Digital Image Processing (DIP)

Assignment#02

……………………………

Submitted by:

Talat Naeem 17I-0213

Submitted To:

Dr. Akhtar Jamil

Digital Image Processing Report of Assignment#02

Q-01:

```
import cv2 #importing the Open cv library
import numpy as np #importing numpy
from PIL import Image #libray for filtering image
from PIL import ImageFilter
import math #library forperforming math functions like log and root
etc
import matplotlib.pyplot as plt  #Library for ploting graphs
```

So, here first of all I have to import all the necessary libraries in the code which will be helpful in the questions. Initially imported open CV library for working on images then included python numpy array to perform different operations. Included the Math library to perfom mathematical operations like log nth root power etc. Then included matplot for plotting the graphs and other things to represent the output of the functions.

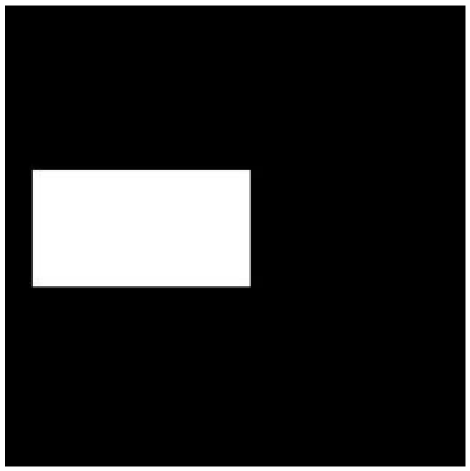
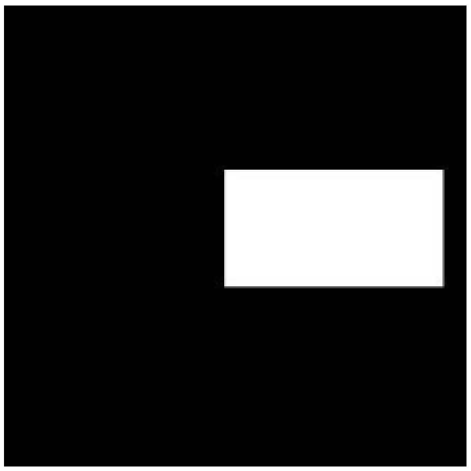
```
# Shift + tab  : Suggestions
#Q#1
path1 = r'D:\Fall 2021\DIP\Assignment02\Assignment # 2\img\Q-1a.jpg'
#path of image from local directory for image a Question#1
img1 = cv2.imread(path1,cv2.IMREAD_COLOR) #Function for reading image
from the path
path2 = r'D:\Fall 2021\DIP\Assignment02\Assignment # 2\img\Q-1b.jpg'
#path of image from local directory for image b Question#2
img2 = cv2.imread(path2,cv2.IMREAD_COLOR) #Function for reading image
from the path
```

In the above snippet of the code I read the image from my computer's directory by giving the path of the image and after that with the help of open cv function imread I read the image and Showed it as an output to check is it the same image or not.

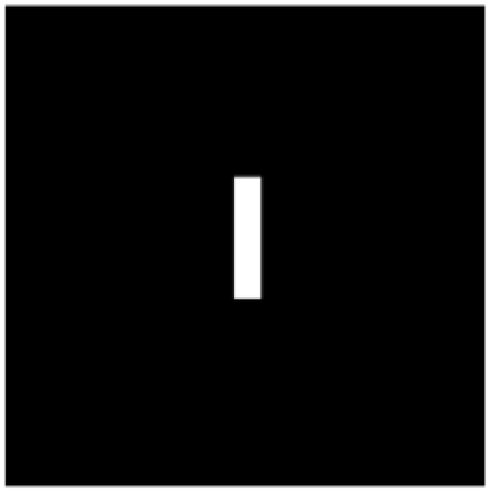
```
grayimg1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY) # convert color from
BGR TO RGB
grayimg2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY) # convert color from
BGR TO RGB
ret, img1thresh = cv2.threshold(grayimg1, 100, 255, cv2.THRESH_BINARY)
#Applying the threshold function on the coverted image
ret, img2thresh = cv2.threshold(grayimg2, 100, 255, cv2.THRESH_BINARY)
#Applying the threshold function on the coverted image
img_area = cv2.bitwise_and(img1thresh,img2thresh) #saving the
overlapping area of the image
plt.imshow(img_area,cmap='gray') #And Function for plotting the
image's region which is overlapping as output
plt.axis('off')
plt.show() #this function will show the image as output
```

Then the function which is shape function which tells us that the given image has how many rows and cols and also tell that is it a color image or grayscale image. And applying the threshold to check the overlapping area of the image. Value of threshold is set 255 just because the overlapped area is white. After that applied the bitwise-and operator which takes and of both images and return the result as the region which is overlapped by both the images. And plt.show() function to show the image as the output.

Original Image: Having two rectangles with some overlapped area

Overlapped Region: The overlapped area of above image



```
Now,
area_pixels=0 #Variable saving the value of area initially has value 0
for i in range(len(img_area)):  #A loop to calculate the area which is
overlapping in the two given images
    for j in range(len(img_area[i])):
        if(img_area[i][j]==255):
            area_pixels=area_pixels+1
```

```python
print("Overlapping Area : ",area_pixels) #Print the value of area
which was overlapped
```

The above snippet is used to calculate the value of pixels that are in the overlapped area of the given image. Run a loop and calculated the area of value of pixels that this region covered.

OutPut = **Overlapping Area: 1292**

Q-02:

In this question we are basically given a hand image including a portion of arm as well. All we have to do is that detect the area of hand and highlight it with green color and also calculate the area of parameter of the hand.

```
#Q#2
path3 = r'D:\Fall 2021\DIP\Assignment02\Assignment # 2\img\Q-2.jpg' #Q
2 image path from local directory
img3 = cv2.imread(path3,cv2.IMREAD_COLOR) #Function for reading image
from the path
grayimg3 = cv2.cvtColor(img3,cv2.COLOR_BGR2GRAY) # convert color to a
grayscale image
img3 = cv2.cvtColor(img3,cv2.COLOR_BGR2RGB) # convert color from BGR
TO RGB
plt.imshow(img3) #Function for plotting the image as output
plt.axis('off')
plt.show() #this function will show the image as output
```

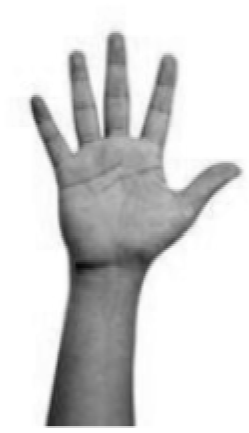
The above snippet is basically used to take the image from local directory and converting that RGB image into BGR and simply after applying basic functions just output the given image as it is with little modifications for further processing. Output of the above snippet is as follows:

Original Image:

```
plt.imshow(grayimg3,cmap='gray') #Function for plotting the image as
output
plt.axis('off')
plt.show() #this function will show the image as output
```

This little snippet is used to convert that original image into a gray scale image and we did it because further operations are easily applicable on a gray scale image.

Gray Scale Image:



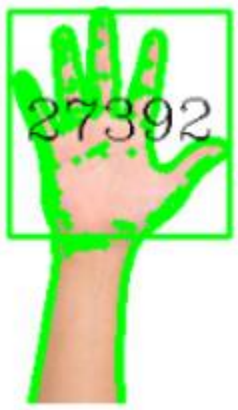
```
#making contours function to get shape boundary of the given hand
image
def getContours(img):

contours,hierarchy=cv2.findContours(img,cv2.RETR_EXTERNAL,cv2.CHAIN_AP
PROX_NONE)
    for cnt in contours:
        cv2.drawContours(imgContour2,cnt,-1,(0,255,0),3)
        peri=cv2.arcLength(cnt,True)
        approx=cv2.approxPolyDP(cnt,0.001*peri,True)
        x,y,w,h=cv2.boundingRect(approx)
    cv2.rectangle(imgContour2,(x-(w//2)-
5,y),(x+w,y+(h//2)+15),(0,255,0),2)
    parameter=0
    for i in range(len(imgContour2)):
        for j in range(len(imgContour2[i])):
            if(imgContour2[i][j][1]==255):
                parameter=parameter+1
    parameter=str(parameter)
    cv2.putText(imgContour2,parameter,(w//2-
20,(h//2)+10),cv2.FONT_HERSHEY_COMPLEX,1,(0,0,0),1)
    plt.imshow(imgContour2) #Function for plotting the image as output
    plt.axis('off')
    plt.show() #Function for showing the image as output
```

This above function is used to get a shape of the boundary of the hand which is our aim to detect in the given question. And we made a function **getContours** and passed it an image which we are using in this question after applying the gray scale image function. In short this is used to draw a boundary around the hand in the given image we used a loop in that function to count the parameters of the hand as well. In the variable **parameter** that's initial value is set 0 then in the nested loop the count will increase by one whenever the parameters are detected by that function. And finally drawn a rectangle as well of the green color to highlight the area of hand specifically in the full image. And in the end the value of parameter will also be shown in the highlighted area of the hand.

OutPut = **Value of parameter: 27392**

Resultant Image:



There is the highlighted hand in the given image.

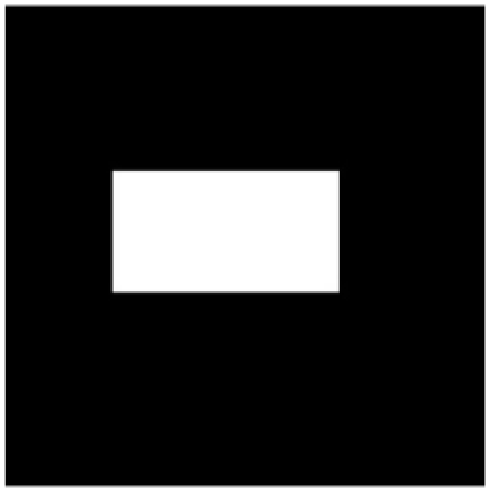
Q-03:

As we know that **Sobel Operator** is an edge detector which is used to detect the edges after sharpening the image and then have to calculate the area of that image using those corners.

```
#Q#3
path4 = r'D:\Fall 2021\DIP\Assignment02\Assignment # 2\img\Q-3.jpg'
#path of image from local directory for image of Question#3
img4 = cv2.imread(path4,cv2.IMREAD_COLOR) #Function for reading image
from the path
grayimg4 = cv2.cvtColor(img4,cv2.COLOR_BGR2GRAY) # convert the image
to grayscale image
img4 = cv2.cvtColor(img4,cv2.COLOR_BGR2RGB) # convert color from BGR
TO RGB
plt.imshow(img4) #Function for plotting the image as output
plt.axis('off')
plt.show() #this function will show the image as output
```

In the above snippet we simply access the image from the local directory and converted it in the gray scale image to perform further operations and just showed it as an output.

Original Image:



This is the original image and now we have to apply Sobel Operator to detect the edges.

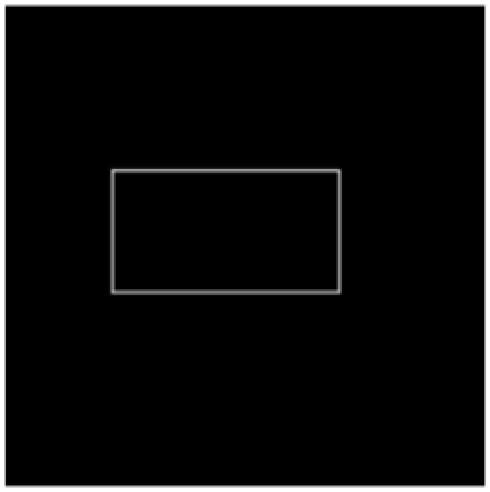
```
ret, img4thresh = cv2.threshold(grayimg4, 100, 255, cv2.THRESH_BINARY)
#applying the threshold on the given image
area_pixels=0 #Variable saving the value of area initially has value 0
for i in range(len(img4thresh)): #Loop for counting the area of pixels
of the boundry
    for j in range(len(img4thresh[i])):
        if(img4thresh[i][j]==255):
            area_pixels=area_pixels+1
print("Area : ",area_pixels) #printing the value of pixels area
```

```
img4y = cv2.Sobel(img4thresh,cv2.CV_64F,1,0,ksize=3) #applying the
sobel operator for sharpning the image
img4yabs = np.absolute(img4y)
img4x = cv2.Sobel(img4thresh,cv2.CV_64F,0,1,ksize=3) #Sobel is helpful
for sharpning the image for detecting the errors
img4xabs = np.absolute(img4x) #getting the absolute value after
applying sobel operators
img4xy=img4yabs+img4xabs
plt.imshow(img4xy,cmap='gray') #Function for plotting the image as
output
plt.axis('off')
plt.show() #this function will show the image as output
```

The above snippet is where we applied the **Sobel Operator** and detected the edges or we can say the boundary of the image furthermore with **area_pixels** variable we used to count the area of the image and showed it with the functions like show() and imshow().

OutPut = **Area: 10792**

Resultant Output Image:



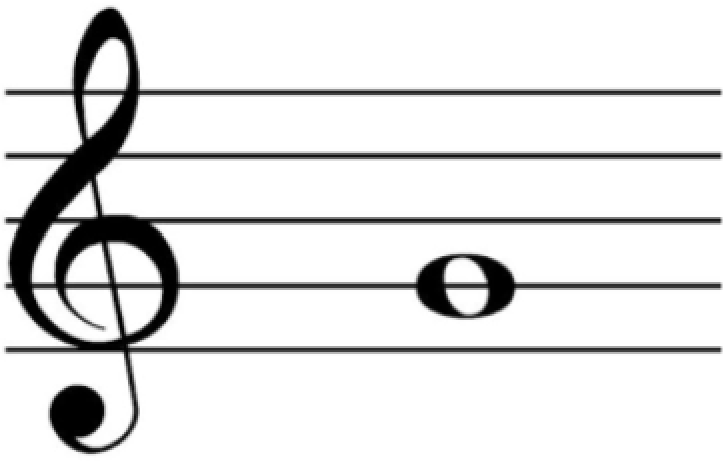
Q-04:

In this question we have to detect only horizontal lines and have to remove the above images on those lines.

```
#Q#4
path5 = r'D:\Fall 2021\DIP\Assignment02\Assignment # 2\img\Q-4.jpg' #path of
image from local directory for image of Question#4
img5 = cv2.imread(path5,cv2.IMREAD_COLOR) #read the image from the local
directory in the notebook
grayimg5 = cv2.cvtColor(img5,cv2.COLOR_BGR2GRAY) #converting the given image
into gray scale image
ret, threshimg5 = cv2.threshold(grayimg5, 100, 255, cv2.THRESH_BINARY)
#applying the threshold operation
negimg5= cv2.bitwise_not(threshimg5) #Bitwise not operator to make the image
negative white to black
plt.imshow(grayimg5,cmap='gray')  #Function for plotting the image as output
plt.axis('off')
plt.show() #Function for plotting the image as output
```

The above snippet is used to take the image from the local directory and convert it into the gray scale image for further processing and applied threshold and then **bitwise_not** applied to convert the white image into black image and finally showed it as an output.

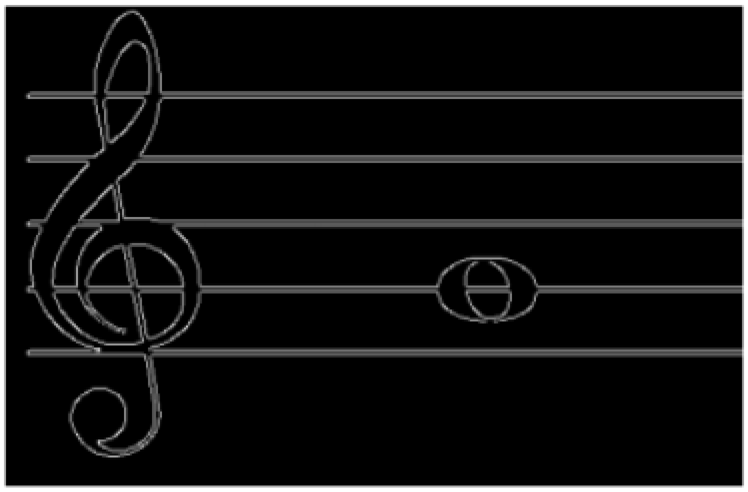
Original Image:



```
cannyimg5=cv2.Canny(negimg5,60,60) #applied canny edge detector to highlight
the edges of the image
plt.imshow(cannyimg5,cmap='gray') #function to show the output of the image
plt.axis('off')
plt.show() #function used to show resultant image as output
```

In the above small snippet we applied **Canny** edge detector on the negative image so that only we can highlight only the edges specifically.

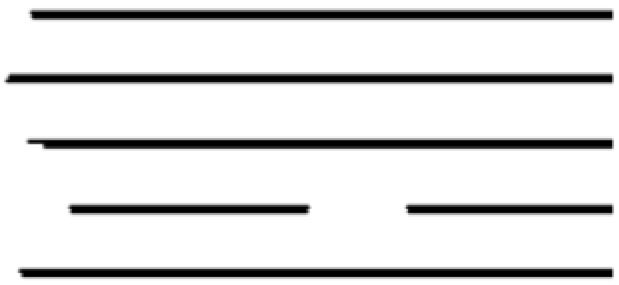
Canny Detected Image:



```
newimg5=np.zeros((grayimg5.shape[0],grayimg5.shape[1]),np.uint8) #applied
numpy matrix function to make an image of specific size where all values are
zero
theta=m.pi/2
horizontal_lines = cv2.HoughLinesP(cannyimg5, 1,theta,2, None, 25, 2) #Hough
line detector used here
i =0
while (i<12):  #This is the loop to access the horizontal lines of the given
image
    for j in horizontal_lines[i]:
        start_pt = (j[0],j[1])
        end_pt = (j[2],j[3])
        cv2.line(newimg5, start_pt, end_pt, (255,255,255), 2)
    i=i+1
negnewimg5 = cv2.bitwise_not(newimg5) #Applied bitwise_not operator on the
image to get updated image
plt.imshow(negnewimg5,cmap='gray') #Function to show image as an output
plt.axis('off')
plt.show() #Function is used the show the image as an output
```

In the above snippet I used initially an numpy array **np.zeros** where initially all the values of matrix are zero but the size of matrix is same as original image. After that using **Hough Transformation** detected the horizontal rows of the image and the result will be absurd and unlinked lines will be drawn. And in the end negative the image again so that we can get the original image color.

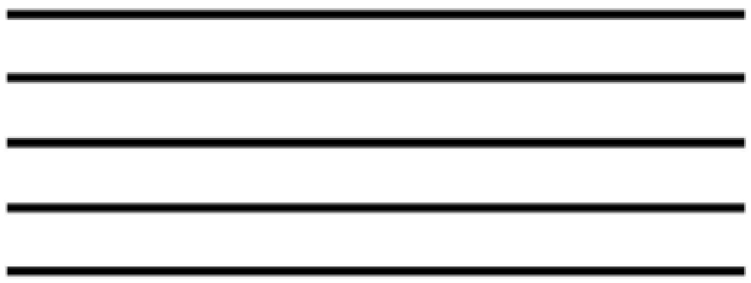
Hough Transformed Image:



```
struct_ele = np.ones((1,350), np.uint8) #numpy function to make all matrix
values one for structural element
dilatedimg5 = cv2.dilate(newimg5, struct_ele, iterations=1) #dialtated the
resultant image for edge linking and reshaping
errodedimg5 = cv2.erode(dilatedimg5, struct_ele, iterations=1) #erod the
image for removing pixels from the boundary of the image
negerrodedimg5 = cv2.bitwise_not(errodedimg5)
plt.imshow(negerrodedimg5,cmap='gray') #function to show the output image
plt.axis('off')
plt.show() #function to show the image as an output
```

In the above snippet we applied the **dialation** just to connect the edges of the horizontal lines. And then applied **erode** to remove the noise and pixels from the edges of the boundary.

Resultant Image:



This will be the resultant output image after applying all the operations.

~~~

``````````````````````````````````````````````````````````````````````````````````````

``````````````````````````````````````````````````````````````````````

**Note:** As in this assignment I have completed all the questions **(1,2,3 and4)** and I am submitting the code file as well and I have included snippet of codes here as well so that, it will be easy for you to check them all.
~~~