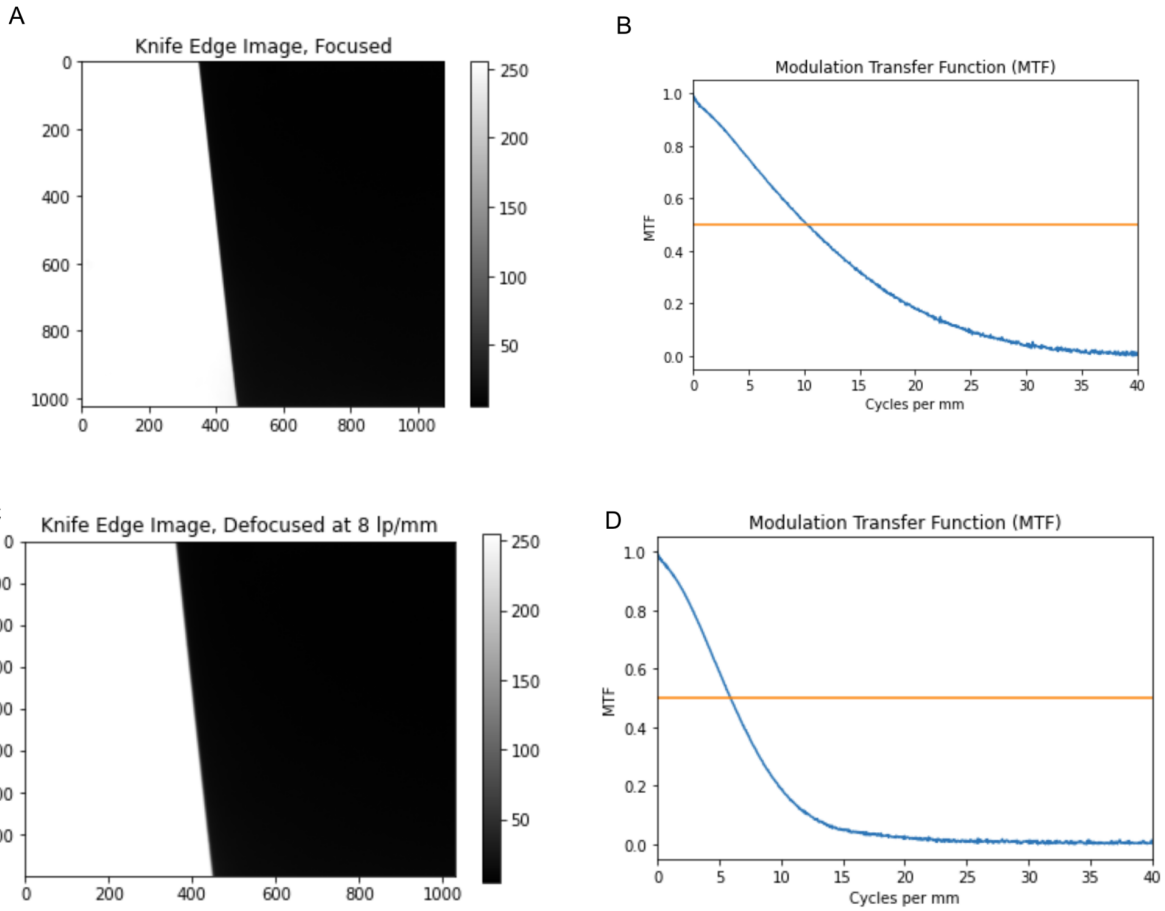


Figure 8: MTF Calculation Process with Focused Knife Edge Image

A) AOI of focused knife edge image. B) Edge Spread Function (ESF) before binning process. C) ESF with binning of 2 bins per pixel. D) Line Spread Function (LSF) and E) Modulation Transfer Function (MTF) of the focused knife edge image.

Figure 9 shows the MTF calculated on a focused knife edge image, the knife edge defocused where the 8 lp/mm line pairs are blurred, and defocused where the 4 lp/mm line pairs are blurred. As the system becomes less focused, the MTF drops off at a steeper slope and approaches 0 at a lower spatial frequency, of cycle per mm. This represents what spatial frequency the imaging system can no longer distinguish. Specifically, the focused image drops to 0 at around 20 cycles/mm, the unfocused at 8 lp/mm drops to 0 around 8 cycles/mm, and the unfocused at 4 lp/mm drops to 0 around 4 lp/mm.



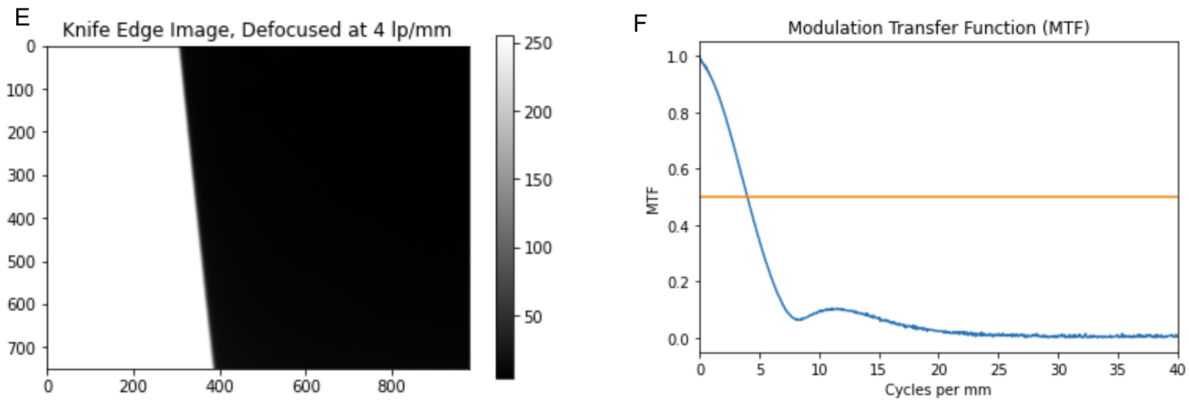


Figure 9: Knife Edge Images at Various Focus Levels, and their Respective MTFs

A,B) Focused knife edge image and its calculated MTF. C,D) Knife edge image defocused so 8 lp/mm line pairs were defocused and its calculated MTF. E,F) Knife edge image defocused so 4 lp/mm line pairs were defocused and its calculated MTF. On the MTF graphs (B, D, F), the yellow line represents MTF = 50%.

Experiment 3: Obtaining the Sharpest Possible Image of the Star Target

Figure 10 shows the sharpest image of the star target (Figure 10A) and the line pair target (Figure 10B) captured during experimentation. This image was taken with the following parameters: Aperture = 8 , Focus = N/A (Hardware is not able to shown) , exposure = 25 ms, pixel_clock = 7MHz, and distance between imaging targets and front of the camera = 5”. The sharpest image was determined by adjusting aperture, exposure, and focus settings where at the smallest spatial frequency, or closest to the center of the target where the lines were closest together, the lines were still visible as separate lines. As the focus increased, the visible lines closest to the center came into view, but appeared to curve and connect. This interesting visual phenomenon is discussed further in the discussion section.

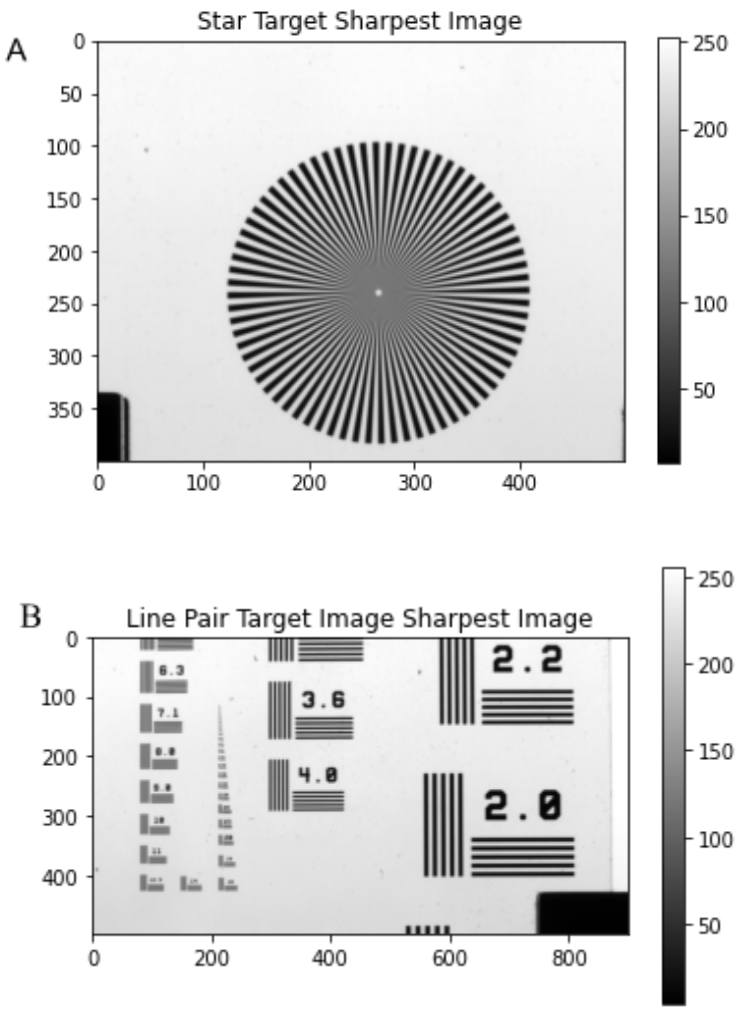


Figure 10: **Sharpest possible star image**

A) Sharpest possible image taken of the star target B) Image of the line pair target taken at the same conditions as the sharpest star target image.

Discussion

Experiment 1: Measuring Spatial Covariance in the Imaging System

From the last lab, we learned that when we captured dark images with caps on, we analyzed the dark noise in the imaging system. The Noise Power Spectrum is a standard measure for the noise of image capture systems. It is derived from captured uniform luminance patches. From the middle image of Figures 1 and 2, we can observe that the center of the covariance image is much brighter than the rest of the image because we picked the center pixel as our

reference to calculate the covariance. The pattern of the rest of the image is uniformly distributed, and the covariance ranges from -0.05 to 0.2 with the center pixel. The Noise Power Spectrum is the Fourier transform of the covariance, which refers to how different frequencies of noise can propagate through a system. The pattern of the intrinsic noise indicates that the center noise power is more pronounced at low-frequency. The reason is that the common sources of intrinsic noise (thermal noise, read noise, dark current, etc.) can produce low-frequency noise as shown in the figures.

For Figures 3 - 6, we collected a uniform patch target with the lens caps off. There are more incident lights hitting the sensor, resulting in more noise signals, as discovered in Lab 1. When we compared the right images of Figure 5 and Figure 6, we could observe that the NPS of the in focus image is higher than the NPS of the defocused image. From this comparison of the color bars, the sharper the image, the more energy is concentrated in the low-frequency components of the NPS. This property is meaningful because the degree of blurriness in an image can impact the spatial distribution of noise in the image. The noise of an in-focus image is mostly a low-frequency variation of the image intensity. On the other hand, defocused images have lower energy in the low-frequency components of NPS.

Compared with intrinsic noise, the properties of signal noise are slightly different quantitatively. The intrinsic noise only reaches 25000, but the noise in both in focus and defocused images reaches 4.5×10^8 and 4.5×10^7, which are much larger than the intrinsic noise. This difference shows that the signal noise impacts the distribution of noise dominantly. As a result, the focus can change the spatial noise properties. Theoretically, when the image is more defocused, it tends to have more energy in the high-frequency components and less energy in the low-frequency components.

Another interesting feature that we found is that the intensity of noise is smaller for 60x60 AOIs compared to that for 100x100 AOIs. This finding caught our attention, so we also collected 60x60 AOIs for a uniform patch scene. Based on the right images in Figure 3 and Figure 4, we could see the NPS of in-focus and defocused images reach 5×10^7 and 3×10^7. So another property could be generalized as the AOI sizes increase, the spatial frequency resolution of the NPS also increases, because the sampling of the NPS depends on the selected AOI sizes.

Experiment 2: Measuring Spatial Resolution Properties of the Imaging System by Calculating the Modulation Transfer Function

To assess the spatial resolution properties of the imaging system, various targets were imaged with different amounts of blur in the system, and the MTF was calculated using the ESF method with varied amounts of blur in the edge image. The width of the MTF is widest for the focused image, and decreases as the edge image becomes less focused and more blurred. When focused, the 50% maximum spatial frequency that is passed by the system is around 10 cycles/mm, and it decreases as the system becomes less focused and more blurred (8 cycles/mm when blurred at 8 lp/mm, and 4 cycles/mm when blurred at 4 lp/mm). A small lobe can be seen in the MTF of the defocused to 4 lp/mm image, where there is a slight peak in MTF value at around 12 cycles/mm (Figure 7F). This is because for the least focused image, the blurred transition at the edge causes a slight increase in the spatial frequencies that can be distinguished. A highly focused image mimics an impulse function, and a less focused image has lobes surrounding the impulse.

The line pair and star target results are related to the calculated MTF. The spatial frequency that we have blurred to is the spatial frequency (cycles/mm) is the 50% maximum

spatial frequency. It represents where the imaging system can only distinguish 50% of the details at that spatial frequency, or equivalently where the image begins to blur. Therefore, when the MTF is at 50% at around 8 cycles/mm, this correlates to the spatial frequency of 8 lp/mm blurring on the line pair target. And while it isn't marked on the star pattern, it's clear that the lines of the star pattern become indistinguishable when the spatial frequency is around 8 lp/mm or 8 cycles/mm.

Experiment 3: Obtaining the Sharpest Possible Image of the Star Target

The sharpest possible star and line pair images were better compared to the part 2 images. At the sharpest setting, one could resolve the 3.2 lp/mm line pairs, but in the part 2 images, the 3.6 lp/mm line pairs represented the limiting resolution. The limiting resolution for part 2 can be estimated around 3.0 lp/mm, since key line pairs were cut off in the image. More qualitatively, in the star image, the separate lines can be distinguished further into the center in part 3 than part 2. As mentioned above, at the highest frequency, the lines in the star pair image appear to curve. This means that although a viewer might think the image is resolved at the highest spatial frequency, there is light being reflected to the wrong point on the detector, so the out of focus center becomes an optical illusion.

Main Points learned in this lab:

- NPS is a useful image quality metric for understanding the noise distribution in an image.
- The focus and AOIs size impact the NPS of images.
- MTF is the spatial frequency response of an imaging system. It can be calculated from an Edge Spread Function, and is useful in assessing the spatial resolution and contrast of an image over a range of spatial frequencies.

- The spatial frequency at which the spatial resolution begins to fall off (or features begin to blur) is where the MTF of the system is roughly 50%

Works Cited

[1] "Introduction to Modulation Transfer Function | Edmund Optics." https://www.edmundoptics.com/knowledge-center/application-notes/optics/introduction-to-modulation-transfer-function/ (accessed Feb. 13, 2023).

[2] S. Dolly, H.-C. Chen, M. Anastasio, S. Mutic, and H. Li, "Practical considerations for noise power spectra estimation for clinical CT scanners," *J. Appl. Clin. Med. Phys.*, vol. 17, no. 3, pp. 392–407, 2016, doi: 10.1120/jacmp.v17i3.5841.

Lab 2:

The main purpose of Lab 0 is to familiarize yourselves with the camera. You should be able to capture an image, establish a basic understanding of camera parameters and adjust them in live image display to obatin the best image possible

Import basic Python modules

```
# Required imports
import sys
import os
from PIL import Image
import numpy as np
import time
import matplotlib.pyplot as plt
import cv2
# from instrumental.drivers.cameras import uc480
import scipy.io as sio

import cv2
import scipy.stats as stats
from scipy.interpolate import interp1d
from sklearn.linear_model import LinearRegression
import plotly.express as px
import skimage.io as skio


from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Camera object instantiation

```
# initialize camera
# lists the cameras that are detected.
instruments = uc480.list_instruments()
# initiliaze the camera using the detected cameras.
cam = uc480.UC480_Camera(instruments[0])
# Open Camera. DONT FORGET TO CLOSE!
cam.open()
```

Check camera properties, change Camera settings and grab an image.

```
# Connect to the camera and set up the image memory
cam.open()
# view current camera params.
cam.get_parameters()
```

```
Exposure Time: 66.62091228070176ms
Exposure Time Range: [0.008982456140350877 66.62091228070176]
Framerate: 14.997947649269046Hz
Pixel Clock: 24MHz
Gain: 1.0
```

Set Framerate

```
# set frame rate
cam.set_framerate(framerate = "7Hz")
# get exposure range, you can see that the frame rate will change the maximum exposure time.
print(cam._get_exposure_range())
# print current frame rate
print(cam.framerate)
```

```
(<Quantity(0.00898245614, 'millisecond')>, <Quantity(142.790456, 'millisecond')>)
7.000578714507065 hertz
```

Set Exposure Time

```
# get exposure range
cam._get_exposure_range()
"""
(0.008982456140350877 <Unit('millisecond')>,
 142.79045614035087 <Unit('millisecond')>)
"""

# set cam exposure (ms)
cam._set_exposure('25 ms')
cam._get_exposure()
```

```
24.999797750787227 millisecond
```

Setting Pixel Clock

```
# Set pixelclock (Use the default)
cam.set_pixelclock(pixel_clock='7MHz')

# show the current pixel clock.
cam.pixelclock
```

```
7 megahertz
```

Setting Gain

```
# The master gain factor; 1.0 is the lowest gain, max gain is 4.0
# Use the default gain.
cam._set_gain(30)

# get max gain
cam.max_master_gain

# get current gain.
cam.master_gain
```

```
1.92
```

```
cam.get_parameters()
```

```
Exposure Time: 85.50282844836643ms
Exposure Time Range: [0.008982456140350877 488.26301754385963]
Framerate: 2.0472896614601814Hz
Pixel Clock: 7MHz
Gain: 1.92
```

```
print(cam._width)
```

```
1280
```

```
print(cam._height)
```

```
1024
```

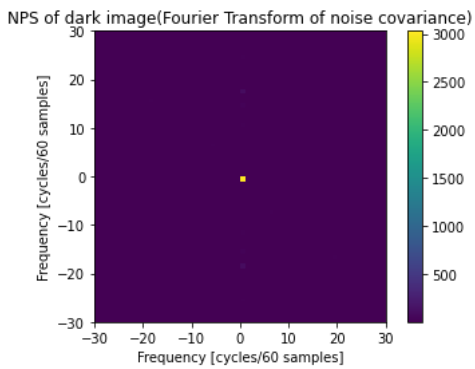
```
x0, y0 = 0, 0
# this sets the width and height of the rectangle.
width, height = 1280, 1024
cam._set_AOI(x0, y0, width, height)
cam._get_AOI()
```

```
(0, 0, 1280, 1024)
```

```
## Part 1
# Taking 300 frames dark images at 100x100 AOI to calculate covariance and noise spectrum
x0, y0 = 0, 0
# this sets the width and height of the rectangle.
width, height = 1280, 1024
cam._set_AOI(x0, y0, width, height)
cam._get_AOI()

x = np.zeros((300,100,100))
for i in range(300):
    print(i)

    #take an image
```

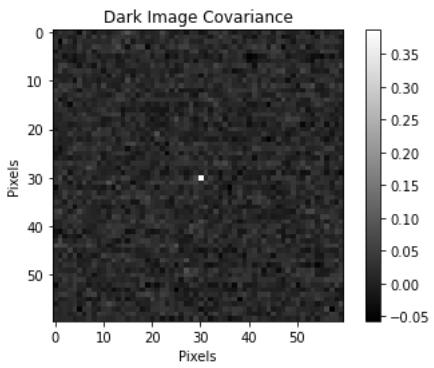
```
    img = cam.grab_image().copy()

    x[i,:,:] = img[200:300, 200:300]
# average = np.mean(avg,axis = 0)
# variance = np.var(avg, axis = 0)

sio.savemat("300framesAOI_part1.mat", {"Images":x})


x = sio.loadmat("/content/drive/MyDrive/Imaging_Instrumentation_Labs/Lab 2/300framesAOI_part1.mat")["Images"]


# x = x[:,0:60,0:60]


## Part 1
# Calculate covariance from 300 frames
'''
x: center image pixel for one frame
mean x: mean of center image pixel for 300 frames
y:  image pixels for one frame
mean y: mean of image pixels for 300 frames
'''
y = x
y_avg = np.mean(y, axis = 0)
y_minus_yavg = np.zeros((x.shape[0], x.shape[1], x.shape[2]))
x_minus_xavg = np.zeros((x.shape[0],1,1))
for i in range(x.shape[0]):
    y_minus_yavg[i,:,:] = y[i,:,:] - y_avg[:,:]
    x_minus_xavg[i] = y[i,30,30] - y_avg[30,30]
product = np.sum(np.multiply(x_minus_xavg ,y_minus_yavg),axis = 0)
covariance = product/299


# Part 1: Find NPS of dark image
# FFT2 of covariance, and fft shift
nps_focused = np.abs(np.fft.fftshift(np.fft.fft2(covariance)))**2
plt.imshow(nps_focused,extent = (-30,30,-30,30))
plt.xlabel("Frequency [cycles/60 samples]")
plt.ylabel("Frequency [cycles/60 samples]")
plt.title("NPS of dark image(Fourier Transform of noise covariance)")
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x7f34b5b83070>
```



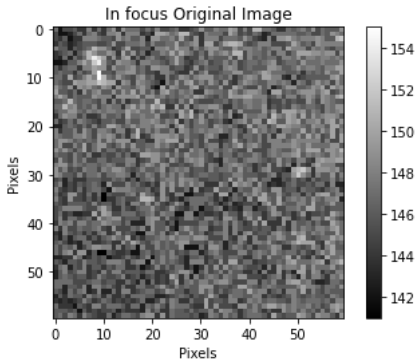

```
f = plt.figure()
plt.xlabel("Pixels")
plt.ylabel("Pixels")
plt.imshow(covariance, cmap = 'gray')
plt.title('Dark Image Covariance')
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x7f34caebd820>
```



```
## Part 1
# Taking 300 frames light images at 100x100 AOI to calculate covariance and noise spectrum
x0, y0 = 0, 0
# this sets the width and height of the rectangle.
# Focus: 0.4
# Aperature: 6

width, height = 1280, 1024
cam._set_AOI(x0, y0, width, height)
cam._get_AOI()

x2 = np.zeros((300,100,100))
for i in range(300):
    print(i)
    #take an image
    img = cam.grab_image().copy()
    x2[i,:,:] = img[200:300,400:500]


## Part 1
# Out of focus
# Focus:
# Aperature: 6
cam._set_AOI(x0, y0, width, height)
x3 = np.zeros((300,100,100))
for i in range(300):
    print(i)
    #take an image
    img = cam.grab_image().copy()
    x3[i,:,:] = img[200:300,400:500]



sio.savemat("300framesAOIlight_focused_part1_Feb13.mat", {"Images":x2})
sio.savemat("300framesAOIlight_unfocused_part1_Feb13.mat", {"Images":x3})


x2 = sio.loadmat("/content/drive/MyDrive/Imaging_Instrumentation_Labs/Lab 2/300framesAOIlight_focused_part1.mat")["Images"]
x3 = sio.loadmat("/content/drive/MyDrive/Imaging_Instrumentation_Labs/Lab 2/300framesAOIlight_unfocused_part1.mat")["Images"]


# x2 = sio.loadmat("/content/drive/MyDrive/Imaging_Instrumentation_Labs/Lab 2/300framesAOIlight_focused_part1.mat")["Images"]
x2 = x2[:,0:60,0:60]
f = plt.figure()
plt.xlabel("Pixels")
plt.ylabel("Pixels")
plt.imshow(x2[1,:,:], cmap = 'gray')
plt.title('In focus Original Image')
plt.colorbar()
```

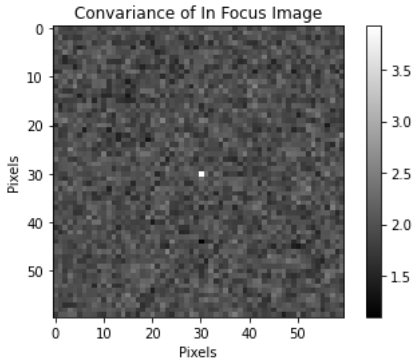
```
<matplotlib.colorbar.Colorbar at 0x7f726234e7c0>
```




```
## Part 1: Find covariance and NPS of focused and unfocused image
## Convariance and NPS of Focused image
y_focused = x2.astype('float')
y_focused_avg = np.mean(y_focused, axis = 0)
y_minus_yavg_focused = np.zeros((x2.shape[0], x2.shape[1], x2.shape[2]))
x_minus_xavg_focused = np.zeros((x2.shape[0],1,1))
for i in range(x2.shape[0]):
    y_minus_yavg_focused[i,:,:] = y_focused[i,:,:] - y_focused_avg[:,:]
    x_minus_xavg_focused[i] = y_focused[i,30,30] - y_focused_avg[30,30]
product_focused = np.sum(np.multiply(x_minus_xavg_focused ,y_minus_yavg_focused),axis = 0)
covariance_focused = product_focused/299


f = plt.figure()
plt.xlabel("Pixels")
plt.ylabel("Pixels")
```

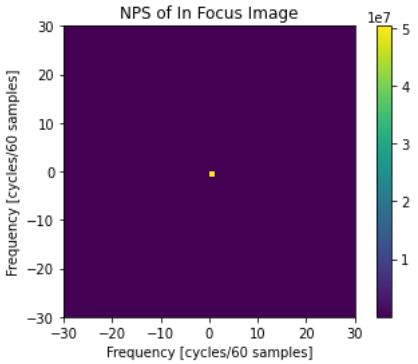
```
plt.imshow(covariance_focused, cmap = 'gray')
plt.title('Convariance of In Focus Image')
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x7f72622658b0>
```



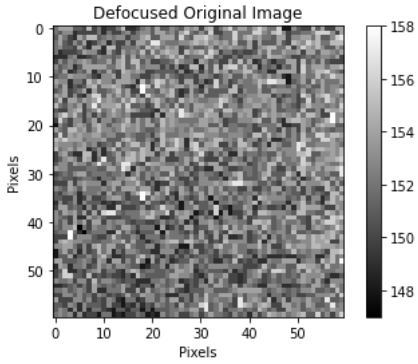

```
##To meansure NPS
##image->2D FFT -> low frequency detrending -> Data reduction -> 1D NPS -> characteristic curve -> 1D NPS interms of fluctuati
nps_focused = np.abs(np.fft.fftshift(np.fft.fft2(covariance_focused)))**2
f = plt.figure()
plt.imshow(nps_focused,extent = (-30,30,-30,30))
plt.xlabel("Frequency [cycles/60 samples]")
plt.ylabel("Frequency [cycles/60 samples]")
plt.title("NPS of In Focus Image")
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x7f72621faa60>
```




```
# x3 = sio.loadmat("/content/drive/MyDrive/Imaging_Instrumentation_Labs/Lab 2/300framesAOIlight_unfocused_part1.mat")["Images"
x3 = x3[:,0:60,0:60]
f = plt.figure()
plt.xlabel("Pixels")
plt.ylabel("Pixels")
plt.imshow(x3[200,:,:], cmap = 'gray')
plt.title('Defocused Original Image')
plt.colorbar()
```

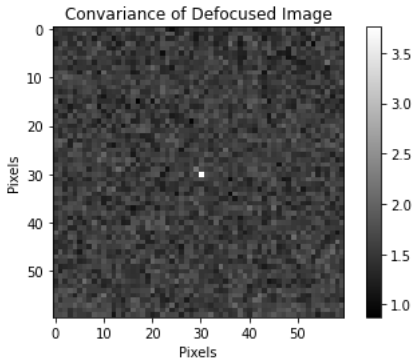
```
<matplotlib.colorbar.Colorbar at 0x7f7262191250>
```




```
## Convariance and NPS of DeFocused image
y_Defocused = x3.astype('float')
y_Defocused_avg = np.mean(y_Defocused, axis = 0)
y_minus_yavg_Defocused = np.zeros((x3.shape[0], x3.shape[1], x3.shape[2]))
x_minus_xavg_Defocused = np.zeros((x3.shape[0],1,1))
for i in range(x3.shape[0]):
    y_minus_yavg_Defocused[i,:,:] = y_Defocused[i,:,:] - y_Defocused_avg[:,:]
    x_minus_xavg_Defocused[i] = y_Defocused[i,30,30] - y_Defocused_avg[30,30]
```

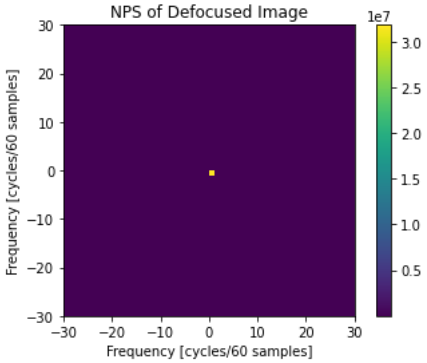
```
product_Defocused = np.sum(np.multiply(x_minus_xavg_Defocused ,y_minus_yavg_Defocused),axis = 0)
covariance_Defocused = product_Defocused/299

f = plt.figure()
plt.xlabel("Pixels")
plt.ylabel("Pixels")
plt.imshow(covariance_Defocused, cmap = 'gray')
plt.title('Convariance of Defocused Image')
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x7f72620ad0a0>
```




```
##To meansure NPS
##image->2D FFT -> low frequency detrending -> Data reduction -> 1D NPS -> characteristic curve -> 1D NPS interms of fluctuati
nps_defocused = np.abs(np.fft.fftshift(np.fft.fft2(covariance_Defocused)))**2
f = plt.figure()
plt.imshow(nps_defocused,extent = (-30,30,-30,30))
plt.xlabel("Frequency [cycles/60 samples]")
plt.ylabel("Frequency [cycles/60 samples]")
plt.title("NPS of Defocused Image")
plt.colorbar()
```

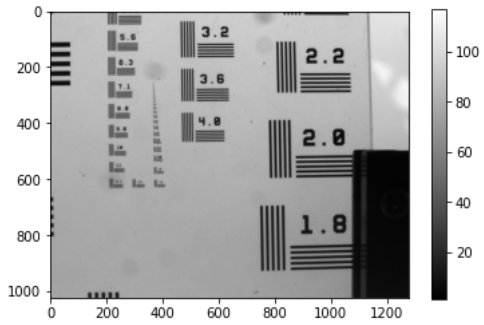
```
<matplotlib.colorbar.Colorbar at 0x7f7262037700>
```




```
## Part 2: 50 frames line target
x0, y0 = 0, 0
# this sets the width and height of the rectangle.


width, height = 1280, 1024
cam._set_AOI(x0, y0, width, height)
cam._get_AOI()

x11 = np.zeros((50,1024,1280))
for i in range(50):
    print(i)

    #take an image
    img = cam.grab_image().copy()

    x11[i,:,:] = img

average_line_defocus8 = np.mean(x11, axis = 0)



sio.savemat("50frames_lineTargetDefocus8_part2.mat", {"Images":x11, "Average":average_line_defocus8})


## Part 2: 50 frames star target
x0, y0 = 0, 0
# this sets the width and height of the rectangle.
```

```
width, height = 1280, 1024
cam._set_AOI(x0, y0, width, height)
cam._get_AOI()

x12 = np.zeros((50,1024,1280))
for i in range(50):
    print(i)

    #take an image
    img = cam.grab_image().copy()

    x12[i,:,:] = img

average_star_defocus8 = np.mean(x12, axis = 0)


sio.savemat("50frames_starTarget_defocus8_part2.mat", {"Images":x12, "Average":average_star_defocus8})


## Part 2: 50 frames knife target (repeated at different angles)
x0, y0 = 0, 0
# this sets the width and height of the rectangle.


width, height = 1280, 1024
cam._set_AOI(x0, y0, width, height)
cam._get_AOI()

x13 = np.zeros((50,1024,1280))
for i in range(50):
    print(i)

    #take an image
    img = cam.grab_image().copy()

    x13[i,:,:] = img

average_knife_defocus8 = np.mean(x13, axis = 0)


sio.savemat("50frames_knifeTarget_defocus8_part2.mat", {"Images":x13, "Average":average_knife_defocus8})


sio.savemat("50frames_knifeTarget_defocus8_angle2_part2.mat", {"Images":x14, "Average":average_knife2_defocus8})


line_focus = sio.loadmat("50frames_lineTarget_part2.mat")["Images"]
line_focus_averaged = sio.loadmat("50frames_lineTarget_part2.mat")["Average"]


plt.figure()
plt.imshow(line_focus[5,:,:],cmap = "gray")
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x2bb430f8f28>
```




```
knife_focus = sio.loadmat("50frames_knifeTarget_angle2_part2.mat")["Images"]
knife_focus_averaged = sio.loadmat("50frames_knifeTarget_angle2_part2.mat")["Average"]


plt.figure()
plt.imshow(knife_focus[5,:,:],cmap = "gray")
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x2bb43193ba8>
```

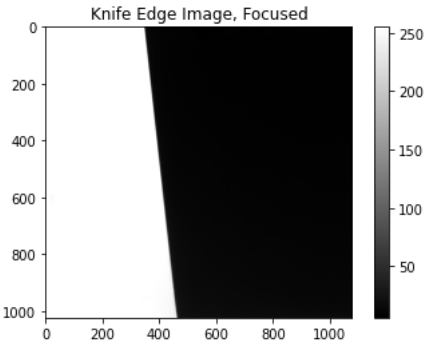
```
import cv2
import scipy.stats as stats
from scipy.interpolate import interp1d
from sklearn.linear_model import LinearRegression
```

```
knife_focus = sio.loadmat('/content/drive/My Drive/50frames_knifeTarget_angle2_part2.mat')["Average"] # Maybe crop differently
#linePair = sio.loadmat('/content/drive/My Drive/50frames_knifeTarget_angle2_part2.mat')["Average"])
# Line pair equivalent to cycles, spatial frequency
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount
```

```
plt.figure()
plt.imshow(knife_focus[:,200:-1], cmap = "gray")
plt.colorbar()
plt.title("Knife Edge Image, Focused")
```

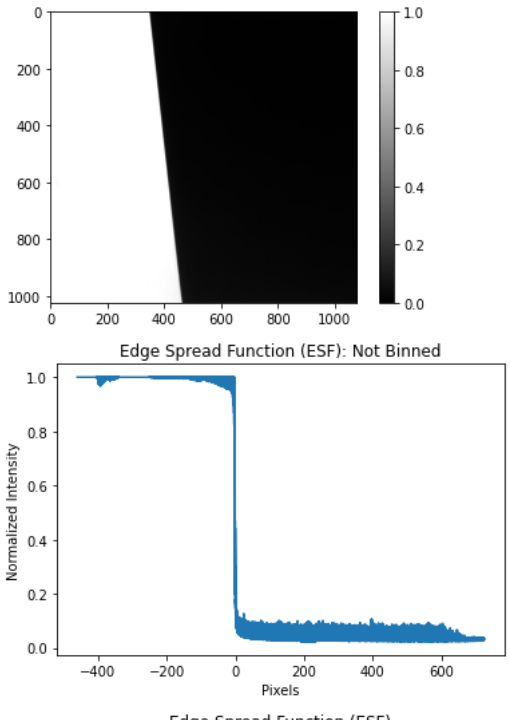
```
Text(0.5, 1.0, 'Knife Edge Image, Focused')
```



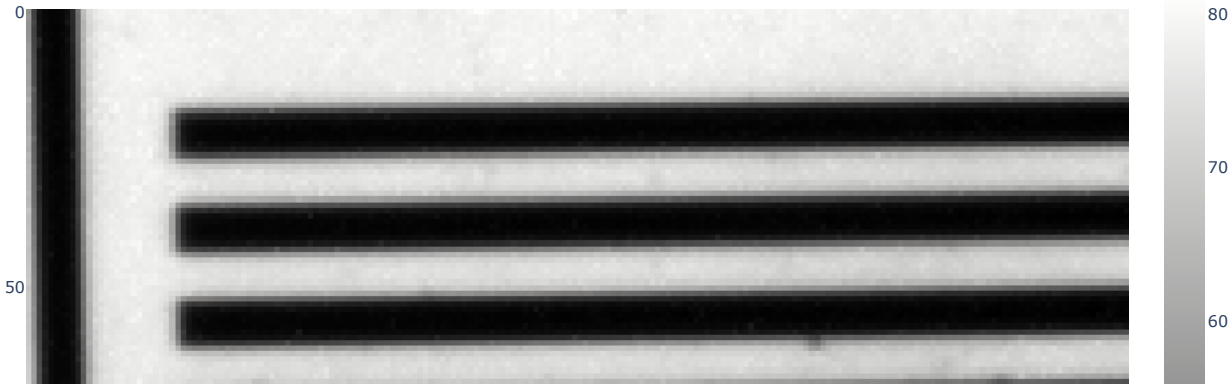

```
## Part 2: Calculating MTF: Focused Knife edge image

# 1) Edge detection to find pixels along line (pixel closest to 50%, normalize)
# Average images

# Normalization
t = 50
# Crop first 200 columns
binary_mask = np.array(1*(knife_focus[:,200:-1] > t))
knife_focus2 = knife_focus[:,200:-1]
plt.imshow(binary_mask,cmap="gray")
plt.colorbar()
print(binary_mask[:,340:500])
# Avg
linePixels_x_axis = np.zeros((1024))
linePixels = np.zeros((2,1024))

for i in range(1024):
    #Going through each row, save the y index for the first time the value is 1
    #print(np.where(binary_mask[i,:] == 1)[0][0])
    #linePixels_x_axis[i] = np.where(binary_mask[i,:] == 1)[0][0]

    linePixels[0,i] = (i)
    linePixels[1,i] = np.where(binary_mask[i,:] == 0)[0][0]


#plt.imshow(binary_mask); plt.plot(linePixels[1], linePixels[0])

# 2) Use pixels along line to get y = mx+b equation of line (point slope form); linear fit function (whole row at once, min di
#    Linear interpolation to get line? https://realpython.com/linear-regression-in-python/
xVals = linePixels[1,:].reshape((-1, 1))
yVals = linePixels[0,:]
#print(xVals[:150])
#print(yVals[:150])
model = LinearRegression().fit(xVals, yVals)
print(f"intercept: {model.intercept_}")

print(f"slope: {model.coef_}")
```

```
# 3) Find distance, D, from line to each pixel to the left (perpendicular distance formula) (all computed simultaneously), cre
#    Use cropped image? nothing left of pixel 200 https://stackoverflow.com/questions/57063442/measuring-distance-from-points-
a = model.coef_
b = -1
c = model.intercept_

x = np.arange(0, knife_focus2.shape[1])
y = np.arange(0, knife_focus2.shape[0])

X, Y = np.meshgrid(x,y)

D = (a*X + b*Y + c)/(np.sqrt(a**2 + b**2))

plt.imshow(knife_focus2, cmap = "gray")

# 4) Plot sorted D vs I (intensity of those pixels) --> Edge Spread Function
D_sort1 = np.argsort(D.ravel())
D_sort2 = np.take_along_axis(D.ravel(), D_sort1, axis = 0)
intensity = np.take_along_axis(knife_focus2.ravel(), D_sort1, axis = 0)/np.max(knife_focus2)

plt.figure()
plt.plot(D_sort2, intensity)
plt.title("Edge Spread Function (ESF): Not Binned")
plt.xlabel("Pixels")
plt.ylabel("Normalized Intensity")
# 5) Bin D vs I plot so it's uniformly sampled (loop), delta d smaller than pixel size
numDistances = np.round(D_sort2[-1] + np.abs(D_sort2[0]))
numBins = 2*numDistances
# 2 bins per pixel
binnedIntensity, binEdges, binNumber = stats.binned_statistic(D_sort2, intensity, 'mean', numBins)
# Check to make sure there's at least one point in every bin
if(1 in np.isnan(binnedIntensity)):
  print('Bins Too Small')
else:
  print('Great Bins')

binEdges = binEdges[1:]
plt.figure()
plt.plot(binEdges, binnedIntensity)
plt.title("Edge Spread Function (ESF)")
plt.xlabel("Distance from Edge [Pixels]")
plt.ylabel("Normalized Intensity")

# 6) Differentiate ESF to get LSF (w filter) (can't have bins w no samples in them)
binSize = binEdges[1] - binEdges[0]
LSF = np.diff(binnedIntensity)/binSize
LSF_x = binEdges[1:]
plt.figure()
plt.plot(LSF_x, LSF)
plt.xlim([-50,50])
plt.title("Line Spread Function (LSF)")
plt.xlabel("Distance from Edge [Pixels]")
plt.ylabel("Normalized Intensity")

# 7) FFT of LSF --> MTF cycles/mm

MTF = np.abs(np.fft.fft(LSF))/np.max(np.abs(np.fft.fft(LSF)[:1000]))
pix = numBins*binSize #bins * pixels/bin
mm = pix * 0.5/34 # pixels * mm/pixel
MTF_x = np.arange(0, len(MTF))/mm # Cycles per mm
half = np.zeros(len(MTF_x))
half[:] = 0.5
plt.figure()
plt.plot(MTF_x, MTF); plt.plot(MTF_x, half)
plt.xlim([0,40])
plt.title("Modulation Transfer Function (MTF)")
plt.xlabel("Cycles per mm")
plt.ylabel("MTF")
```

```
[[1 1 1 ... 0 0 0]
 [1 1 1 ... 0 0 0]
 [1 1 1 ... 0 0 0]
 ...
 [1 1 1 ... 0 0 0]
 [1 1 1 ... 0 0 0]
 [1 1 1 ... 0 0 0]]
intercept: -3117.895496984186
slope: [8.93007557]
Great Bins
Text(0, 0.5, 'MTF')
```



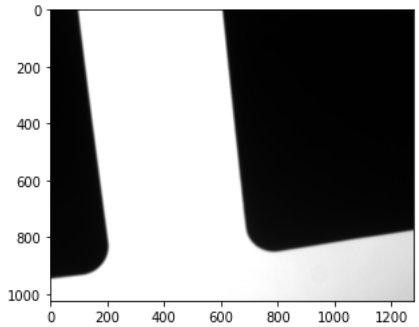

```
line_focus = sio.loadmat('/content/drive/My Drive/50frames_lineTarget_part2.mat')["Average"] # Maybe crop differently

## Part 2: Finding number of pixels per mm in our images using the line pair targets

line_focus = line_focus.astype('float64')

px.imshow(line_focus[500:700, 850:1050], color_continuous_scale = "gray", width = 1000, height = 1000) # Rows 500 to 700, colu
# 34 pixels per mm
```




```
knife_defocus4 = sio.loadmat('/content/drive/My Drive/50frames_knifeTarget_defocus4_angle2_part2.mat')["Average"]
```

```
plt.figure()
plt.imshow(knife_defocus4, cmap = "gray")
```

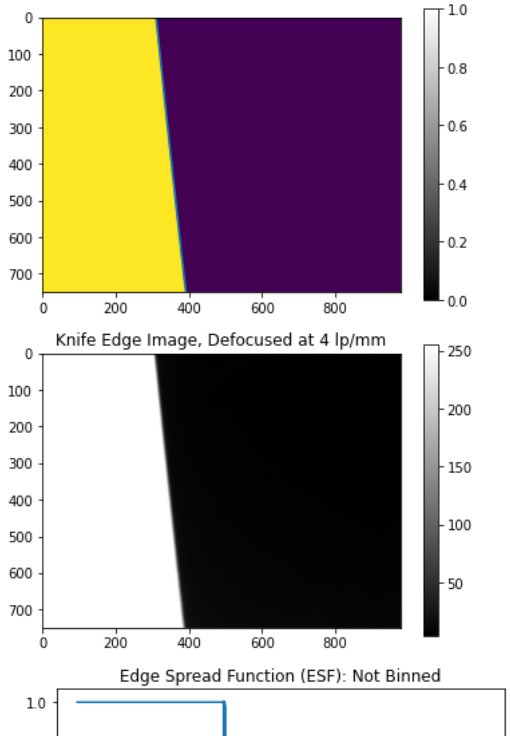
```
<matplotlib.image.AxesImage at 0x7ff0a557a130>
```




```
## Part 2: Calculating MTF: Defocused to the 4 line pairs/ mm Knife edge image

# 1) Edge detection to find pixels along line (pixel closest to 50%, normalize)
# Average images

# Normalization
t = 50
# Crop first 200 columns
binary_mask = np.array(1*(knife_defocus4[:750,300:-1] > t))
knife_defocus4_2 = knife_defocus4[:750,300:-1]
plt.imshow(binary_mask,cmap="gray")
plt.colorbar()
print(binary_mask[:,340:500])
# Avg
linePixels_x_axis = np.zeros((750))
linePixels = np.zeros((2,750))

for i in range(750):
    #Going through each row, save the y index for the first time the value is 1
    #print(np.where(binary_mask[i,:] == 1)[0][0])
    #linePixels_x_axis[i] = np.where(binary_mask[i,:] == 1)[0][0]

    linePixels[0,i] = (i)
    linePixels[1,i] = np.where(binary_mask[i,:] == 0)[0][0]


plt.imshow(binary_mask); plt.plot(linePixels[1], linePixels[0])

# 2) Use pixels along line to get y = mx+b equation of line (point slope form); linear fit function (whole row at once, min di
#    Linear interpolation to get line? https://realpython.com/linear-regression-in-python/
xVals = linePixels[1,:].reshape((-1, 1))
yVals = linePixels[0,:]
#print(xVals[:150])
#print(yVals[:150])
model = LinearRegression().fit(xVals, yVals)
print(f"intercept: {model.intercept_}")

print(f"slope: {model.coef_}")
# 3) Find distance, D, from line to each pixel to the left (perpendicular distance formula) (all computed simultaneously), cre
#    Use cropped image? nothing left of pixel 200 https://stackoverflow.com/questions/57063442/measuring-distance-from-points-
a = model.coef_
```

```
b = -1
c = model.intercept_

x = np.arange(0, knife_defocus4_2.shape[1])
y = np.arange(0, knife_defocus4_2.shape[0])

X, Y = np.meshgrid(x,y)

D = (a*X + b*Y + c)/(np.sqrt(a**2 + b**2))

plt.figure()
plt.imshow(knife_defocus4_2, cmap = "gray")
plt.title("Knife Edge Image, Defocused at 4 lp/mm")
plt.colorbar()

# 4) Plot sorted D vs I (intensity of those pixels) --> Edge Spread Function
D_sort1 = np.argsort(D.ravel())
D_sort2 = np.take_along_axis(D.ravel(), D_sort1, axis = 0)
intensity = np.take_along_axis(knife_defocus4_2.ravel(), D_sort1, axis = 0)/np.max(knife_defocus4_2)

plt.figure()
plt.plot(D_sort2, intensity)
plt.title("Edge Spread Function (ESF): Not Binned")
plt.xlabel("Pixels")
plt.ylabel("Normalized Intensity")
# 5) Bin D vs I plot so it's uniformly sampled (loop), delta d smaller than pixel size
numDistances = np.round(D_sort2[-1] + np.abs(D_sort2[0]))
numBins = 2*numDistances
# 2 bins per pixel
binnedIntensity, binEdges, binNumber = stats.binned_statistic(D_sort2, intensity, 'mean', numBins)
# Check to make sure there's at least one point in every bin
if(1 in np.isnan(binnedIntensity)):
  print('Bins Too Small')
else:
  print('Great Bins')

binEdges = binEdges[1:]
plt.figure()
plt.plot(binEdges, binnedIntensity)
plt.title("Edge Spread Function (ESF)")
plt.xlabel("Distance from Edge [Pixels]")
plt.ylabel("Normalized Intensity")

# 6) Differentiate ESF to get LSF (w filter) (can't have bins w no samples in them)
binSize = binEdges[1] - binEdges[0]
LSF = np.diff(binnedIntensity)/binSize
LSF_x = binEdges[1:]
plt.figure()
plt.plot(LSF_x, LSF)
plt.xlim([-50,50])
plt.title("Line Spread Function (LSF)")
plt.xlabel("Distance from Edge [Pixels]")
plt.ylabel("Normalized Intensity")

# 7) FFT of LSF --> MTF cycles/mm

MTF = np.abs(np.fft.fft(LSF))/np.max(np.abs(np.fft.fft(LSF)[:1000]))
pix = numBins*binSize #bins * pixels/bin
mm = pix * 0.5/34 # pixels * mm/pixel
MTF_x = np.arange(0, len(MTF))/mm # Cycles per mm

half = np.zeros(len(MTF_x))
half[:] = 0.5
plt.figure()
plt.plot(MTF_x, MTF); plt.plot(MTF_x, half)
plt.xlim([0,40])
plt.title("Modulation Transfer Function (MTF)")
plt.xlabel("Cycles per mm")
plt.ylabel("MTF")
```

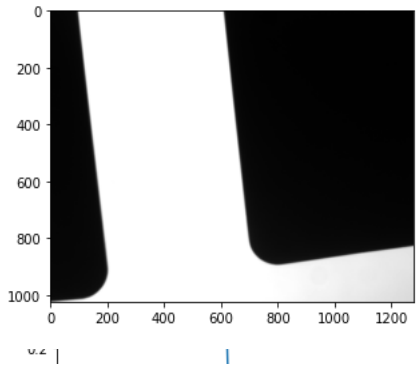
```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [1 1 1 ... 0 0 0]
 [1 1 1 ... 0 0 0]
 [1 1 1 ... 0 0 0]]
intercept: -2868.360378724122
slope: [9.25487939]
Great Bins
Text(0, 0.5, 'MTF')
```




```
knife_defocus8 = sio.loadmat('/content/drive/My Drive/50frames_knifeTarget_defocus8_angle2_part2.mat')["Average"]
```

```
plt.figure()
plt.imshow(knife_defocus8, cmap = "gray")
```

```
<matplotlib.image.AxesImage at 0x7ff0a410ca00>
```



```
## Part 2: Calculating MTF: Defocused to the 8 line pairs/ mm Knife edge image

# 1) Edge detection to find pixels along line (pixel closest to 50%, normalize)
# Average images

# Normalization
t = 50
# Crop first 200 columns
binary_mask = np.array(1*(knife_defocus8[:800,250:-1] > t))
knife_defocus8_2 = knife_defocus8[:800,250:-1]
plt.imshow(binary_mask,cmap="gray")
plt.colorbar()
print(binary_mask[:,340:500])
# Avg
linePixels_x_axis = np.zeros((800))
linePixels = np.zeros((2,800))

for i in range(800):
    #Going through each row, save the y index for the first time the value is 1
    #print(np.where(binary_mask[i,:] == 1)[0][0])
```

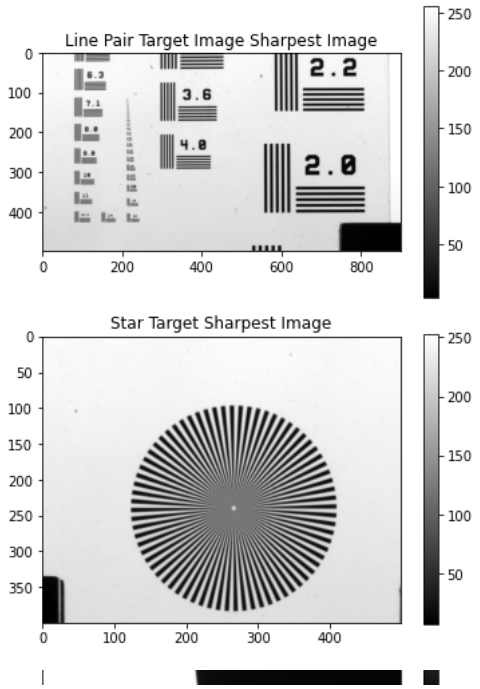
```
      #linePixels_x_axis[i] = np.where(binary_mask[i,:] == 1)[0][0]

      linePixels[0,i] = (i)
      linePixels[1,i] = np.where(binary_mask[i,:] == 0)[0][0]


  plt.imshow(binary_mask); plt.plot(linePixels[1], linePixels[0])


  # 2) Use pixels along line to get y = mx+b equation of line (point slope form); linear fit function (whole row at once, min di
  #    Linear interpolation to get line? https://realpython.com/linear-regression-in-python/
  xVals = linePixels[1,:].reshape((-1, 1))
  yVals = linePixels[0,:]
  #print(xVals[:150])
  #print(yVals[:150])
  model = LinearRegression().fit(xVals, yVals)
  print(f"intercept: {model.intercept_}")

  print(f"slope: {model.coef_}")
  # 3) Find distance, D, from line to each pixel to the left (perpendicular distance formula) (all computed simultaneously), cre
  #    Use cropped image? nothing left of pixel 200 https://stackoverflow.com/questions/57063442/measuring-distance-from-points-
  a = model.coef_
  b = -1
  c = model.intercept_

  x = np.arange(0, knife_defocus8_2.shape[1])
  y = np.arange(0, knife_defocus8_2.shape[0])

  X, Y = np.meshgrid(x,y)

  D = (a*X + b*Y + c)/(np.sqrt(a**2 + b**2))

  plt.figure()
  plt.imshow(knife_defocus8_2, cmap = "gray")
  plt.title("Knife Edge Image, Defocused at 8 lp/mm")
  plt.colorbar()

  # 4) Plot sorted D vs I (intensity of those pixels) --> Edge Spread Function
  D_sort1 = np.argsort(D.ravel())
  D_sort2 = np.take_along_axis(D.ravel(), D_sort1, axis = 0)
  intensity = np.take_along_axis(knife_defocus8_2.ravel(), D_sort1, axis = 0)/np.max(knife_defocus8_2)

  plt.figure()
  plt.plot(D_sort2, intensity)
  plt.title("Edge Spread Function (ESF): Not Binned")
  plt.xlabel("Pixels")
  plt.ylabel("Normalized Intensity")
  # 5) Bin D vs I plot so it's uniformly sampled (loop), delta d smaller than pixel size
  numDistances = np.round(D_sort2[-1] + np.abs(D_sort2[0]))
  numBins = 2*numDistances
  # 2 bins per pixel
  binnedIntensity, binEdges, binNumber = stats.binned_statistic(D_sort2, intensity, 'mean', numBins)
  # Check to make sure there's at least one point in every bin
  if(1 in np.isnan(binnedIntensity)):
    print('Bins Too Small')
  else:
    print('Great Bins')

  binEdges = binEdges[1:]
  plt.figure()
  plt.plot(binEdges, binnedIntensity)
  plt.title("Edge Spread Function (ESF)")
  plt.xlabel("Distance from Edge [Pixels]")
  plt.ylabel("Normalized Intensity")

  # 6) Differentiate ESF to get LSF (w filter) (can't have bins w no samples in them)
  binSize = binEdges[1] - binEdges[0]
  LSF = np.diff(binnedIntensity)/binSize
  LSF_x = binEdges[1:]
  plt.figure()
  plt.plot(LSF_x, LSF)
  plt.xlim([-50,50])
  plt.title("Line Spread Function (LSF)")
  plt.xlabel("Distance from Edge [Pixels]")
  plt.ylabel("Normalized Intensity")

  # 7) FFT of LSF --> MTF cycles/mm

  MTF = np.abs(np.fft.fft(LSF))/np.max(np.abs(np.fft.fft(LSF)[:1000]))
  pix = numBins*binSize #bins * pixels/bin
  mm = pix * 0.5/34 # pixels * mm/pixel
  MTF_x = np.arange(0, len(MTF))/mm # Cycles per mm
```

```
half = np.zeros(len(MTF_x))
half[:] = 0.5
plt.figure()
plt.plot(MTF_x, MTF); plt.plot(MTF_x, half)
plt.xlim([0,40])
plt.title("Modulation Transfer Function (MTF)")
plt.xlabel("Cycles per mm")
plt.ylabel("MTF")
```

```
# Loading required images for analysis
line_focus = sio.loadmat('/content/drive/My Drive/50frames_lineTarget_part2.mat')["Average"]
line_defocus8 = sio.loadmat('/content/drive/My Drive/50frames_lineTargetDefocus8_part2.mat')["Average"]
line_defocus4 = sio.loadmat('/content/drive/My Drive/50frames_lineTargetDefocus4_part2.mat')["Average"]
star_focus = sio.loadmat('/content/drive/My Drive/50frames_starTarget_part2.mat')["Average"]
star_defocus8 = sio.loadmat('/content/drive/My Drive/50frames_starTarget_defocus8_part2.mat')["Average"]
star_defocus4 = sio.loadmat('/content/drive/My Drive/50frames_starTarget_defocus4_part2.mat')["Average"]

#knife_defocus8 = sio.loadmat('/content/drive/My Drive/50frames_knifeTarget_defocus8_angle2_part2.mat')["Average"]


##Part 3

line_part3 = sio.loadmat('/content/drive/My Drive/LineF_part3.mat')["Average"]
star_part3 = sio.loadmat('/content/drive/My Drive/StarF_part3.mat')["Average"]


plt.figure()
plt.imshow(line_part3[:500, 200:1100], cmap = "gray")
plt.title("Line Pair Target Image Sharpest Image")
```

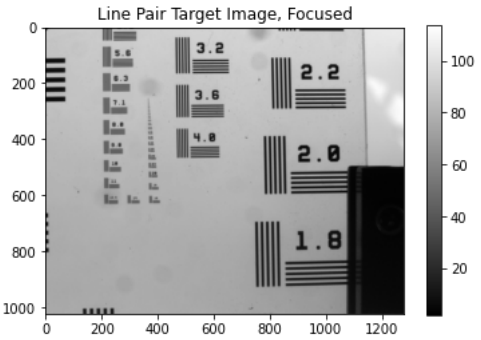
```
plt.colorbar()

plt.figure()
plt.imshow(star_part3[0:400, 400:900], cmap = "gray")
plt.title("Star Target Sharpest Image")
plt.colorbar()
```

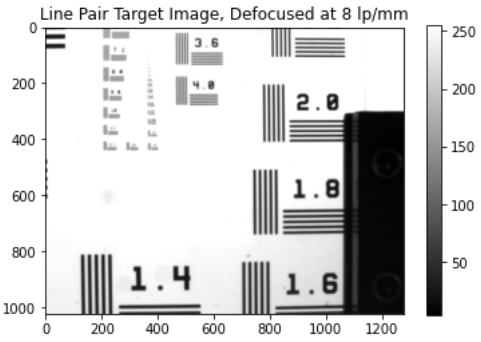
```
<matplotlib.colorbar.Colorbar at 0x7f7386d005b0>
```

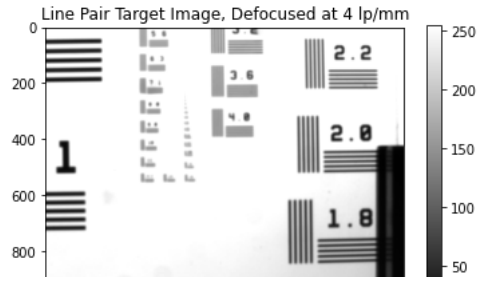



```
# Displaying required images

plt.figure()
plt.imshow(line_focus, cmap = "gray")
plt.title("Line Pair Target Image, Focused")
plt.colorbar()

plt.figure()
plt.imshow(line_defocus8, cmap = "gray")
plt.title("Line Pair Target Image, Defocused at 8 lp/mm")
plt.colorbar()

plt.figure()
plt.imshow(line_defocus4, cmap = "gray")
plt.title("Line Pair Target Image, Defocused at 4 lp/mm")
plt.colorbar()

plt.figure()
plt.imshow(star_focus, cmap = "gray")
plt.title("Star Target Image, Focused")
plt.colorbar()

plt.figure()
plt.imshow(star_defocus8, cmap = "gray")
plt.title("Star Target Image, Defocused at 8 lp/mm")
plt.colorbar()

plt.figure()
plt.imshow(star_defocus4, cmap = "gray")
plt.title("Star Target Image, Defocused at 4 lp/mm")
plt.colorbar()

#plt.title("Knife Edge Image, Defocused at 8 lp/mm")
```

```
<matplotlib.colorbar.Colorbar at 0x7ff09e9d5f40>
```








```
binEdges[56] - binEdges[55]
```

```
0.5000629114463777
```

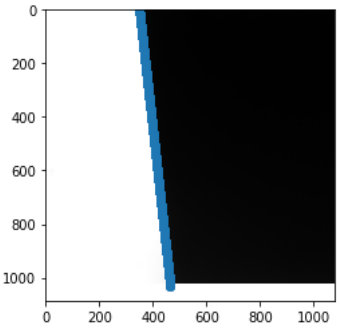
```
## Part 2: MTF of defocused 4 knife image (most defocused)
from google.colab import drive
drive.mount('/content/drive')
knife_focus = sio.loadmat('/content/drive/My Drive//50frames_knifeTarget_defocus4_angle2_part2.mat')["Average"] # Maybe crop d
```

```
## Part 2: MTF of defocused 8 knife image (middle defocused)
```



```
plt.imshow(knife_focus2, cmap = "gray") ; plt.plot(linePixels[1,:], a*(linePixels[1,:]) + c, "o")
```

```
[<matplotlib.lines.Line2D at 0x7f07516de4c0>]
```



```
plt.plot(linePixels[1,:], a*(linePixels[1,:]) + c, "o")
```

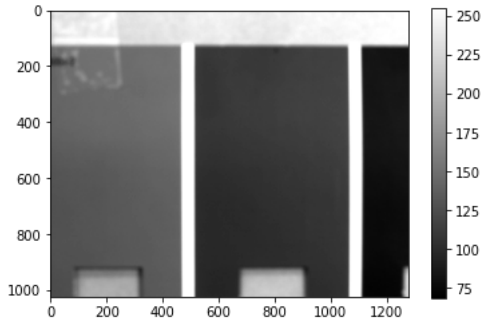
```
[<matplotlib.lines.Line2D at 0x7f07516170a0>]
```


```
print(x)
print(x.shape)
print(y)
print(y.shape)
print(X)
print(X.shape)
print(Y)
print(Y.shape)
```

```
[   0    1    2 ... 1021 1022 1023]
(1024,)
[   0    1    2 ... 1076 1077 1078]
(1079,)
[[   0    1    2 ... 1021 1022 1023]
 [   0    1    2 ... 1021 1022 1023]
 [   0    1    2 ... 1021 1022 1023]
 ...
 [   0    1    2 ... 1021 1022 1023]
 [   0    1    2 ... 1021 1022 1023]
 [   0    1    2 ... 1021 1022 1023]]
(1079, 1024)
[[   0    0    0 ...    0    0    0]
 [   1    1    1 ...    1    1    1]
 [   2    2    2 ...    2    2    2]
 ...
 [1076 1076 1076 ... 1076 1076 1076]
 [1077 1077 1077 ... 1077 1077 1077]
 [1078 1078 1078 ... 1078 1078 1078]]
(1079, 1024)
```

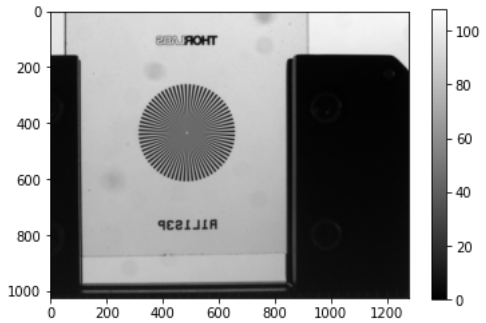
```
img3 = cam.grab_image().copy()
im3 = plt.imshow(img3,cmap='gray')
plt.colorbar(im3)
```

```
<matplotlib.colorbar.Colorbar at 0x234d59ed4a8>
```




```
img4 = cam.grab_image().copy()
im4 = plt.imshow(img4,cmap='gray')
plt.colorbar(im4)
```

```
<matplotlib.colorbar.Colorbar at 0x234d5dbb358>
```



Live Video Feed for tuning focal length and aperture.

```
cam._get_AOI() # look at the Area of Interest.
```

```
(0, 0, 1280, 1024)
```

```
# set frame rate
cam.set_framerate(framerate = "10Hz")
# get exposure range, you can see that the frame rate will change the maximum exposure time.
print(cam._get_exposure_range())
# print current frame rate
print(cam.framerate)
```

```
(<Quantity(0.00898245614, 'millisecond')>, <Quantity(197.594526, 'millisecond')>)
5.05606735178449 hertz
```

```
# to quit press live feed press "q"
cam.start_live_video()
while cam.is_open:

    frame = cam.grab_image(timeout='100s', copy=True)
    frame1 = np.stack((frame,) * 3,-1) #make frame as 1 channel image
    frame1 = frame1.astype(np.uint8)
    gray = cv2.cvtColor(frame1, cv2.COLOR_BGR2GRAY)
    #now u can apply opencv features
    cv2.imshow('Camera', gray)
    if cv2.waitKey(30) & 0xFF == ord('q'):
       break
cam.stop_live_video()
cv2.destroyAllWindows()
```

```
2.0472896614601814 hertz
```

```
cam.close()
```

```
del cam
```