Purdue University, West Lafayette

Spring 2022

ECE 63700 – Digital Image Processing I

Lab-8

Submission Date- 04/14/2022

Submitted by

Nahian Ibn Hasan

PUID: 0032764564

Email: hasan34@purdue.edu

PURDUE

UNIVERSITY®

#Section 3.1

Original Image

Binary Thresholded Image

RMSE = 87.3933165438293
Fidelity = 77.33714917243346

Code for fidelity function:

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

def calculate_RMSE(X,Y):
	X = X.astype(float)
	Y = Y.astype(float)
	RMSE = np.sqrt(np.sum(np.power(X-Y,2))/np.prod(X.shape))
	return RMSE

def fidelity(X,Y,gamma,verbose=False):
	X_g_corrected = 255*np.power(X/255,gamma)
	X_b_g_corrected = 255*np.power(Y/255,gamma)
	plt.figure()
	plt.subplot(121)
	plt.imshow(X_g_corrected,cmap='gray')
	plt.title('Gamma Corrected Original')
	plt.subplot(122)
	plt.imshow(X_b_g_corrected,cmap='gray',interpolation='none')
	plt.title('Gamma Corrected Binary')
	plt.savefig('Gamma_Corrected_Image.png')

	X_filtered = np.zeros(X_g_corrected.shape)
	X_b_filtered = np.zeros(X_b_g_corrected.shape)

	#Low Pass FIlter
	X_g_corrected = np.pad(X_g_corrected,((3,3),(3,3)),mode='constant')
	X_b_g_corrected = np.pad(X_b_g_corrected,((3,3),(3,3)),mode='constant')
	normalizing_constant = 0.081438997244403883
	for i in range(3,X_g_corrected.shape[0]-3):
		for j in range(3,X_g_corrected.shape[1]-3):
```

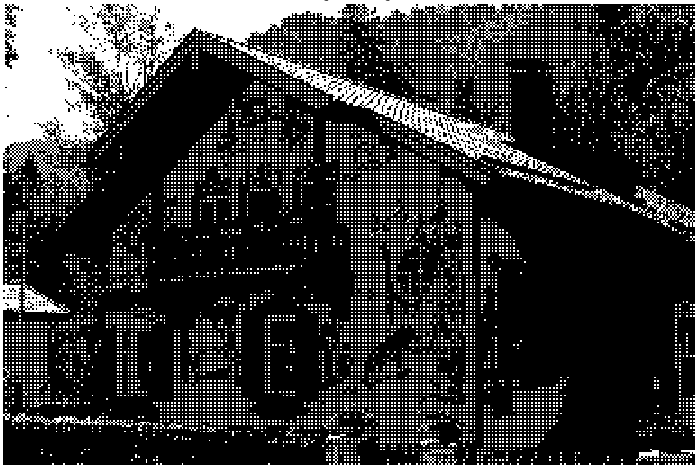
```
                sum1 = 0
                sum2 = 0
                for k in range(i-3,i+4):
                        for l in range(j-3,j+4):
                                sum1 = sum1 + np.exp(-((k-i)*(k-i)+(l-j)*(l-j))/4) * X_g_corrected[k,l]
                                sum2 = sum2 + np.exp(-((k-i)*(k-i)+(l-j)*(l-j))/4) * X_b_g_corrected[k,l]
                sum1 *= normalizing_constant
                sum2 *= normalizing_constant
                X_filtered[i-3,j-3] = sum1
                X_b_filtered[i-3,j-3] = sum2
        plt.figure()
        plt.subplot(121)
        plt.imshow(X_filtered,cmap='gray')
        plt.title('Filtered Original')
        plt.subplot(122)
        plt.imshow(X_b_filtered,cmap='gray',interpolation='none')
        plt.title('Filtered Binary')
        plt.savefig('Filtered_Image.png')
        if verbose:
                plt.show()
        X_filtered = 255*np.power(X_filtered/255,1/3)
        X_b_filtered = 255*np.power(X_b_filtered/255,1/3)

        fid = calculate_RMSE(X_filtered,X_b_filtered)
        return fid
```

#Section 4.2

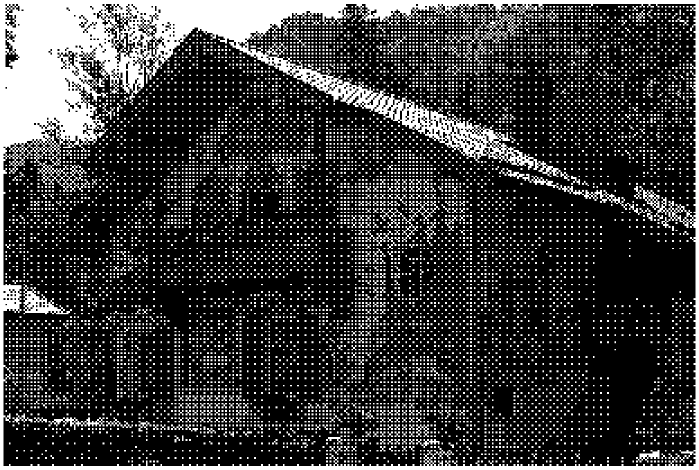
```
Bayer Matrix 2*2 =    [1 2
                        3 0]
Bayer Matrix 4*4 =    [ 5  9  6 10
                        13  1 14  2
                         7 11  4  8
                        15  3 12  0]
Bayer Matrix 8*8 =    [21 37 25 41 22 38 26 42
                       53  5 57  9 54  6 58 10
                       29 45 17 33 30 46 18 34
                       61 13 49  1 62 14 50  2
                       23 39 27 43 20 36 24 40
                       55  7 59 11 52  4 56  8
                       31 47 19 35 28 44 16 32
                       63 15 51  3 60 12 48  0]
```

Three half-toned images

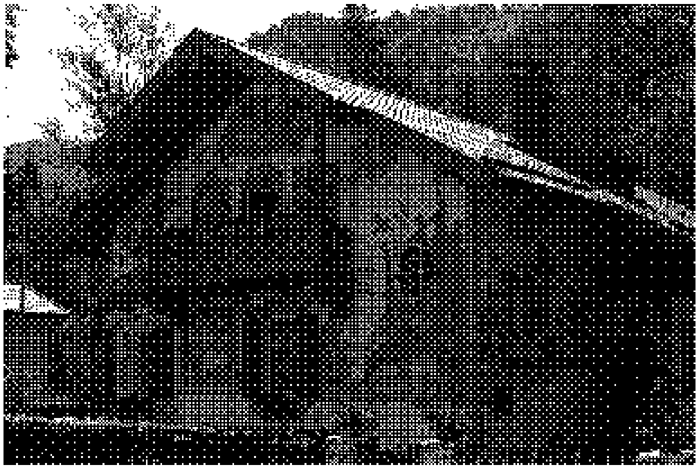
Half toned by Bayer matrix 2*2



Half toned by Bayer matrix 4*4



Half toned by Bayer matrix 8*8



Parameter	**Bayer Matrix 2*2**	**Bayer Matrix 4*4**	**Bayer Matrix 8*8**
RMSE	97.66897219213996	101.00692201569473	100.91452962396079
Fidelity	50.05694796030144	16.558342510690387	14.69177343381637

#Section 5.1

Code for Error Diffusion

```
### Ordered Dithering ##########
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
import utils as U

def Diffusion_Error(I,T):
	Im_out = np.zeros(I.shape)
	print(I.shape)
	I = np.pad(I,((1,1),(1,1)))
	print(I.shape)
	for i in range(1,I.shape[0]-1):
		for j in range(1,I.shape[1]-1):
			Im_out[i-1,j-1] = 255 if I[i,j] > T else 0
			error = I[i,j] - Im_out[i-1,j-1]
			I[i,j+1] = I[i,j+1]+7/16*error
			I[i+1,j-1] = I[i+1,j-1]+3/16*error
			I[i+1,j] = I[i+1,j]+5/16*error
			I[i+1,j+1] = I[i+1,j+1]+1/16*error
	return Im_out

def Main():
	name = '../house.tif'
	im = Image.open(name)
	X = np.array(im)
	gamma = 2.2
	Threshold = 127

	X_g_corrected = 255*np.power(X/255,gamma)
	Im_out = Diffusion_Error(X_g_corrected,Threshold)
	RMSE = U.calculate_RMSE(X,Im_out)
	fidelity = U.fidelity(X,Im_out,gamma,verbose=False)
	print('RMSE = ',RMSE)
	print('Fidelity = ',fidelity)

	Im_out = Image.fromarray(Im_out.astype(np.uint8))
	Im_out.save('Error_Diffusion.tif')
Main()
```

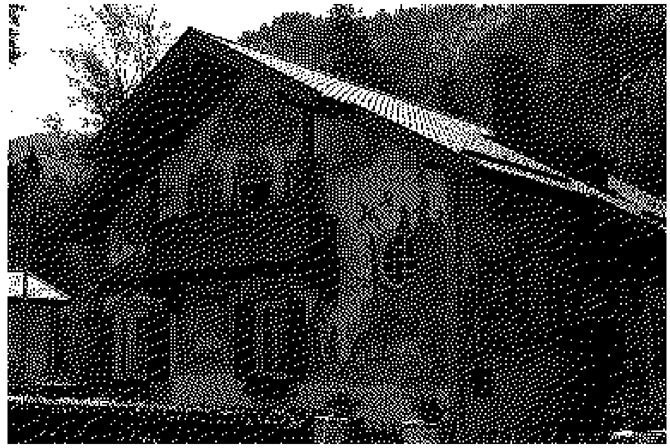
Code for utils.py

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

def calculate_RMSE(X,Y):
	X = X.astype(float)
	Y = Y.astype(float)
	RMSE = np.sqrt(np.sum(np.power(X-Y,2))/np.prod(X.shape))
	return RMSE

def fidelity(X,Y,gamma,verbose=False):
	X_g_corrected = 255*np.power(X/255,gamma)
	X_b_g_corrected = 255*np.power(Y/255,gamma)
	plt.figure()
```

```
plt.subplot(121)
plt.imshow(X_g_corrected,cmap='gray')
plt.title('Gamma Corrected Original')
plt.subplot(122)
plt.imshow(X_b_g_corrected,cmap='gray',interpolation='none')
plt.title('Gamma Corrected Binary')
plt.savefig('Gamma_Corrected_Image.png')

X_filtered = np.zeros(X_g_corrected.shape)
X_b_filtered = np.zeros(X_b_g_corrected.shape)

#Low Pass FIlter
X_g_corrected = np.pad(X_g_corrected,((3,3),(3,3)),mode='constant')
X_b_g_corrected = np.pad(X_b_g_corrected,((3,3),(3,3)),mode='constant')
normalizing_constant = 0.08143899724403883
for i in range(3,X_g_corrected.shape[0]-3):
	for j in range(3,X_g_corrected.shape[1]-3):
		sum1 = 0
		sum2 = 0
		for k in range(i-3,i+4):
			for l in range(j-3,j+4):
				sum1 = sum1 + np.exp(-((k-i)*(k-i)+(l-j)*(l-j))/4) * X_g_corrected[k,l]
				sum2 = sum2 + np.exp(-((k-i)*(k-i)+(l-j)*(l-j))/4) * X_b_g_corrected[k,l]
		sum1 *= normalizing_constant
		sum2 *= normalizing_constant
		X_filtered[i-3,j-3] = sum1
		X_b_filtered[i-3,j-3] = sum2
plt.figure()
plt.subplot(121)
plt.imshow(X_filtered,cmap='gray')
plt.title('Filtered Original')
plt.subplot(122)
plt.imshow(X_b_filtered,cmap='gray',interpolation='none')
plt.title('Filtered Binary')
plt.savefig('Filtered_Image.png')
if verbose:
	plt.show()
X_filtered = 255*np.power(X_filtered/255,1/3)
X_b_filtered = 255*np.power(X_b_filtered/255,1/3)

fid = calculate_RMSE(X_filtered,X_b_filtered)
return fid
```

Original Image

Error Diffusion Image



RMSE = 98.84711671109255
Fidelity = 13.427253039026654

Comparison among different half-tone images

Metrics	Ordered Dithering - Bayer Matrix 2*2	Ordered Dithering -Bayer Matrix 4*4	Ordered Dithering -Bayer Matrix 8*8	Error Diffusion	Simple Thresholding
RMSE	97.6689721921399	101.006922015694	100.914529623960	98.847116711092	87.39331654382
Fidelity	50.0569479603014	16.5583425106903	14.691773433816	13.427253039026	77.33714917243

Comments

The RMSE is almost similar for all of the halftone methods. On the other hand, for ordered dithering method, the image fidelity becomes lower as the Bayer matrix size increases. The error diffusion process performs as good as larger Bayer matrix. Image fidelity is much larger for the simple thresholding process. The lower the fidelity metric, the more realistic and the better the gray level tone is.

NB: The codes and report can be found in the following Github Directory.
Link: https://github.com/NahianHasan/ECE63700-Digital_Image_Processing/tree/main/Lab_8_Image_Halftoning