CVE Assignment Report Outline

Question 2 (PS1-2)

Algorithm

The heart of the problem involves identifying the differentiating threshold between the bright and the dark areas of the image which are to be highlighted. The following algorithm describes the solution to the problem

1. Read image based on the filename provided
2. Specify the interested region to be highlighted (bright or dark)
3. Convert RGB image to Grayscale and plot the intensity histogram to identify the differentiating threshold
4. Use Binary based thresholding to create a binary file image (depending on the region to be highlighted)
5. Assign the area of interest red by maximizing the red channel output for those pixels in the interested area and tuning the other channels at that pixel to 0
6. Save and display the image in the required formats

Iterations to find the best threshold for each image provided the following results.

Results

Image: circuit.png

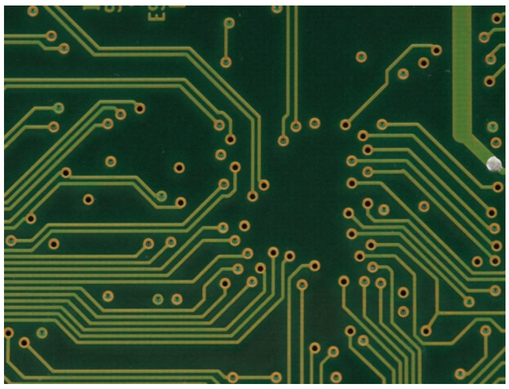
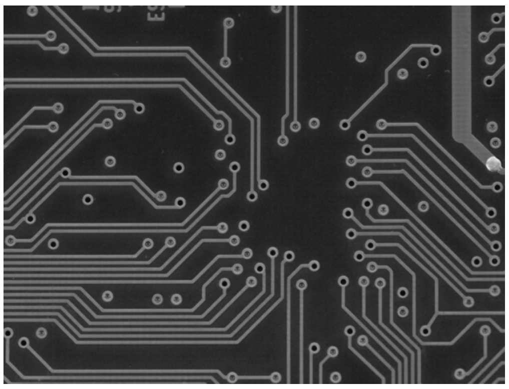
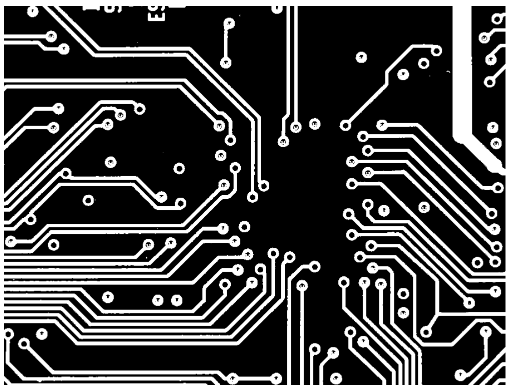
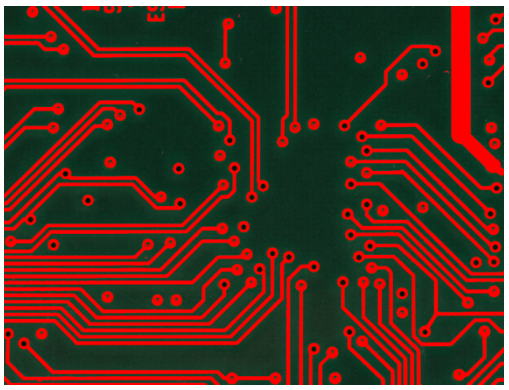
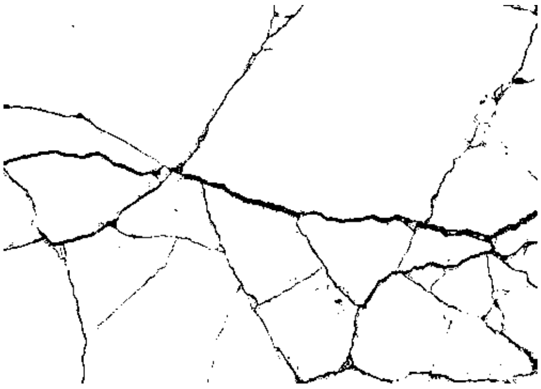
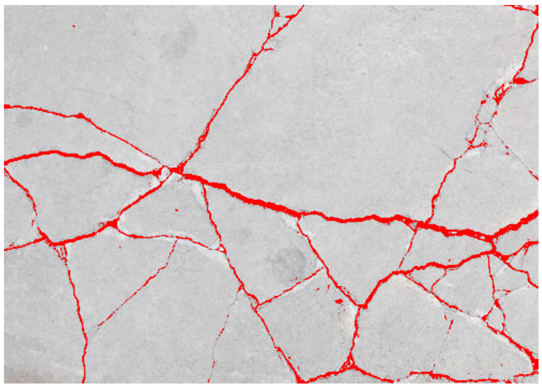
 1 Original Image	 2 Gray Scale Image
 3 Binary Image	 4 Final Image

Image: crack.png

5 Original Image	6 Gray Scale Image
 7 Binary Image	 8 Final Image

Source Code

```
import cv2 as cv
def rgb2gray (img):
    img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    # dst = cv.calcHist(img_gray, [0], None, [256], [0,256])
    # plt.hist(img_gray.ravel(),256,[0,256])
    # plt.title('Histogram for gray scale image')
    # plt.show()
    return img_gray
def grey2bin (img, op):
    if op:                  # For highlighting bright parts
        th, out_img = cv.threshold(img, 70, 255, cv.THRESH_BINARY)
        return out_img
    else:                   # For highlighting dark parts
        th, out_img = cv.threshold(img, 163, 255, cv.THRESH_BINARY_INV)
        return out_img
def main (img, name, op):  #Main Function
    cv.imshow('Input Color Image', img)
    img_gray = rgb2gray(img)            #Function to Convert color
image to greyscale
```

```
    fname = name[0]+'_grayscale'+'.'+name[1]
    cv.imwrite(fname, img_gray)
    img_bin = grey2bin(img_gray, op)    #Function to Convert to binary
map based on User choice
    fname = name[0]+'_binary'+'.'+name[1]
    cv.imwrite(fname, img_bin)

    rows = len(img_bin)
    cols = len(img_bin[0])

    img_out = img
    for x in range(0, rows):            # Converts white parts of the
binary image red in the final image
        for y in range(0, cols):
            if img_bin[x][y] == 0 and op == 0:  #For highlighting dark
parts
                img_out[x][y][0] = 0
                img_out[x][y][1] = 0
                img_out[x][y][2] = 255
            elif img_bin[x][y] == 255 and op == 1:  #For highlighting
bright parts
                img_out[x][y][0] = 0
                img_out[x][y][1] = 0
                img_out[x][y][2] = 255

    fname = name[0]+'_output'+'.'+name[1]
    cv.imwrite(fname, img_out)

    cv.imshow('Grayscale Image', img_gray)
    cv.imshow('Binary Image', img_bin)
    cv.imshow('Output Color Image', img_out)
    cv.waitKey(0)
    cv.destroyAllWindows()

filename = input("Enter File name : ")
img = cv.imread(filename)
filename = filename.split('.')
choice = input("Highlight Bright or Dark Portions? : ")
op = 0
if choice.lower() == "bright":
    op = 1
main(img, filename, op)
```

Question 3 (PS1-3)

Algorithm

The objective of the problem was to repeatedly change the intensity of the image based on a defined gamma value until an optimum image was found. The following algorithm describes the solution to the problem

1. Read the image from the given filename
2. Accept a gamma value for the gamma correction
3. Split the image into its respective channels
4. For each channel, modify the intensity at each pixel according to the following equation $New = 255 \times (Original/255)^{1/gamma}$
5. Merge the channels to create the final image
6. Display the new image along with the original image for comparison
7. Repeat from step 2 if a new gamma value is to be used, else escape the loop
8. Save the image

Iterations to find the best gamma value for each image provided the following results.

Results

Image: carnival.jpg

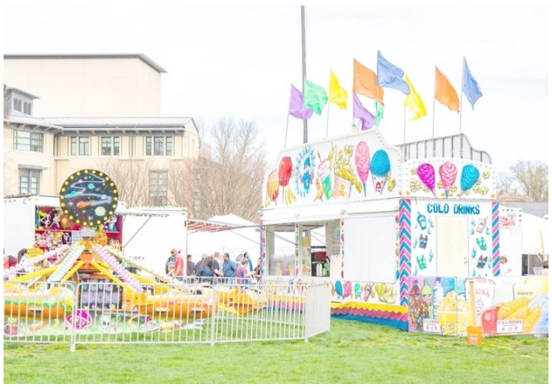
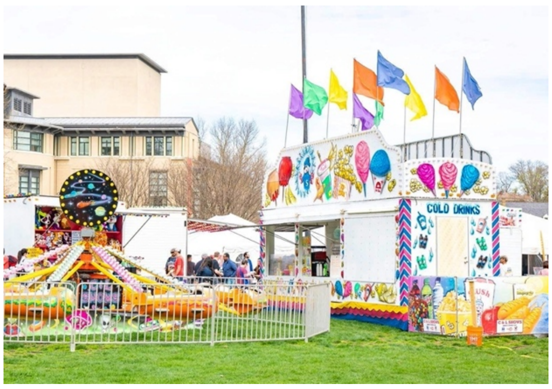
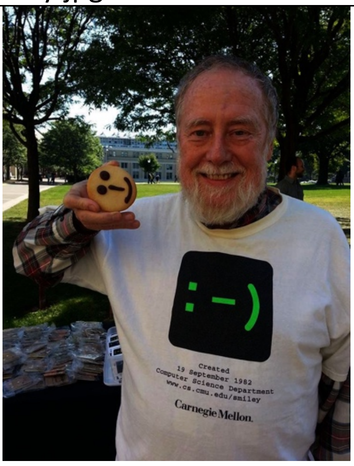
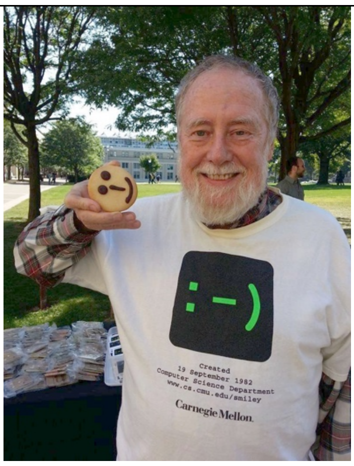
 9 Original Image	 10 Gamma Corrected Image

Image: smiley.jpg

 11 Original Image	 12 Gamma Corrected Image

Source Code

```
import cv2 as cv
import numpy as np

def gammacorrection(img, gamma):    #main function
    (B,G,R) = cv.split(img)     #Split image into individual channels
    B = np.array(255*(B/255)**(1/gamma), dtype='uint8') #Applying
correction formula on each channel
    G = np.array(255*(G/255)**(1/gamma), dtype='uint8')
    R = np.array(255*(R/255)**(1/gamma), dtype='uint8')
    img_out = cv.merge([B,G,R]) #merge channels
    return img_out

name = input("Enter file name = ")
img = cv.imread(name)
g = input("Enter Gamma value = ")
while(g != 'x'):
    cv.destroyAllWindows()      #Destroy all present windows
    nimg = gammacorrection(img, float(g))   #Gamma corrected image
    cv.imshow('Original Image', img)    #Display image
    cv.imshow('Edited Image', nimg)     #Display corrected image
    cv.waitKey(1000)                    #Stop windows from closing
    g = input("Enter Gamma value or press x to save image = ")  #new
gamma values

name = name.split('.')
fname = name[0]+'_gcorrected.'+name[1]
cv.imwrite(fname, nimg) #save file
cv.destroyAllWindows()
```

System Specifications

Operating System: macOS Monterey Version 12.5.1
Hardware: MacBook Air 2017 (Intel Core i5)
Python: Conda environment utilizing Python 3.9.1
IDLE: Visual Studio Code