Homework 2

November 6, 2021

Name - Parthasarathi Kumar
PID - A59003519
ECE 253 - Fundamentals of Digital Image Processing

Academic Integrity Policy: Integrity of scholarship is essential for an academic community. The University expects that both faculty and students will honor this principle and in so doing protect the validity of University intellectual work. For students, this means that all academic work will be done by the individual to whom it is assigned, without unauthorized aid of any kind.By including this in my report, I agree to abide by the Academic Integrity Policy mentioned above.

Assignment2_Problem1

November 7, 2021

[4]:
```
import numpy as np
import cv2
from matplotlib import pyplot as plt
```

Function to perform adaptive histogram equalization given an image and window size.

[5]:
```
def adaptive_histogram_equalization(I, win_size):
    '''
    I - RGB image
    win_size - window size

    This function performs adaptive histogram equalization
    on the input image and returns the output image
    '''
    # Pad image
    p = int((win_size-1)/2)
    I_pad = np.pad(I, ((p,p),(p,p)), 'symmetric')

    # AHE
    I_op = np.zeros_like(I)
    for i in range (p,I_pad.shape[0]-p):
        for j in range (p,I_pad.shape[1]-p):

            window = I_pad[i-p:i+p+1,j-p:j+p+1]
            window = window < I_pad[i,j]
            rank = np.count_nonzero(window, axis = (0,1))
            I_op[i-p,j-p] = rank*255/(win_size*win_size)

    return I_op
```

Driver code to run on given image with different window sizes.

[6]:
```
I = cv2.imread('/home/parth/work/UCSD/Fall 2021/ECE 253 Image Processing/
 ↪Assignment 2/beach.png')
I = cv2.cvtColor(I, cv2.COLOR_BGR2GRAY)

# plot original image
ax = plt.subplot(1,1,1)
```

```
ax.axis('off')
ax.title.set_text('Original Image')
ax.imshow(I, cmap = 'gray')
plt.show()

window_size = [33,65,129]

fig = plt.figure(figsize=(15, 15))
for w in range(3):
    I_AHE = adaptive_histogram_equalization(I,window_size[w]);
    ax = fig.add_subplot(3,1,w+1)
    ax.axis('off')
    ax.title.set_text('Window size' + str(window_size[w]))
    ax.imshow(I_AHE, cmap = 'gray')
plt.show()

I_HE = cv2.equalizeHist(I)
# plot HE image
ax = plt.subplot(1,1,1)
ax.axis('off')
ax.title.set_text('Image after Histogram Equalization')
ax.imshow(I_HE, cmap = 'gray')
plt.show()
```

Window size33

Window size65

Window size129

The image has better contrast after AHE and HE, making it easier to identify the 3 men in the darker part of the image.However, there are some artifacts introduced due to the image transformations applied. AHE leads to a better image in terms of identifying the object in dark areas (4 men) in comparision to HE.

Adaptive Histogram Equalization works better in case of beach.png, especially with a larger window size. Adaptive histogram equalization tends to be suited for images where the objective is to improve the localized contast value, as is the case for beach.png.
Selecting AHE or HE would depend on the image and its histogram distribution or current state of contrast. HE would make more sense if the histogram contains a single narrow cluster where as AHE would be more suited for images with multiple clusters across the range of internsity values.

Assignment2_Problem2

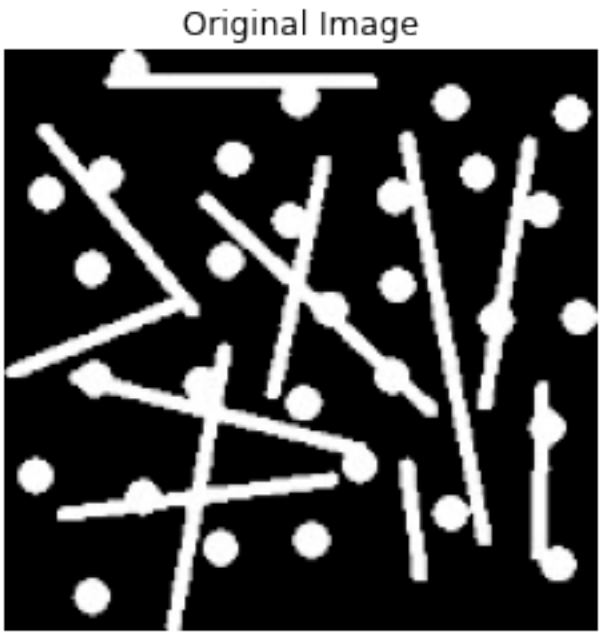
November 7, 2021

Import libraries

[95]:
```
import cv2
import numpy as np
from matplotlib import pyplot as plt
import scipy
from skimage import morphology
import pandas as pd
```

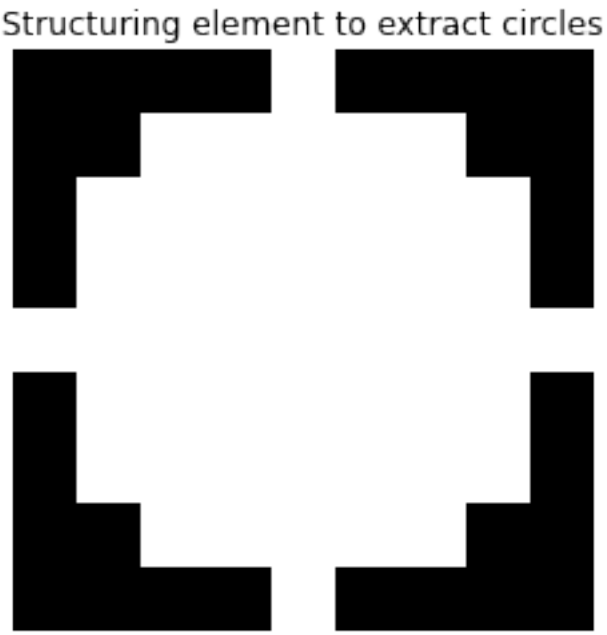
Problem2 Q(i)

[96]:
```
I = cv2.imread('circles_lines.jpg',0)
I[I<127] = 0
ax = plt.subplot(1,1,1)
ax.axis('off')
ax.imshow(I, cmap = 'gray')
ax.set_title('Original Image')
plt.show()
```



Perform opening to remove lines from image

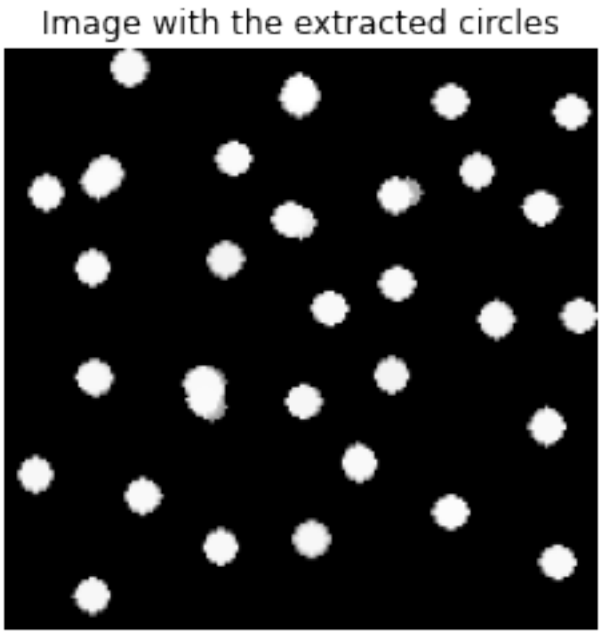
[97]:
```
SE = morphology.disk(4)
ax = plt.subplot(1,1,1)
ax.axis('off')
ax.imshow(SE, cmap = 'gray')
ax.set_title('Structuring element to extract circles')
plt.show()
print(SE.shape)
```




```
(9, 9)
```

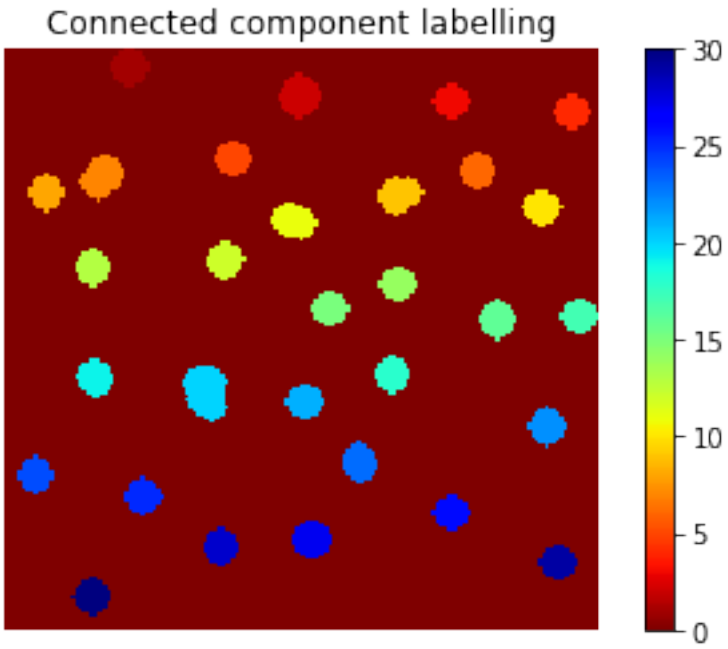
Used a disk as a structuring element of radius 4. The size of the SE is (9,9)

[98]:
```
I_open = np.copy(I)
I_open = morphology.opening(I_open,SE)

ax = plt.subplot(1,1,1)
ax.axis('off')
ax.imshow(I_open, cmap = 'gray')
ax.set_title('Image with the extracted circles')
plt.show()
```



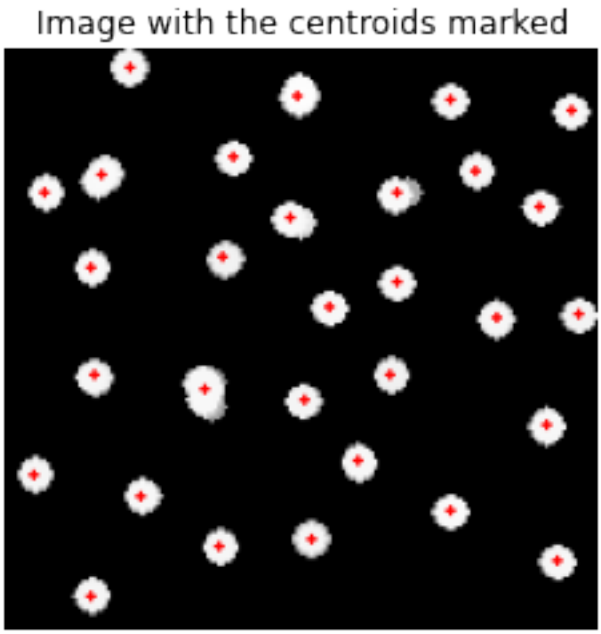
[99]:
```
label,num_features = scipy.ndimage.label(I_open)
fig, ax = plt.subplots()
ax.axis('off')
im = ax.imshow(label, cmap = plt.cm.jet_r, interpolation='nearest')
ax.set_title('Connected component labelling')
fig.colorbar(im, ax=ax)
```

[99]:
```
<matplotlib.colorbar.Colorbar at 0x7fca47deb9d0>
```



[100]:
```
centroid = []
area = []
for l in range(1,num_features+1):
    idx = np.where(label == l)
    y = sum(idx[1])/len(idx[1])
    x = sum(idx[0])/len(idx[0])
    a = len(idx[0])
    centroid.append((int(y), int(x)))
    area.append(a)
```

[101]:
```
I_centroid = np.stack([I_open, I_open, I_open], axis=-1)
for p in centroid:
    I_centroid = cv2.circle(I_centroid, (p[0],p[1]), radius=1, color=(255, 0,␣
 ↪0), thickness=-1)
ax = plt.subplot(1,1,1)
ax.axis('off')
ax.imshow(I_centroid)
ax.set_title('Image with the centroids marked')
plt.show()
```



[102]:
```
data = {};
data['Centoids'] = centroid
data['Area'] = area
pd.DataFrame(data)
```

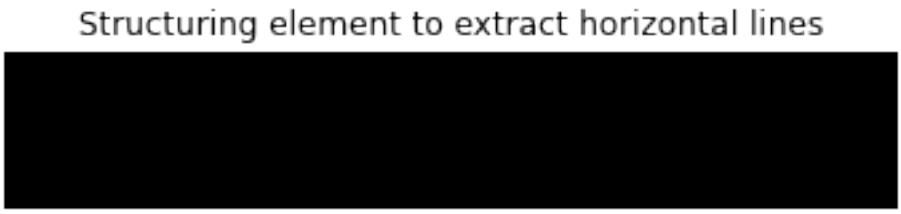
[102]:

	Centoids	Area
0	(35, 5)	89
1	(82, 13)	106
2	(125, 14)	78
3	(159, 17)	78
4	(64, 30)	78
5	(132, 34)	78
6	(27, 35)	109
7	(11, 40)	78
8	(110, 40)	97
9	(150, 44)	80
10	(80, 47)	97
11	(61, 58)	85
12	(24, 61)	78
13	(110, 65)	78
14	(91, 72)	78
15	(138, 75)	84
16	(161, 74)	78
17	(108, 91)	78
18	(25, 91)	85
19	(56, 95)	148

```
20    (84, 98)    78
21   (152, 105)    84
22    (99, 115)    84
23     (8, 119)    78
24    (38, 125)    81
25   (125, 129)    78
26    (86, 137)    89
27    (60, 139)    78
28   (155, 143)    78
29    (24, 153)    81
```

[103]:
```
I = cv2.imread('lines.jpg',0)
ax = plt.subplot(1,1,1)
ax.axis('off')
ax.imshow(I, cmap = 'gray')
ax.set_title('Original Image')
plt.show()
```

[104]:
```
SE = morphology.rectangle(3,17)
ax = plt.subplot(1,1,1)
ax.axis('off')
ax.imshow(SE, cmap = 'gray')
ax.set_title('Structuring element to extract horizontal lines')
plt.show()
print(SE.shape)
```



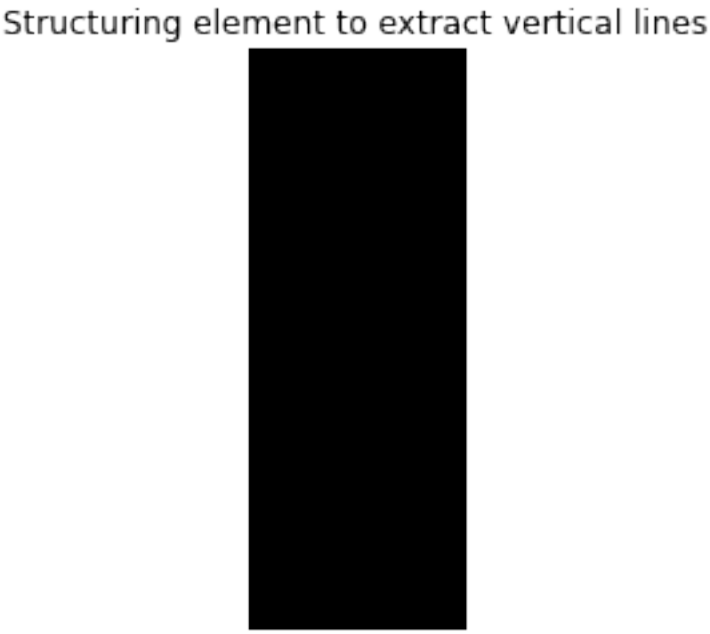

```
(3, 17)
```

Used a rectangle as a structuring element of length 17 and breadth 3. The size of the SE is (3,17)

[105]:
```
I_hor = I
# Extract the horizontal lines
I_hor = morphology.opening(I_hor,SE)
I_hor[I_hor>0] = 255

ax = plt.subplot(1,1,1)
ax.axis('off')
ax.imshow(I_hor, cmap = 'gray')
ax.set_title('Image with horizontal lines extracted')
plt.show()
```

[106]:
```
SE = morphology.rectangle(8,3)
ax = plt.subplot(1,1,1)
```

```
ax.axis('off')
ax.imshow(SE, cmap = 'gray')
ax.set_title('Structuring element to extract vertical lines')
plt.show()
print(SE.shape)
```




```
(8, 3)
```

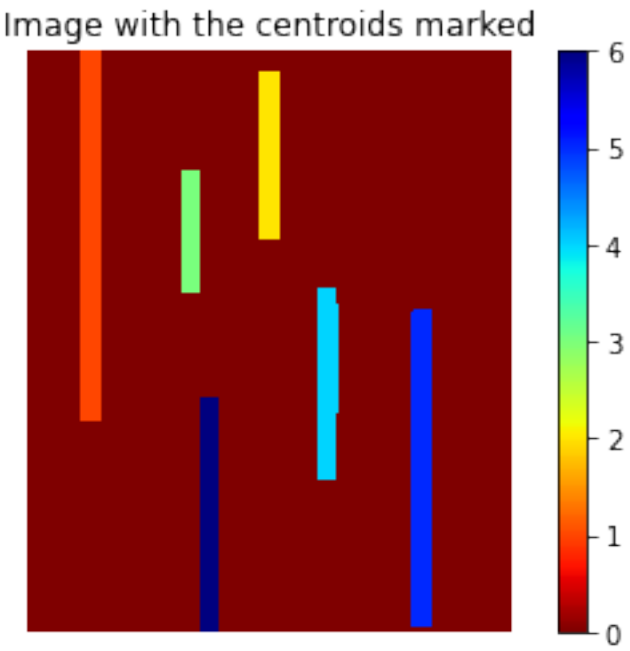
Used a rectangular structuring element of length 8 and width 3 to dilate the image. This was used to remove the horizontal lines completely (using mask subtraction) i.e. closing the gaps.

[107]:
```
# Use above as a mask to remove the horizontal lines from the image
I_op = I
I_op[I_hor>0] = 0
I_op = morphology.dilation(I_op, SE)
I_op[I_op<255] = 0

ax = plt.subplot(1,1,1)
ax.axis('off')
ax.imshow(I_op, cmap = 'gray')
ax.set_title('Image without the horizontal lines extracted')
plt.show()
```

[108]:
```
# Extract vertical lines from the image
SE = morphology.rectangle(10,5)
I_ver = I
# Extract the horizontal lines
I_ver = morphology.opening(I_ver,SE)
SE = morphology.rectangle(8,2)
I_ver = morphology.dilation(I_ver,SE)
I_ver[I_ver>0] = 255

ax = plt.subplot(1,1,1)
ax.axis('off')
ax.imshow(I_ver, cmap = 'gray')
ax.set_title('Image with vertical lines extracted')
plt.show()
```

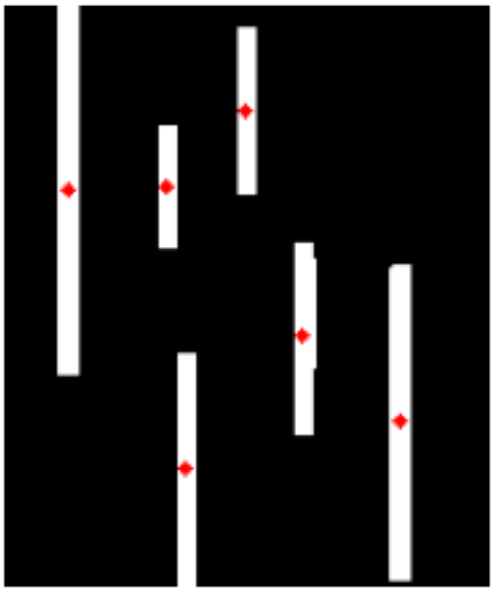
[109]:
```
label,num_features = scipy.ndimage.label(I_ver)
fig, ax = plt.subplots()
ax.axis('off')
ax.set_title('Image with the centroids marked')
im = ax.imshow(label, cmap = plt.cm.jet_r, interpolation='nearest')
fig.colorbar(im, ax=ax)
```

[109]:
```
<matplotlib.colorbar.Colorbar at 0x7fca4c3bc370>
```



[110]:
```
centroid = []
length = []
for l in range(1,num_features+1):
    mask_extract = label == l
    idx = np.where(mask_extract == True)
    y = sum(idx[1])/len(idx[1])
    x = sum(idx[0])/len(idx[0])
    centroid.append((int(y), int(x)))
    l = max(idx[0]) - min(idx[0])
    length.append(l)
```

[111]:
```
I_centroid = np.stack([I_ver, I_ver, I_ver], axis=-1)
for p in centroid:
    I_centroid = cv2.circle(I_centroid, (p[0],p[1]), radius=2, color=(255, 0,␣
 ↪0), thickness=-1)
ax = plt.subplot(1,1,1)
ax.axis('off')
ax.imshow(I_centroid)
plt.show()
```



[112]:
```
data = {};
data['Centoids'] = centroid
data['Length'] = length
pd.DataFrame(data)
```

[112]:

	Centoids	Length
0	(20, 58)	116
1	(76, 33)	52
2	(51, 57)	38
3	(94, 104)	60
4	(125, 131)	99
5	(57, 146)	73

Assignment2_Problem3

November 7, 2021

[1]:
```
import cv2
import numpy as np
from lloyd_python import lloyds
from matplotlib import pyplot as plt
```

Function to generate codebook and partition for uniform quantizer

[2]:
```
def generateUniform(s):
    '''
    performs uniform quantization
    on the image I.
    Outputs a s level image
    '''
    step = 255/pow(2,s)
    partition = np.arange(step,255,step)
    codebook = (np.arange(0,pow(2,s))+0.5)*step
    return partition, codebook
```

Function quantizes a images given a codebook and partiion

[3]:
```
def quantize(I, partition, codebook):
    '''
    Performs quantization on input image based on
    the codebook and partition passed
    Returns quantized image
    '''
    I_op = np.ones_like(I)*codebook[0]
    for i in range(len(partition)-1):
        I_op[np.logical_and(I>partition[i] ,I<=partition[i+1])]=codebook[i+1]
    I_op[I>partition[-1]] = codebook[-1]
    I_op = np.rint(I_op)
    return I_op
```

Function to perform uniform quantization

[4]:
```
def uniformQuantize(I,s):
    '''
    Fucntion first generates partition and codebook
```

```
    and then perfrom quantization using the 2
    '''
    partition, codebook = generateUniform(s)
    I_uniform = quantize(I,partition, codebook)
    return I_uniform
```

Function calculates MSE between 2 images

[5]:
```
def calculateMSE(I1, I2):
    '''
    Calculate the MSE between I1 and I
    '''
    MSE = np.mean(np.square(I1-I2))
    return MSE
```

Helper function to sweep across different bit lebels

[6]:
```
def sweep(I):
    MSE_Uniform = []
    MSE_MaxLLoyd = []
    m = I.shape[0]
    n = I.shape[1]
    for s in range(1,8):
        I_uniform = uniformQuantize(I,s)
        partition, codebook = lloyds(I.reshape((m*n,1)), [pow(2,s)])
        I_MaxQuant = quantize(I, partition, codebook)
        MSE_Uniform.append(calculateMSE(I, I_uniform))
        MSE_MaxLLoyd.append(calculateMSE(I, I_MaxQuant))
    return MSE_MaxLLoyd, MSE_Uniform
```

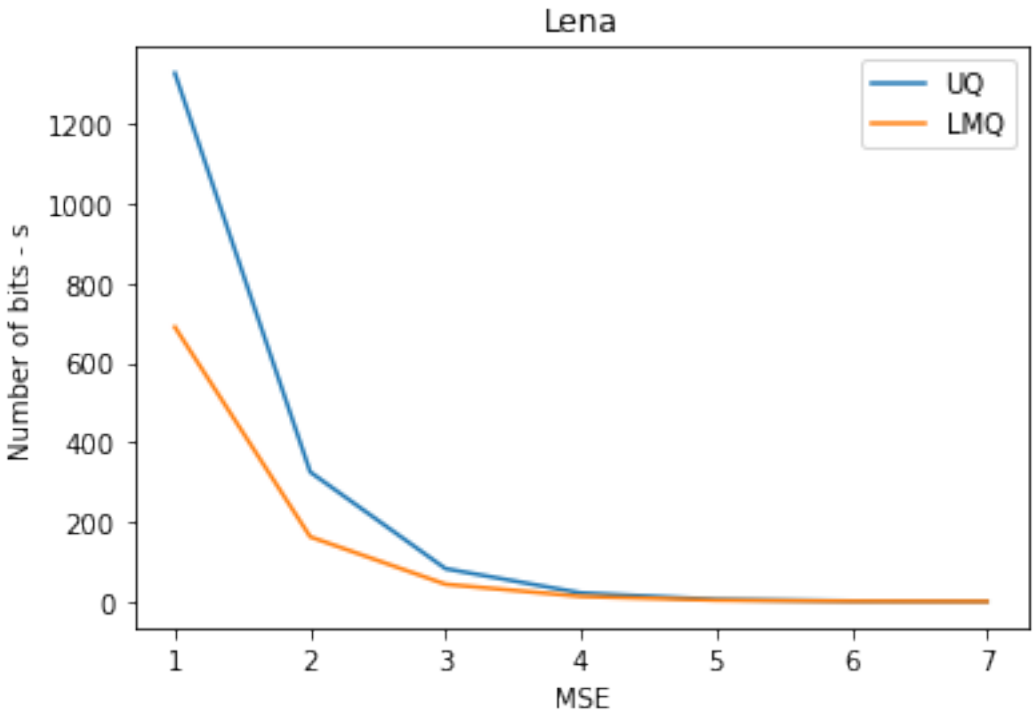
[7]:
```
I1 = cv2.imread('lena512.tif')
I1 = cv2.cvtColor(I1, cv2.COLOR_BGR2GRAY)
I2 = cv2.imread('diver.tif')
I2 = cv2.cvtColor(I2, cv2.COLOR_BGR2GRAY)
plt.subplot(1,2,1)
plt.imshow(I1, cmap= 'Greys')
plt.axis('off')
plt.subplot(1,2,2)
plt.axis('off')
plt.imshow(I2, cmap= 'Greys')
```

[7]:
```
<matplotlib.image.AxesImage at 0x7f5141ff36d0>
```

[8]:
```
I_uniform = uniformQuantize(I1,5)
```

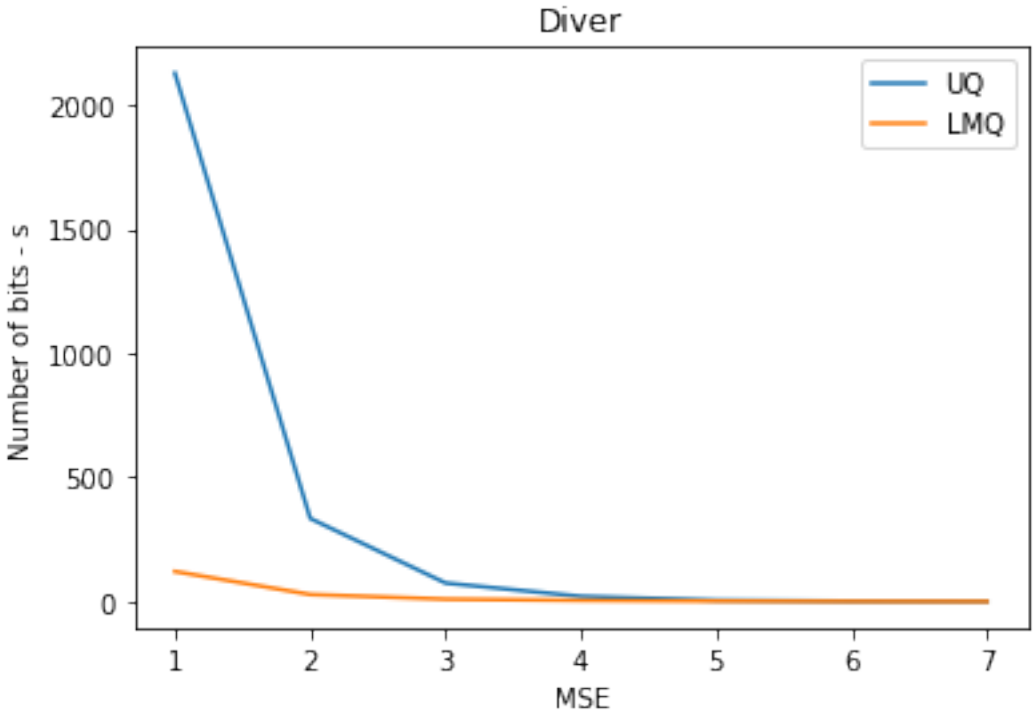
Unifrom Quantization vs Lloyd Max Quantization over Lena.tif

[9]:
```
MSE_MaxLLoyd, MSE_Uniform = sweep(I1)
plt.plot(np.arange(1,8),MSE_Uniform, label = 'UQ')
plt.plot(np.arange(1,8), MSE_MaxLLoyd, label = 'LMQ')
plt.legend()
plt.xlabel('MSE')
plt.ylabel('Number of bits - s')
plt.title('Lena')
plt.show()
```



Unifrom Quantization vs Lloyd Max Quantization over Diver.tif

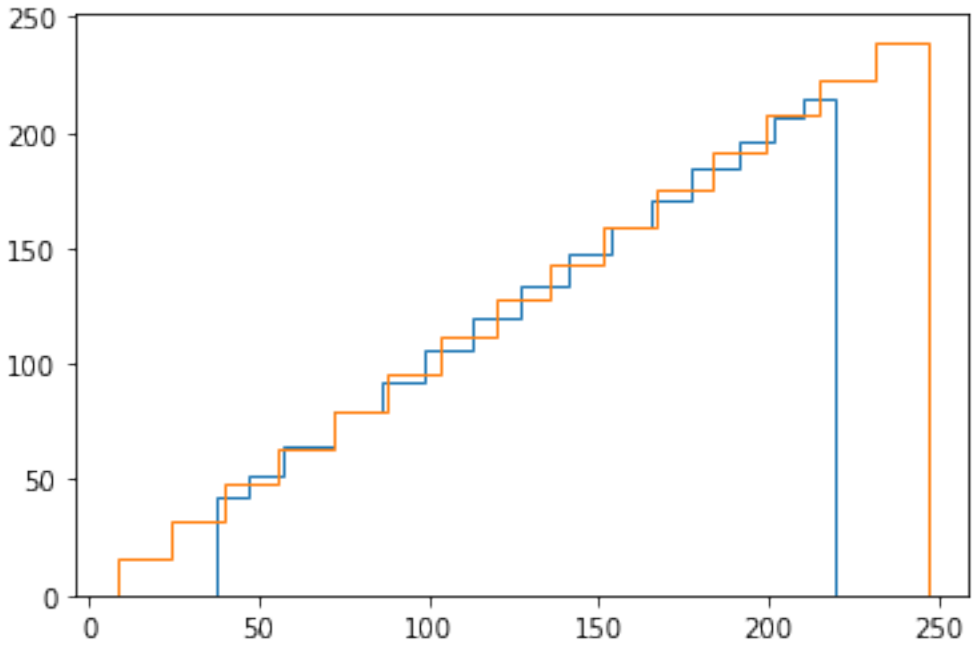
[10]:
```
MSE_MaxLLoyd, MSE_Uniform = sweep(I2)
ax = plt.plot(np.arange(1,8),MSE_Uniform, label = 'UQ')
ax = plt.plot(np.arange(1,8), MSE_MaxLLoyd, label = 'LMQ')
plt.title('Diver')
plt.xlabel('MSE')
plt.ylabel('Number of bits - s')
plt.legend()
plt.show()
```



(ii)
Uniform quantization show similar trends across the 2 images and perfroms worse in comparision to the LLoyd Max Quantizer.
The Lloyd Max Quantizer does a better job in preserving the information or reduces the loss of information by mapping the relevant region of the input values with more information (or more pixels) to finer levels and regions with lower amount of information (or lesser pixels) to wider levels. The plot below shows this for Lena, for s = 4
The performance gap between the 2 images is mainly because of the nature of the 2 images. Lena image has more contrast, that is it has a more spread out histogram than diver image. This leads to more loss of information or larger MSE with uniform quantization in comparision to Lloyd Max quantization

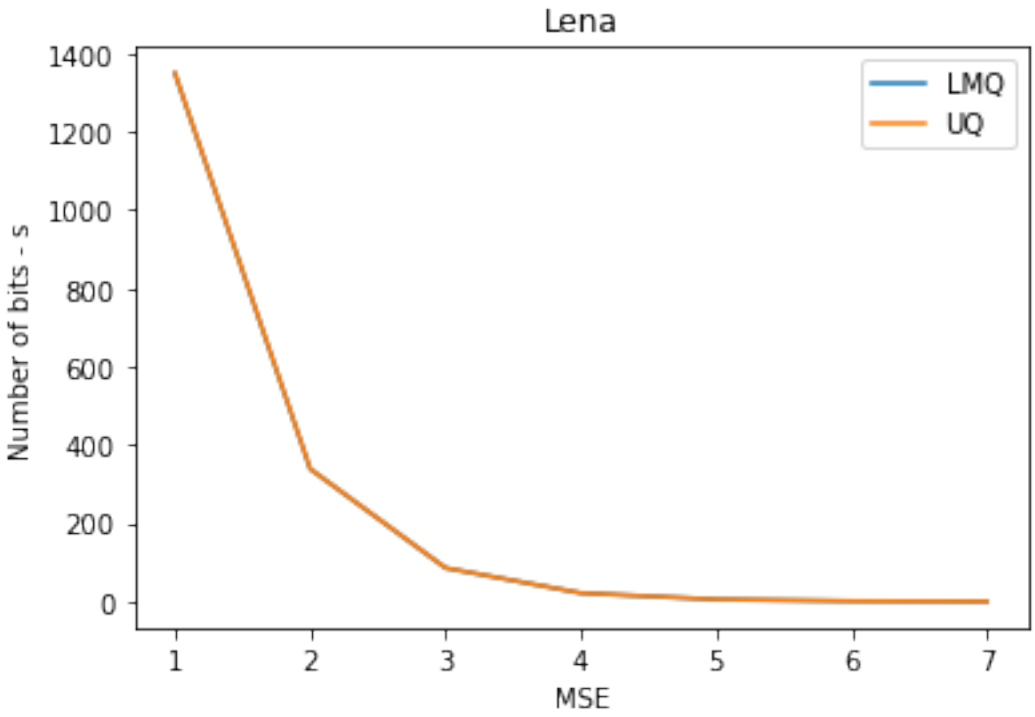
[11]:
```
m = I1.shape[0]
n = I1.shape[1]
partition, codebook = lloyds(I1.reshape((m*n,1)), [pow(2,4)])
plt.stairs(partition, codebook, label = 'LMQ')
partition, codebook = generateUniform(4)
plt.stairs(partition, codebook, label = 'UQ')
plt.legend
plt.show
```

[11]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



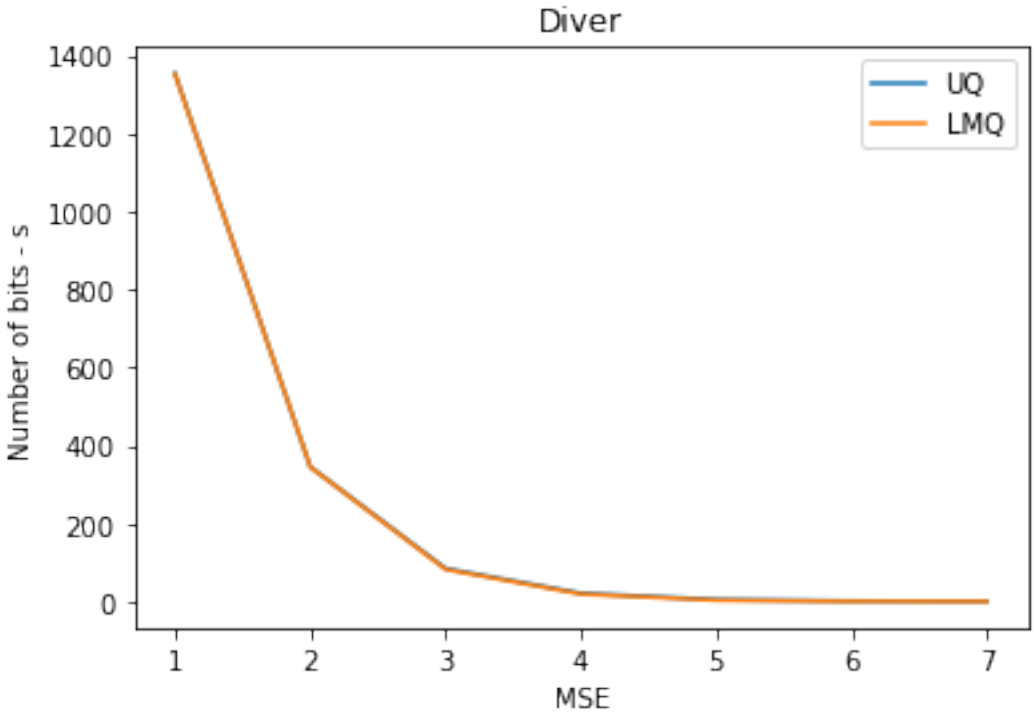
Unifrom Quantization vs Lloyd Max Quantization over Lena.tif after global histogram equalization

[14]:
```
I1 = cv2.equalizeHist(I1)
MSE_MaxLLoyd, MSE_Uniform = sweep(I1)
ax = plt.plot(np.arange(1,8), MSE_MaxLLoyd, label = 'LMQ')
ax = plt.plot(np.arange(1,8),MSE_Uniform, label = 'UQ')
plt.title('Lena')
plt.xlabel('MSE')
plt.ylabel('Number of bits - s')
plt.legend()
plt.show()
```



Unifrom Quantization vs Lloyd Max Quantization over Lena.tif after global histogram equalization

[13]:
```
I2 = cv2.equalizeHist(I2)
MSE_MaxLLoyd, MSE_Uniform = sweep(I2)
ax = plt.plot(np.arange(1,8),MSE_Uniform, label = 'UQ')
ax = plt.plot(np.arange(1,8), MSE_MaxLLoyd, label = 'LMQ')
plt.title('Diver')
plt.xlabel('MSE')
plt.ylabel('Number of bits - s')
plt.legend()
plt.show()
```



(iii)
Equalization leads to lloyd max quantizer behaving like a uniform quantizer.
(iv)
The MSE tends to zero for s = 7 as size of the level i.e. $t_i - t_{i-1}$ reduces as we increase s or the number of levels. The max possible quantization error is the least in this case which leads to near 0 MSE. Intuitively speaking, s = 7 has lowset loss of information due to the quantization process.Histogram equalization aids Lloyd Max quantizer similar to Uniform Quantizer. This is because after histogram equalization the Lloyd max quantizer tends to behave like a uniform quantizer as the pixel intensities are now uniformly distributed.

Assignment2_Problem4

November 7, 2021

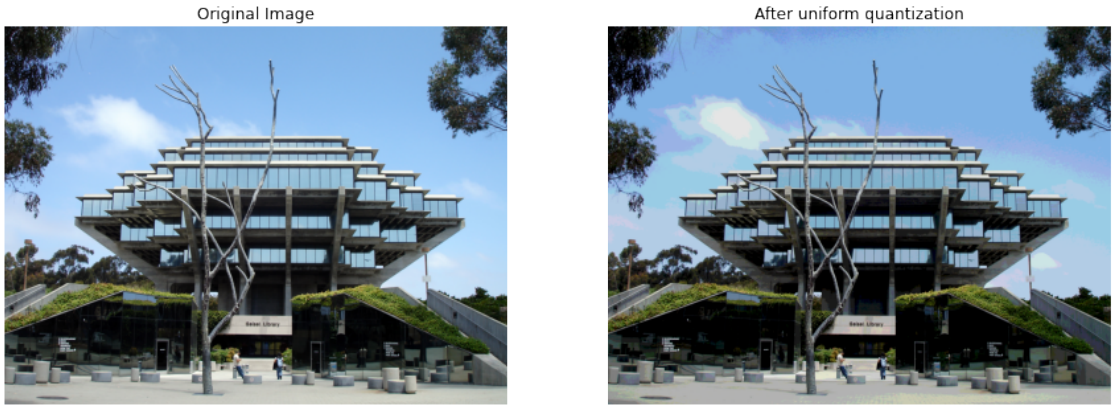
[171]:
```
import cv2
import numpy as np
from matplotlib import pyplot as plt
```

[172]:
```
def uniformQuantization(I):
    '''
    Quantizes the image to 10 levels
    Returns quantized image
    '''
    step = 255/10
    partitions = np.arange(step,255,step)
    codebook = np.arange(10)*step

    I[I<= partitions[0]] = codebook[0]
    for i in range(partitions.shape[0]-1):
        I[np.logical_and(I>partitions[i], I<=partitions[i+1])] = codebook[i+1]
    I[I>partitions[-1]] = codebook[-1]

    return I
```

[173]:
```
I = cv2.imread('geisel.jpg')
I = cv2.cvtColor(I, cv2.COLOR_BGR2RGB)
#I = cv2.imread('diver.tif')
#I = cv2.cvtColor(I, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(15,15))

plt.subplot(1,2,1)
plt.axis('off')
plt.title('Original Image')
plt.imshow(I)

I_uq = uniformQuantization(np.copy(I))

plt.subplot(1,2,2)
plt.axis('off')
plt.title('After uniform quantization')
```

```
plt.imshow(I_uq)
```

[173]: <matplotlib.image.AxesImage at 0x7f0ece8bf520>



Original image and quantized image

[174]:
```
def find_closest_palette_color(pixel):
    step = 255/10
    pixel_q = np.floor(np.copy(pixel)/step)*step
    return pixel_q
```

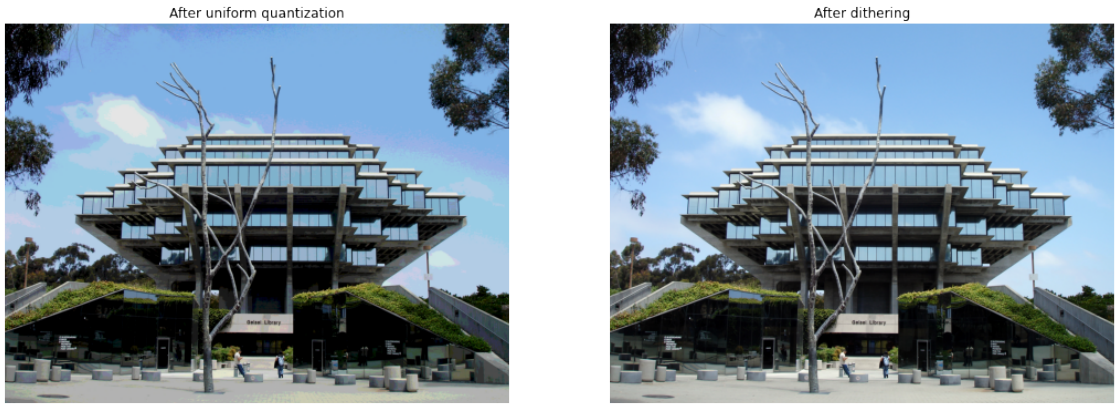
[175]:
```
def clip(pixel):
    return np.where(pixel>255,255,pixel)
```

[176]:
```
def FSD(I):
    '''
    Implements the Floyd Steinberg Dithering algorithm
    '''
    I_pad = np.pad(I, [(1, 1), (1, 1), (0,0)], mode='constant',␣
↪constant_values=0)
    I_pad = I_pad.astype('float')
    for j in range(1,I_pad.shape[1]-1):
        for i in range(1,I_pad.shape[0]-1):
            old_pixel = np.copy(I_pad[i,j,:])
            new_pixel = find_closest_palette_color(I_pad[i,j,:])
            I_pad[i,j,:] = new_pixel
            qError = old_pixel - new_pixel
            I_pad[i+1,j,:] = clip(I_pad[i+1,j,:] + qError * 7 / 16)
            I_pad[i-1,j+1,:] = clip(I_pad[i-1,j+1,:] + qError * 3 / 16)
            I_pad[i,j+1,:] = clip(I_pad[i,j+1,:] + qError * 5 / 16)
            I_pad[i+1,j+1,:] = clip(I_pad[i+1,j+1,:] + qError * 1 / 16)
    I_pad = I_pad[1:-2,1:-2,:]
    return I_pad.astype(np.uint8)
```

[177]:
```
I_dither = FSD(I)
```

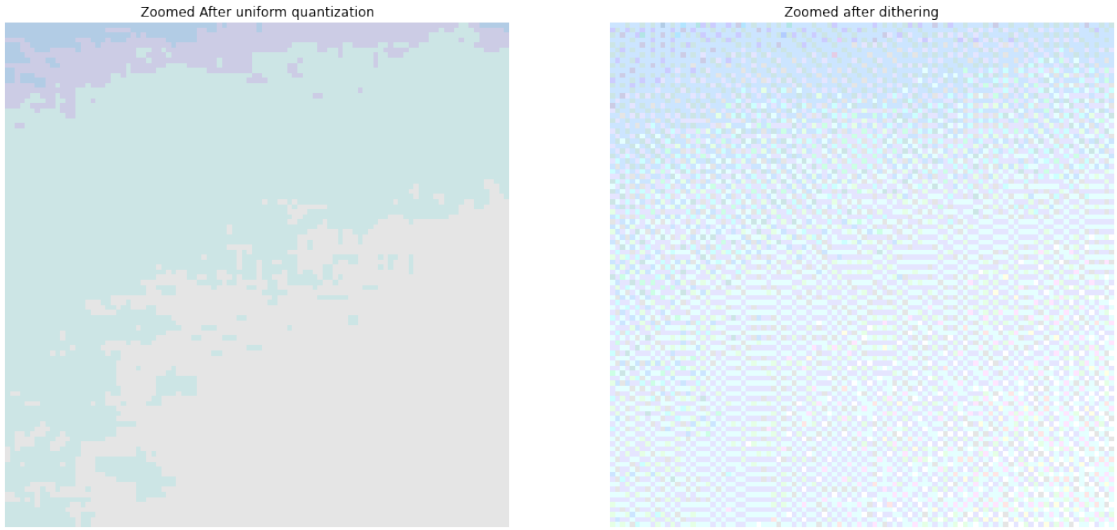
[190]:
```
plt.figure(figsize=(18,18))
plt.subplot(1,2,1)
plt.axis('off')
plt.title('After uniform quantization')
plt.imshow(I_uq)

plt.subplot(1,2,2)
plt.axis('off')
plt.title('After dithering')
plt.imshow(I_dither)
```

[190]:
```
<matplotlib.image.AxesImage at 0x7f0ecd2fd1c0>
```



[195]:
```
l = I_dither.shape[0]
b = I_dither.shape[1]
n = 100
k = 0.2
I_dither_zoomed = I_dither[int(k*l):int(k*l + n),int(k*b):int(k*b+n),:]
I_uq_zoomed = I_uq[int(k*l):int(k*l + n),int(k*b):int(k*b+n),:]

plt.figure(figsize=(18,18))
plt.subplot(1,2,1)
plt.axis('off')
plt.title('Zoomed After uniform quantization')
plt.imshow(I_uq_zoomed)

plt.subplot(1,2,2)
plt.axis('off')
plt.title('Zoomed after dithering')
plt.imshow(I_dither_zoomed)
```

[195]: <matplotlib.image.AxesImage at 0x7f0ecd0805e0>



1)
The resultant image after 10 level- quantization has a number of artifacts. We can especially observe this in the sky that the edges and border of the cloud gets lost. However, upon Floyd Steinberg Dithering, the resultant image looks almost similar to the orginal image.

2)
The quantized image has artifacts because of the quantization error introduced. This is eliminated in case od the dithered image. This is because by introducing a psuedo structured noise in the image we are trying to elminate the quantization error or distribute it in a smart way in order to make the quantized image look similar visually. More specifically, the algorithm is trying to push the quantization error of the current pixels to its neighbours that have not been visited yet. By doing so some pixels have a positive quantization error while other have a negative quantization error, which in total is summing to zero.