**HW 1 – ASSIGNMENT**

1) Demosaicing

import rawpy

import imageio

import cv2

from PIL import Image

import numpy as np

import math

import matplotlib.pyplot as plt

#########1a.reading and saving the image in png format and subsampling ##############################################

path = 'tetons.nef'

raw = rawpy.imread(path)

rgb = raw.postprocess()

imageio.imsave('tetons\_original.png', rgb)

k=cv2.imread('tetons\_original.png');

cvuint8 = cv2.convertScaleAbs(k)

print(cvuint8.dtype)

bayer = raw.raw\_image

print(bayer.shape[1])

print(bayer.shape[0])

rr=bayer[::2, ::2]

gg=(bayer[1::2, ::2]+bayer[::2, 1::2])\*0.5

bb=bayer[1::2, 1::2]

r1=(rr/float(np.max(rr)))\*255

g1=(gg/float(np.max(gg)))\*255

b1=(bb/float(np.max(bb)))\*255

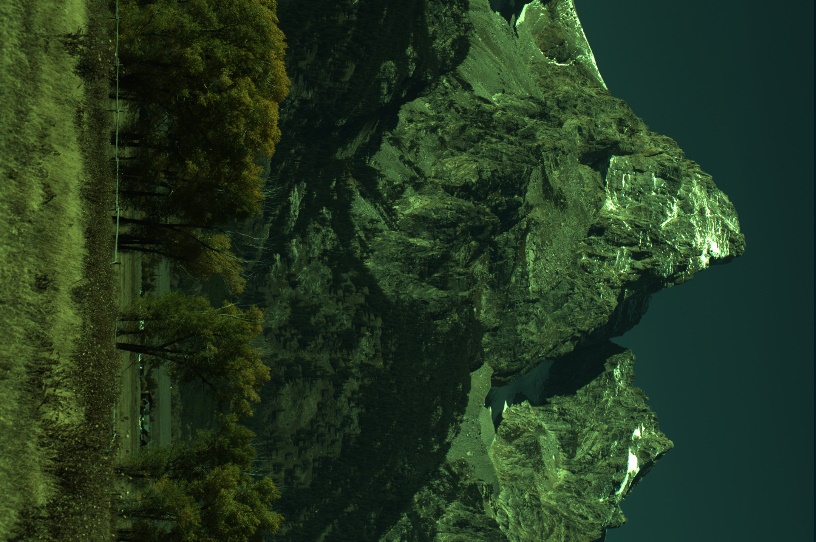
im=cv2.merge((b1,g1,r1))

cv2.imwrite("tetons\_subsample.png",im)

print(im.shape[1])

print(im.shape[0])

##################################################################

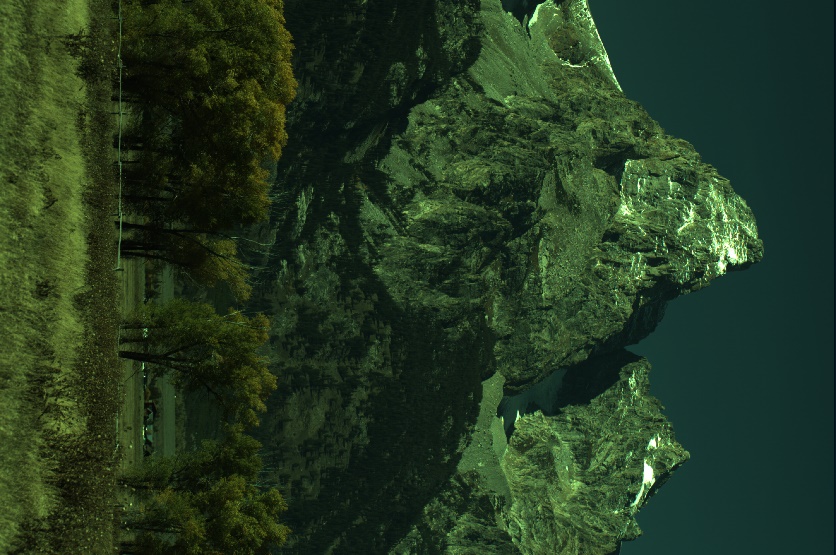


#########1b. nearest neighbour demosaicing ##############################################

img\_scaled2 = cv2.resize(im,None,fx=2, fy=2, interpolation = cv2.INTER\_NEAREST)

cv2.imwrite('tetons\_nn.png', img\_scaled2)

#####################################################################

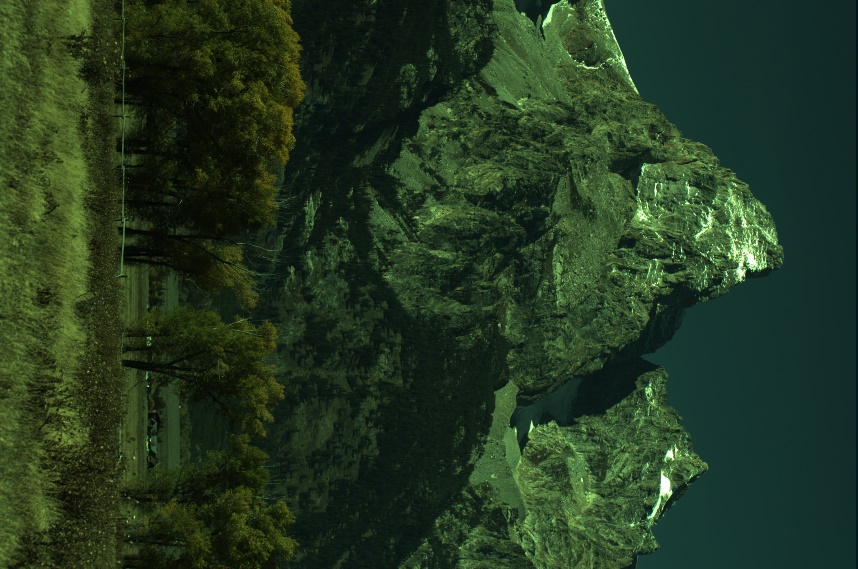


#########1c. Bilinear demosaicing ##############################################

img\_scaled3 = cv2.resize(im,None,fx=2, fy=2, interpolation = cv2.INTER\_LINEAR)

cv2.imwrite('tetons\_bl.png', img\_scaled3)

####################################################################



#########1d. Gunturk demosaicing ##############################################

rn=bayer[::2, ::2]

gn=np.zeros([int(bayer.shape[0]/2),int(bayer.shape[1]/2)])

bn=bayer[1::2, 1::2]

for x in range(0,bayer.shape[0],2):

for y in range(0,bayer.shape[1],2):

dh=np.abs(((bayer[x,0]+bayer[x,bayer.shape[1]-1])/2)-bayer[x,y])

dv=np.abs(((bayer[0,y]+bayer[bayer.shape[0]-1,y])/2)-bayer[x,y])

if dh>dv:

gn[int(x/2),int(y/2)]=(bayer[x-1,y]+bayer[x+1,y])/2

elif dh<dv:

gn[int(x/2),int(y/2)]=(bayer[x,y-1]+bayer[x,y+1])/2

else:

gn[int(x/2),int(y/2)]=(bayer[x-1,y]+bayer[x,y-1]+bayer[x+1,y]+bayer[x,y+1])/4

rk=(rn/float(np.max(rn)))\*255

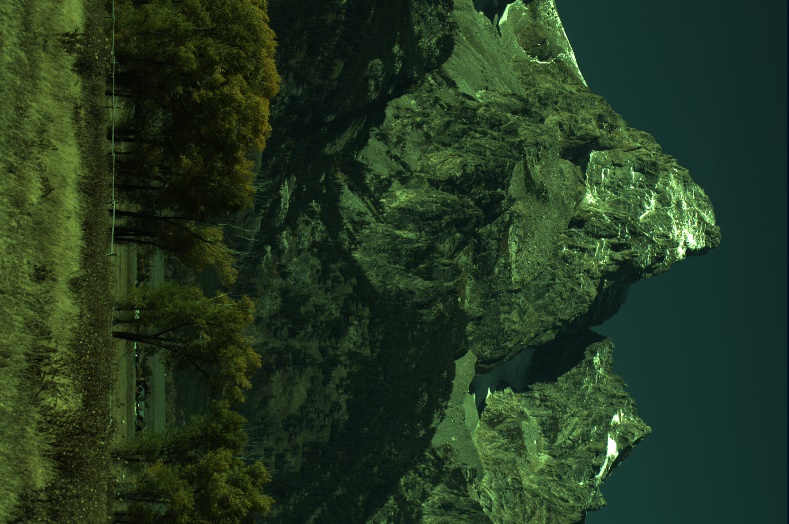
gk=(gn/float(np.max(gn)))\*255

bk=(bn/float(np.max(bn)))\*255

imk = cv2.merge((bk,gk,rk))

cv2.imwrite('tetons\_dm.png',imk)

#####################################################################



#########1e. Comparision of NN, Bilinear and Gunturk methods ##########################

img = cv2.imread('tetons\_nn.png')

img1 = cv2.imread('tetons\_bl.png')

img2 = cv2.imread('tetons\_dm.png')

fig = plt.figure()

a=fig.add\_subplot(2,3,1)

az = img[100:150,100:150]

imgk = plt.imshow(az)

a.set\_title('nearest neighbour')

plt.colorbar(orientation='horizontal')

a1=fig.add\_subplot(2,3,2)

az1 = img1[100:150,100:150]

imgk = plt.imshow(az1)

a1.set\_title('bilinear')

plt.colorbar(orientation='horizontal')

a2=fig.add\_subplot(2,3,3)

az2 = img2[100:150,100:150]

imgk = plt.imshow(az2)

a2.set\_title('gunturk')

plt.colorbar(orientation='horizontal')

a3=fig.add\_subplot(2,3,4)

az3 = img[150:200,150:200]

imgk = plt.imshow(az3)

a3.set\_title('nearest neighbour')

plt.colorbar(orientation='horizontal')

a4=fig.add\_subplot(2,3,5)

az4 = img1[150:200,150:200]

imgk = plt.imshow(az4)

a4.set\_title('bilinear')

plt.colorbar(orientation='horizontal')

a5=fig.add\_subplot(2,3,6)

az5 = img2[150:200,150:200]

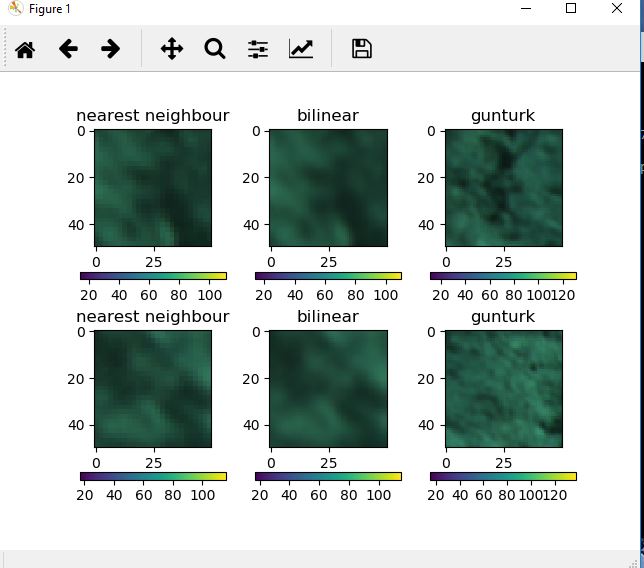
imgk = plt.imshow(az5)

a5.set\_title('gunturk')

plt.colorbar(orientation='horizontal')

plt.show()

#####################################################################

****

**2)HDR imaging**

import cv2

import numpy as np

import matplotlib.pyplot as plt

import rawpy

import imageio

from PIL import Image

import tifffile as tiff

import math

paths = [ ]

tk = []

for k in range(1,17):

paths.append('exposure%d.nef' % k )

tk.append(2\*\*(k-12))

#print(k)

#print(paths)

#print(tk)

i = 0;

########## section 3.a <processing raw images and resizing the images >#######################

for path in paths:

i= i +1

with rawpy.imread(path) as raw:

rgb = raw.postprocess(gamma=(1,1), no\_auto\_bright=True, output\_bps=16)

xnew,ynew=rgb.shape[1]/10,rgb.shape[0]/10

xnew = int(xnew)

ynew = int(ynew)

#print (xnew)

rgb=cv2.resize(rgb,(xnew,ynew))

imageio.imsave('processed\_exposure%d.tiff' % i,rgb)

####################################

########## section 3.b <calculating hdr image using the formula >#######################

Ihdr=np.zeros((400,600,3))

Ihdr1=0

Ihdr11=0

for i in range(0,400):

for j in range(0,600):

for c in range(0,3):

for k in range(1,17):

img=cv2.imread('processed\_exposure'+str(k)+'.tiff')

norm=img[i,j][c]/(255.0)

Ihdr1=Ihdr1+math.exp((-4\*(norm-0.5)\*\*2)/(0.5\*\*2))\*((Pex[i,j][c])/((1.0/2048)\*(2\*\*(k-1))))

Ihdr11=Ihdr11+math.exp((-4\*(norm-0.5)\*\*2)/(0.5\*\*2))

Ihdr[i,j][c]=Ihdr1/Ihdr11

Ihdr1=0

Ihdr11=0

#print( i )

####################################

########## section 3.c <displaying and saving the tonemapped image >#######################

cv2.imshow('image',lhdr)

cv2.waitKey(0)

cv2.destroyAllWindows()

cv2.imwrite('HDR\_phototonemap.png',Ihdr)

####################################



########## section 3. d<built In tonemapping >#######################

ps =[]

for k in range(1,17):

ps.append('processed\_exposure%d.tiff' % k )

img\_list = [cv2.imread(fn) for fn in ps]

ks = np.array(tk)

#print(type(ks))

# images alignment

alignMTB = cv2.createAlignMTB()

alignMTB.process(img\_list, img\_list)

merge\_robertson = cv2.createMergeRobertson()

hdr\_robertson = merge\_robertson.process(img\_list, times=ks)

#Tonemap HDR image with gamma and saturation parameters

tonemap2 = cv2.createTonemapDrago(1.2,0.7)

res\_robertson = tonemap2.process(hdr\_robertson.copy())

# Convert datatype to 8-bit and save

res\_robertson\_8bit = np.clip(res\_robertson\*255, 0, 255).astype('uint8')

cv2.imshow('image',res\_robertson\_8bit)

cv2.waitKey(0)

cv2.destroyAllWindows()

cv2.imwrite("ldr\_robertson\_builtintonemap.jpg", res\_robertson\_8bit)

####################################



**3Bilateral filtering**

import cv2

import numpy as np

import matplotlib.pyplot as plt

import rawpy

import imageio

from PIL import Image

import tifffile as tiff

import math

##########distance function#######################

def distance(x, y, i, j):

return np.sqrt((x-i)\*\*2 + (y-j)\*\*2)

#################################

########## gaussain function#######################

def gaussian(x, sigma):

return (1.0 / np.sqrt((2 \* math.pi \* (sigma \*\* 2)))) \* np.exp(- (x \*\* 2) / (2 \* sigma \*\* 2))

#################################

########## section 3.b <bilateral filtering function >#######################

def bilateral\_filter\_own(source, x, y, diameter, sigma\_i, sigma\_s):

#print(type(source))

hl = int(diameter/2)

i\_filtered = 0

Wp = 0

i = 0

while i < diameter:

j = 0

while j < diameter:

neighbour\_x = x - (hl - i)

neighbour\_y = y - (hl - j)

if neighbour\_x >= len(source):

neighbour\_x -= len(source)

if neighbour\_y >= len(source[0]):

neighbour\_y -= len(source[0])

gi = gaussian(source[neighbour\_x][neighbour\_y] - source[x][y], sigma\_i)

gs = gaussian(distance(neighbour\_x, neighbour\_y, x, y), sigma\_s)

w = gi \* gs

i\_filtered += source[neighbour\_x][neighbour\_y] \* w

Wp += w

j += 1

i += 1

return (i\_filtered / Wp)

###################################################

########## section 3.b <calling bilateral filtering function with following parameters>#######################

if \_\_name\_\_ == "\_\_main\_\_":

imh= cv2.imread('babyelephant.jpg')

diam = 5

ele\_new = np.zeros([imh.shape[0], imh.shape[1],3])

for c in range(0,3):

#print(src.shape[0])

#print(src.shape[1])

for l in range(0, imh.shape[0]):

for m in range(0, imh.shape[1]):

imh[l,m,c] = bilateral\_filter\_own(imh[:,:,c],l,m,diam,15,30)

cv2.imwrite('bilateral\_bl\_elephant.jpg', np.uint8(imh))

################################################################



########## section 3.a <Loading and displaying the image >#######################

#load image

img = cv2.imread('babyelephant.jpg')

#display image

cv2.imshow('image',img)

cv2.waitKey(0)

cv2.destroyAllWindows()

####################################

########## section 3.a <displaying the image after applying gaussian blur >#######################

#display image

dst = cv2.GaussianBlur(img, (5,5),0)

cv2.imshow('gaussian blurred image',dst)

cv2.waitKey(0)

cv2.destroyAllWindows()

cv2.imwrite("gaussianblurred.jpg", dst)

####################################



###################3.b comparision of original, gaussian blurred and bilateral images#######################

img = cv2.imread('original\_image\_grayscale.png')

img1 = cv2.imread('gaussianblurred.jpg')

img2 = cv2.imread('bilateral\_bl\_elephant.jpg')

fig = plt.figure()

a=fig.add\_subplot(2,3,1)

az = img[180:250,180:250]

imgk = plt.imshow(az)

a.set\_title('original')

plt.colorbar(orientation='horizontal')

a1=fig.add\_subplot(2,3,2)

az1 = img1[180:250,180:250]

imgk = plt.imshow(az1)

a1.set\_title('gaussian blurred')

plt.colorbar(orientation='horizontal')

a2=fig.add\_subplot(2,3,3)

az2 = img2[180:250,180:250]

imgk = plt.imshow(az2)

a2.set\_title('bilinear')

plt.colorbar(orientation='horizontal')

a3=fig.add\_subplot(2,3,4)

az3 = img[250:350,250:350]

imgk = plt.imshow(az3)

a3.set\_title('original')

plt.colorbar(orientation='horizontal')

a4=fig.add\_subplot(2,3,5)

az4 = img1[250:350,250:350]

imgk = plt.imshow(az4)

a4.set\_title('gaussian blurred')

plt.colorbar(orientation='horizontal')

a5=fig.add\_subplot(2,3,6)

az5 = img2[250:350,250:350]

imgk = plt.imshow(az5)

a5.set\_title('bilinear')

plt.colorbar(orientation='horizontal')

plt.show()

####################################################################

