School of Computer Science and Applied Mathematics
UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG

APPM4058A Assignment 1 – Report

Wesley Earl Stander - 1056114 - Coms Hons

May 4, 2020

1 Low light image contrast enhancement

1.1 Contrast stretching

Figure 1: Contrast Stretching of Road 1

Figure 2: Contrast Stretching of Road 2

The stretched images for Road 1 and 2 look the same as they already had a full dynamic range. The sports image looks

Figure 3: Contrast Stretching of Sports

better because the original image had a small dynamic range. To contrast stretch the values are mapped from their current dynamic range to [0,255].

1.2 Histogram equalization

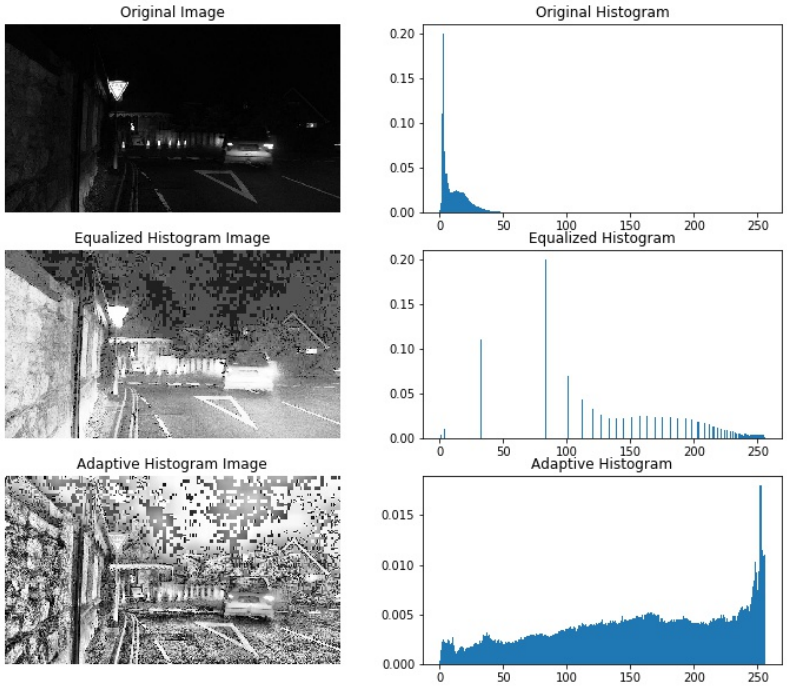

Figure 4: Histogram Equalization of Road 1

To equalize the histogram the bins are remapped so that each are equally likely.

The original image in figure 4 is very dark and it is difficult to see the objects in the background. The adaptive histogram image in figure 4 is subjectively the best looking version of this image. All the details are visible and they don't look washed out like in the equalized histogram image. The equalized histogram image in figure 4 is better than the original but worse than the adaptive histogram. The equalized histogram image in figure 4 has a washed out appearance and the wall and car can't be seen clearly.

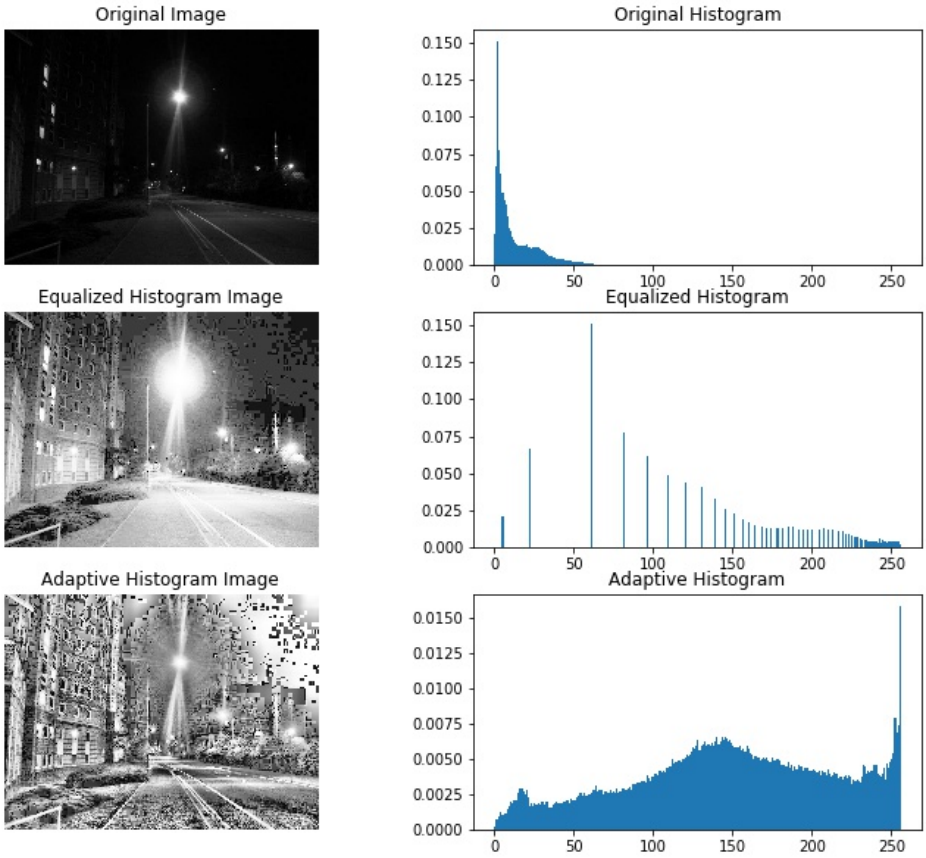

Figure 5: Histogram Equalization of Road 2

The original image in figure 5 is very dark and it is difficult to see the objects in the background. The equalized histogram image in figure 5 is subjectively the best looking version of this image. All the details are visible although the slightly incorrect. The adaptive histogram equalization image in figure 5 is better than the original but worse than the equalized histogram. The adaptive histogram image in figure 5 has block artifacts and the colouring is not linear as it should be such as in the top right hand corner.

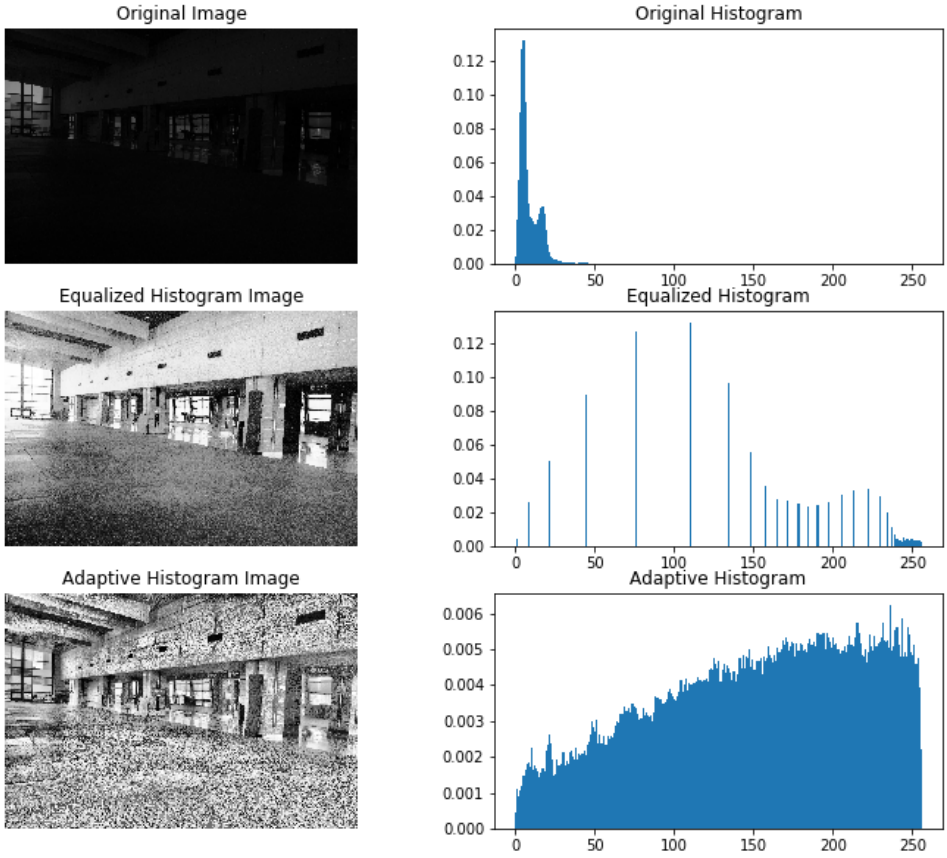

Figure 6: Histogram Equalization of Sports

The original image in figure 6 has a very low dynamic range and the details are not visible. The equalized histogram image in figure 6 is the best looking image in my opinion as all details are visible and the colouring looks correct. The adaptive histogram image in figure 6 is better than the original as the details are visible but the colouring is very wrong so it's worse than the equalized histogram image in figure 6.

To perform the adaptive histogram equalization a block of the image is extracted at each pixel value and histogram equalization is performed on the block to remap the pixel in question.

2 Spatial and Frequency domain filtering

2.1 Frequency domain filtering

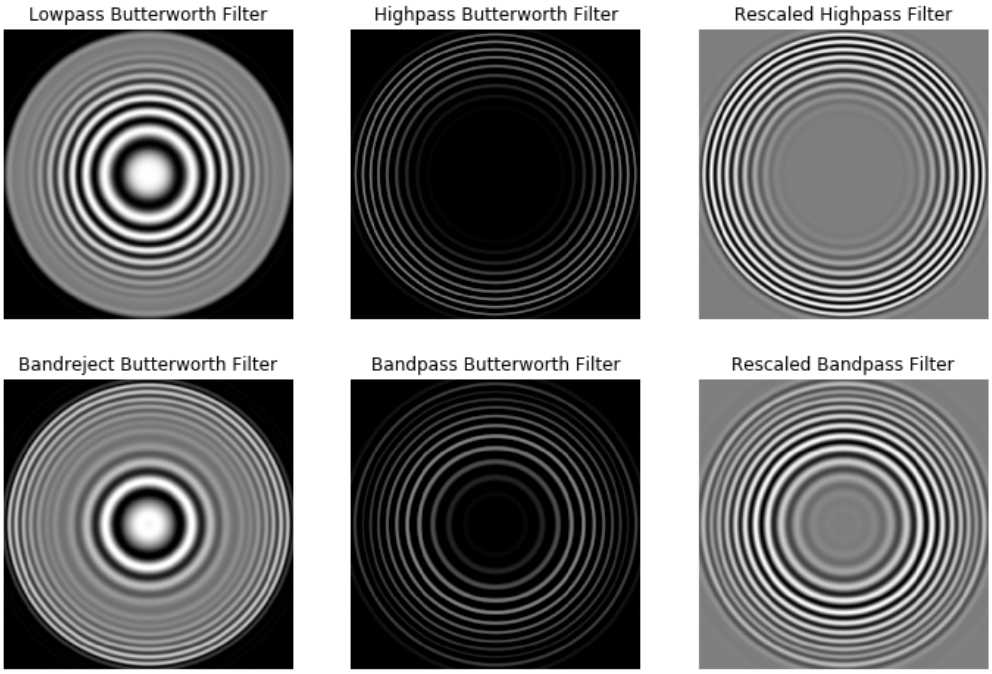

Figure 7: Frequency domain filtering of zoneplate

The images have some slight changes to the output in the document so it is recommended to view the original images in the attached folder to see the results. The exact images were reproduced up to the details in the background such as extra rings. The parameters of the filters were changed appropriately for each image to replicate the example images. All images and filters are padded. The butter-worth filter is generated using the formula and then element-wise multiplied with the image to return the low-pass butter-worth filter image. The filter is then subtracted from 1 and element-wise multiplied with the image to attain the high-pass butter-worth filter. The re-scaled high-pass filter is obtained by re-scaling the high-pass butter-worth filter image to a dynamic range of [0,255]. The band-reject image is generated by element-wise multiplying the band reject generated filter with the image. The band pass is generated similarly by element-wise subtracting the band-reject filter from 1 and element-wise multiplying it with the image. The re-scaled band-pass filter is generated by re-scaling the dynamic range of the band-pass butter-worth filtered image to a dynamic range of [0,255].

2.2 spatial domain filtering

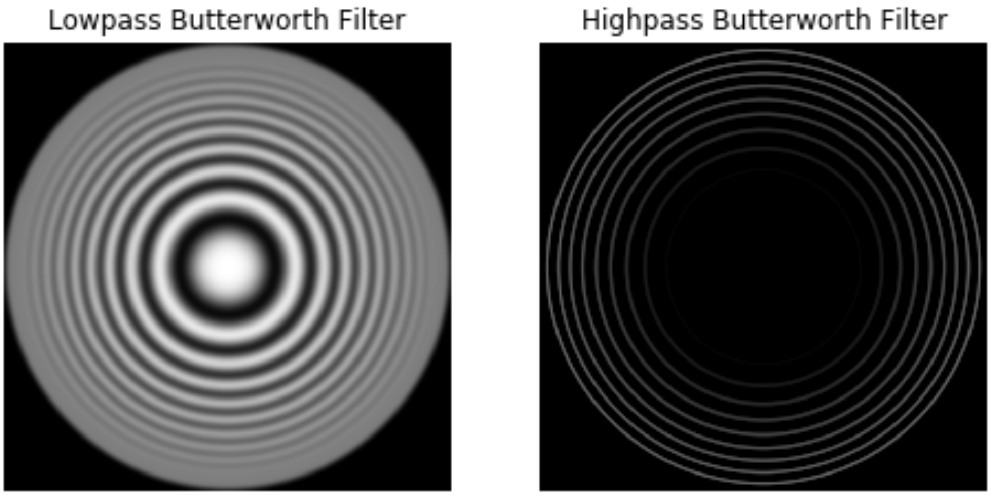

Figure 8: Spatial domain filtering of zoneplate

The spatial filtering provides similar results but not exact as there is no equivalent to a butter-worth filter in the spatial domain. The filter was generated using a Gaussian kernel that was made into a disk to prevent corner artifacts. The low-pass butter-worth filter image is generated by performing a 2d convolution with the filter and the image. The image is padded by half filter size. The high-pass butter-worth filter image is generated by subtracting the low-pass image from the original image.

3 Source Code list

3.1 Libraries

1. numpy
2. matplotlib.pyplot
3. math
4. skimage.color.rgb2gray
5. skimage.morphology.disk
6. scipy.fftpack

3.2 Question 1.1

```
import numpy as np
import matplotlib.pyplot as plt

def contrast(image):
    return np.max(image) - np.min(image)

def ScaleIntensity (image, K=1):
    m = contrast(image)
    out = np.array(image).astype(float)
    offset = K*float(np.min(image)/m)
    for j in range(image.shape[1]):
        for i in range(image.shape[0]):
            out[i][j] = K*(out[i][j]/m) - offset
    return out

img1 = plt.imread("imgs/road_low_1.jpg")
img2 = plt.imread("imgs/road_low_2.jpg")
img3 = plt.imread("imgs/sports_low.png")
plt.gray()

img11 = ScaleIntensity(img1,255)

img21 = ScaleIntensity(img2,255)

img31 = ScaleIntensity(img3,255)

fig, axs = plt.subplots(1,2,figsize=(14,4))
[axi.set_axis_off() for axi in axs.ravel()]
axs[0] = fig.add_subplot(1,2,1)
axs[0].set_title("Original")
axs[0].imshow(img1, vmin = 0, vmax= 255)
plt.axis('off')
axs[1] = fig.add_subplot(1,2,2)
axs[1].set_title("Stretched")
axs[1].imshow(img11, vmin = 0, vmax= 255)
plt.axis('off')
fig.savefig("output/Road1_1-1.jpg")

fig, axs = plt.subplots(1,2,figsize=(12,5))
[axi.set_axis_off() for axi in axs.ravel()]
axs[0] = fig.add_subplot(1,2,1)
axs[0].set_title("Original")
axs[0].imshow(img2, vmin = 0, vmax= 255)
plt.axis('off')
axs[1] = fig.add_subplot(1,2,2)
axs[1].set_title("Stretched")
axs[1].imshow(img21, vmin = 0, vmax= 255)
plt.axis('off')
fig.savefig("output/Road2_1-1.jpg")

fig, axs = plt.subplots(1,2,figsize=(12,4))
[axi.set_axis_off() for axi in axs.ravel()]
axs[0] = fig.add_subplot(1,2,1)
axs[0].set_title("Original")
```

```
axs[0].imshow(img3, vmin = 0, vmax= 1)
plt.axis('off')
axs[1] = fig.add_subplot(1,2,2)
axs[1].set_title("Stretched")
axs[1].imshow(img31, vmin = 0, vmax= 255)
plt.axis('off')
fig.savefig("output/Sports_1-1.png")
```

3.3 Question 1.2 and 1.3

```
import numpy as np
import matplotlib.pyplot as plt

def contrast(image):
    return np.max(image) - np.min(image)

def ScaleIntensity (image, K=1):
    m = contrast(image)
    out = np.array(image).astype(float)
    offset = K*float(np.min(image)/m)
    for j in range(image.shape[1]):
        for i in range(image.shape[0]):
            out[i][j] = K*(out[i][j]/m) - offset
    return out

img1 = plt.imread("imgs/road_low_1.jpg")
img2 = plt.imread("imgs/road_low_2.jpg")
img3 = plt.imread("imgs/sports_low.png")
plt.gray()

def histEqual(f):
    out = np.copy(f)
    rk, nk = np.unique(out, return_counts=True) #unique intensities and respective counts
    pk = nk/float(np.size(f)) #probability of each unique intensity
    sk = np.cumsum(pk) #cumulative sums of pixel probabilities
    mul = sk*255.0 #cumulative frequency multiplied by max value
    roundVal = np.round(mul) #round each multiple
    for i in range(len(f)):
        for j in range(len(f[0])):
            out[i][j] = roundVal[np.where(rk==f[i][j])] #map pixels for equalization
    return out

hist11 = histEqual(img1)
print("Equalize Hist 1")

hist21 = histEqual(img2)
print("Equalize Hist 2")

img311 = ScaleIntensity(img3, contrast(img3)*255) #image must be scaled as it is 0-1

hist31 = histEqual(img311)
print("Equalize Hist 3")
```

```
def AdaptHistEqual(f, blockSize = 16):
    out = np.copy(f)
    padAmount = int(round(blockSize / 2)) #amount to pad at each edge for block to be used
    padded = np.pad(out, pad_width=((padAmount, padAmount), (padAmount, padAmount)),
         mode='symmetric') #pad array
    for i in range(len(f)):
        for j in range(len(f[0])):
            block = padded[i:i+blockSize,j:j+blockSize] #extract square for each pixel
            rk, nk = np.unique(block, return_counts=True) #unique intensities and respective counts
            pk = nk/float(np.size(block)) #probability of each unique intensity
            sk = np.cumsum(pk) #cumulative sums of pixel probabilities
            mul = sk*255.0 #cumulative frequency multiplied by max value
            roundVal = np.round(mul) #round each multiple
            out[i][j] = roundVal[np.where(rk==block[padAmount-1][padAmount-1])] #extract central
                pixel value
    return out

hist12 = AdaptHistEqual(img1,64)
print("adaptHist 1")
hist22 = AdaptHistEqual(img2,64)
print("adaptHist 2")
hist32 = AdaptHistEqual(img311,64)
print("adaptHist 3")

fig, axs = plt.subplots(3,2,figsize=(12,10))
[axi.set_axis_off() for axi in axs.ravel()]
axs[0][0] = fig.add_subplot(3,2,1)
axs[0][0].set_title("Original Image")
axs[0][0].imshow(img1, vmin = 0, vmax= 255)
plt.axis('off')
axs[1][0] = fig.add_subplot(3,2,3)
axs[1][0].set_title("Equalized Histogram Image")
axs[1][0].imshow(hist11, vmin = 0, vmax= 255)
plt.axis('off')
axs[2][0] = fig.add_subplot(3,2,5)
axs[2][0].set_title("Adaptive Histogram Image")
axs[2][0].imshow(hist12, vmin = 0, vmax= 255)
plt.axis('off')
axs[0][1] = fig.add_subplot(3,2,2)
axs[0][1].set_title("Original Histogram")
axs[0][1].hist(img1.ravel(), 256, [0,256], density=1)
axs[1][1] = fig.add_subplot(3,2,4)
axs[1][1].set_title("Equalized Histogram")
axs[1][1].hist(hist11.ravel(), 256, [0,256], density=1)
axs[2][1] = fig.add_subplot(3,2,6)
axs[2][1].set_title("Adaptive Histogram")
axs[2][1].hist(hist12.ravel(), 256, [0,256], density=1)
fig.savefig("output/Road1_1-2+3.jpg")

fig, axs = plt.subplots(3,2,figsize=(12,10))
[axi.set_axis_off() for axi in axs.ravel()]
axs[0][0] = fig.add_subplot(3,2,1)
axs[0][0].set_title("Original Image")
axs[0][0].imshow(img2, vmin = 0, vmax= 255)
plt.axis('off')
axs[1][0] = fig.add_subplot(3,2,3)
```

```
axs[1][0].set_title("Equalized Histogram Image")
axs[1][0].imshow(hist21, vmin = 0, vmax= 255)
plt.axis('off')
axs[2][0] = fig.add_subplot(3,2,5)
axs[2][0].set_title("Adaptive Histogram Image")
axs[2][0].imshow(hist22, vmin = 0, vmax= 255)
plt.axis('off')
axs[0][1] = fig.add_subplot(3,2,2)
axs[0][1].set_title("Original Histogram")
axs[0][1].hist(img2.ravel(), 256, [0,256], density=1)
axs[1][1] = fig.add_subplot(3,2,4)
axs[1][1].set_title("Equalized Histogram")
axs[1][1].hist(hist21.ravel(), 256, [0,256], density=1)
axs[2][1] = fig.add_subplot(3,2,6)
axs[2][1].set_title("Adaptive Histogram")
axs[2][1].hist(hist22.ravel(), 256, [0,256], density=1)
fig.savefig("output/Road2_1-2+3.jpg")

fig, axs = plt.subplots(3,2,figsize=(12,10))
[axi.set_axis_off() for axi in axs.ravel()]
axs[0][0] = fig.add_subplot(3,2,1)
axs[0][0].set_title("Original Image")
axs[0][0].imshow(img3, vmin = 0, vmax= 1)
plt.axis('off')
axs[1][0] = fig.add_subplot(3,2,3)
axs[1][0].set_title("Equalized Histogram Image")
axs[1][0].imshow(hist31, vmin = 0, vmax= 255)
plt.axis('off')
axs[2][0] = fig.add_subplot(3,2,5)
axs[2][0].set_title("Adaptive Histogram Image")
axs[2][0].imshow(hist32, vmin = 0, vmax= 255)
plt.axis('off')
axs[0][1] = fig.add_subplot(3,2,2)
axs[0][1].set_title("Original Histogram")
axs[0][1].hist(img311.ravel(), 256, [0,256], density=1)
axs[1][1] = fig.add_subplot(3,2,4)
axs[1][1].set_title("Equalized Histogram")
axs[1][1].hist(hist31.ravel(), 256, [0,256], density=1)
axs[2][1] = fig.add_subplot(3,2,6)
axs[2][1].set_title("Adaptive Histogram")
axs[2][1].hist(hist32.ravel(), 256, [0,256], density=1)
fig.savefig("output/Sports_1-2+3.png")
```

3.4 Question 2.1

```
import numpy as np
import matplotlib.pyplot as plt
from skimage.color import rgb2gray
from scipy import fftpack as ff
import math

def contrast(image):
    return np.max(image) - np.min(image)
```

```
def ScaleIntensity (image, K=1):
    m = contrast(image)
    out = np.array(image).astype(float)
    offset = K*float(np.min(image)/m)
    for j in range(image.shape[1]):
        for i in range(image.shape[0]):
            out[i][j] = K*(out[i][j]/m) - offset
    return out

imgZ = rgb2gray(plt.imread("imgs/zoneplate.tif"))
plt.gray()

def paddedSize(img):
    return (2*img.shape[0])-1, (2*img.shape[1])-1

def freqz2(P, Q, T, param, w=1, order=3):
    u = np.arange(0,P,1.0)
    v = np.arange(0,Q,1.0)
    idx = np.where(u>P/2)
    u[idx] = u[idx] - P
    idy = np.where(v>Q/2)
    v[idy] = v[idy]-Q
    V,U = np.meshgrid(v,u)
    D = (V**2 + U**2)**(1/2)
    out = np.zeros((P, Q))
    if (T == "ILPF"):
        for i in range(P):
            for j in range(Q):
                if (D[i][j] <= param):
                    out[i][j] = 1
    elif (T == "GLPF"):
        for i in range(P):
            for j in range(Q):
                out[i][j] = math.exp((-1*(D[i][j]**2))/(2*(param**2)))
    elif (T == "BLPF"):
        for i in range(P):
            for j in range(Q):
                out[i][j] = 1/(1+(D[i][j]/param)**(2*order)) #butter-worth low pass filter
    elif (T == "BRBF"):
        for i in range(P):
            for j in range(Q):
                out[i][j] = 1 / (1+ ((w * D[i][j])/((D[i][j])**2 - param**2))**(2*order)) #n=3
    return out

def filterImageButterWorth (img, amount, order =3):
    p = paddedSize(img)
    originalSize = [img.shape[0], img.shape[1]]
    img = np.pad(img, ((0, p[0]-originalSize[0]), (0, p[1]-originalSize[1])))
    imgDFT = ff.fft2(img)
    H = freqz2(p[0], p[1], "BLPF", amount, 1, order)
    imgFiltered = np.multiply(imgDFT, H)
    imgOut = ff.ifft2(imgFiltered)[0:originalSize[0], 0:originalSize[1]]
    return np.real(imgOut)

def UnsharpMaskButterWorth (img, amount, order=3):
```

```
    p = paddedSize(img)
    originalSize = [img.shape[0], img.shape[1]]
    img = np.pad(img, ((0, p[0]-originalSize[0]), (0, p[1]-originalSize[1])))
    imgDFT = ff.fft2(img)
    H = freqz2(p[0], p[1], "BLPF", amount, 1, order)
    H = np.subtract(1,H)
    imgFiltered = np.multiply(imgDFT, H)
    imgOut = ff.ifft2(imgFiltered)[0:originalSize[0], 0:originalSize[1]]
    return np.real(imgOut)

def BandRejectButterWorth (img, amount, w, order=3):
    p = paddedSize(img)
    originalSize = [img.shape[0], img.shape[1]]
    img = np.pad(img, ((0, p[0]-originalSize[0]), (0, p[1]-originalSize[1])))
    imgDFT = ff.fft2(img)
    H = freqz2(p[0], p[1], "BRBF", amount, w, order)
    imgFiltered = np.multiply(imgDFT, H)
    imgOut = ff.ifft2(imgFiltered)[0:originalSize[0], 0:originalSize[1]]
    return np.real(imgOut)

def BandPassButterWorth (img, amount, w, order=3):
    p = paddedSize(img)
    originalSize = [img.shape[0], img.shape[1]]
    img = np.pad(img, ((0, p[0]-originalSize[0]), (0, p[1]-originalSize[1])))
    imgDFT = ff.fft2(img)
    H = freqz2(p[0], p[1], "BRBF", amount, w, order)
    H = np.subtract(1,H)
    imgFiltered = np.multiply(imgDFT, H)
    imgOut = ff.ifft2(imgFiltered)[0:originalSize[0], 0:originalSize[1]]
    return np.real(imgOut)

imgZ1 = filterImageButterWorth(imgZ,57, 6)

imgZ2 = UnsharpMaskButterWorth(imgZ, 57, 6)

imgZ3 = ScaleIntensity(imgZ2,255)

imgZ4 = BandRejectButterWorth(imgZ, 53, 37, 3)

imgZ5 = BandPassButterWorth(imgZ, 50, 37, 2)

imgZ6 = ScaleIntensity(imgZ5,255)

fig, axs = plt.subplots(2,3,figsize=(12,8))
[axi.set_axis_off() for axi in axs.ravel()]
axs[0][0] = fig.add_subplot(2,3,1)
axs[0][0].set_title("Lowpass Butterworth Filter")
axs[0][0].imshow(imgZ1, vmin = 0, vmax= 255)
plt.axis('off')
axs[1][0] = fig.add_subplot(2,3,2)
axs[1][0].set_title("Highpass Butterworth Filter")
axs[1][0].imshow(imgZ2, vmin=0, vmax=255)
plt.axis('off')
axs[0][1] = fig.add_subplot(2,3,3)
axs[0][1].set_title("Rescaled Highpass Filter")
axs[0][1].imshow(imgZ3)
```

```
plt.axis('off')
axs[1][1] = fig.add_subplot(2,3,4)
axs[1][1].set_title("Bandreject Butterworth Filter")
axs[1][1].imshow(imgZ4, vmin=1, vmax=255)
plt.axis('off')
axs[0][2] = fig.add_subplot(2,3,5)
axs[0][2].set_title("Bandpass Butterworth Filter")
axs[0][2].imshow(imgZ5, vmin=0, vmax=255)
plt.axis('off')
axs[1][2] = fig.add_subplot(2,3,6)
axs[1][2].set_title("Rescaled Bandpass Filter")
axs[1][2].imshow(imgZ6)
plt.axis('off')
fig.savefig("imgs/2-1.png", vmin=0, vmax=255)
```

3.5 Question 2.2

```
import numpy as np
import matplotlib.pyplot as plt
from skimage.color import rgb2gray
from skimage import morphology as m
import math

def contrast(image):
    return np.max(image) - np.min(image)

def ScaleIntensity (image, K=1):
    m = contrast(image)
    out = np.array(image).astype(float)
    offset = K*float(np.min(image)/m)
    for j in range(image.shape[1]):
        for i in range(image.shape[0]):
            out[i][j] = K*(out[i][j]/m) - offset
    return out

imgZ = rgb2gray(plt.imread("imgs/zoneplate.tif"))
plt.gray()

def gaussKernel(m,sig,K=1):
    out = np.zeros((m,m))
    total = 0
    for i in range(m):
        for j in range(m):
            r = ((i-(m-1)/2)**2) + ((j-(m-1)/2)**2)
            out[i][j] = K * math.exp(-1 * (r/(2*(sig**2))))
            total += out[i][j]
    return out/total;

H = m.disk(9)
H = np.logical_and(H, gaussKernel(19, 2)) #disk guassian filter to reduce corner artifacts in output
    image

def twodConv(f,w):
```

```
    m = len(w)
    n = len(w[0])
    if m == n:
        y = len(f)
        x = len(f[0])
        g = np.pad(f, int((m-1)/2), mode='constant')
        new = np.zeros((y,x))
        for i in range(y):
            for j in range(x):
                    new[i][j] = np.sum(g[i:i+m, j:j+m] * w)
        return new

imgZ7 = twodConv(imgZ, H)
imgZ7 = ScaleIntensity(imgZ7,255)

imgZ8 = imgZ - imgZ7

fig, axs = plt.subplots(1,2,figsize=(8,4))
[axi.set_axis_off() for axi in axs.ravel()]
axs[0] = fig.add_subplot(1,2,1)
axs[0].set_title("Lowpass Butterworth Filter")
axs[0].imshow(imgZ7, vmin = 0, vmax= 255)
plt.axis('off')
axs[1] = fig.add_subplot(1,2,2)
axs[1].set_title("Highpass Butterworth Filter")
axs[1].imshow(imgZ8, vmin=37, vmax=255)
plt.axis('off')
fig.savefig("output/2-2.png", vmin=0, vmax=255)
```