**PYTHON FOR VISION TECHNIQUES**

**Name:** Vatsal Panchal

**Registration number:** 18BEC1125

1. Capture an image of you in an indoor environment and perform the following subtasks:
2. Plot the histogram of the grayscale version of the image; and
3. Perform any one smoothing technique (mean filtering, Gaussian filtering or median filtering)

**CODE:**

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

img=mpimg.imread('C:/Users/WELCOME/Downloads/Snapchat-118920799.jpg')

def rgb2gray(rgb):

r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]

gray = 0.2989 \* r + 0.5870 \* g + 0.1140 \* b

return gray

img\_gray=rgb2gray(img)

plt.figure(1)

plt.imshow(img\_gray,cmap='gray')

plt.figure(2)

plt.show()

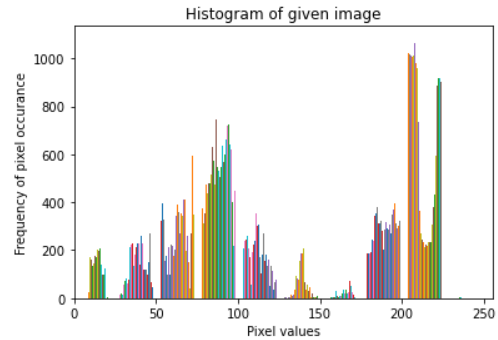
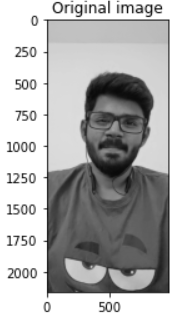
plt.hist(img\_gray)

plt.title("Histogram of given image")

plt.xlabel("Pixel values")

plt.ylabel("Frequency of pixel occurance")

plt.xlim([0, 255])



import numpy as np

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

from scipy import ndimage # Multi-dimensional image processing

img=mpimg.imread('C:/Users/WELCOME/Downloads/Snapchat-118920799.jpg')

def rgb2gray(rgb):

r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]

gray = 0.2989 \* r + 0.5870 \* g + 0.1140 \* b

return gray

img=rgb2gray(img)

img\_noise = img + 0.05\* np.random.randn(\*img.shape)

blurred\_img1=ndimage.gaussian\_filter(img\_noise, sigma=1.5) # Multidimensional Gaussian filter

blurred\_img2=ndimage.median\_filter(img\_noise,5) # Calculate a multidimensional median filter

plt.figure(1)

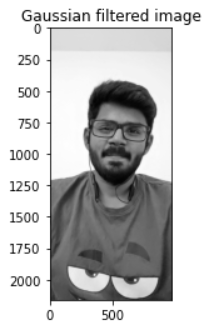
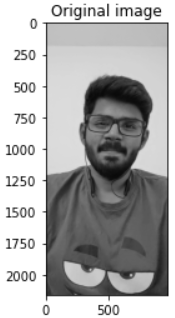
plt.imshow(img,cmap='gray')

plt.title("Original image")

plt.figure(2)

plt.imshow(blurred\_img1,cmap='gray')

plt.title("Gaussian filtered image")



1. Create an image containing your registration number (similar to how we created images with names) and include some shapes in it (e.g. lines, rectangles or circles). Apply
2. Sobel and Prewitt operators
3. Line detection
4. Circle detection
5. Boundary extraction

**CODE:**

1. **Sobel and Prewitt operators**

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

from scipy import ndimage # Multi-dimensional image processing

img=mpimg.imread('C:/Users/WELCOME/Downloads/capture.jpg')

def rgb2gray(rgb):

r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]

gray = 0.2989 \* r + 0.5870 \* g + 0.1140 \* b

return gray

img=rgb2gray(img)

sharpened\_img1=ndimage.prewitt(img,axis=0) # Prewitt - vertical

sharpened\_img2=ndimage.prewitt(img, axis=1) # Prewitt - horizontal

sharpened\_img3=ndimage.sobel(img,axis=0) # Sobel - vertical

sharpened\_img4=ndimage.sobel(img,axis=1) # Sobel - horizontal

plt.figure(1)

plt.imshow(img,cmap='gray')

plt.title("Original image")

plt.figure(2)

plt.imshow(sharpened\_img1,cmap='gray')

plt.title("Prewitt - vertical")

plt.figure(3)

plt.imshow(sharpened\_img2,cmap='gray')

plt.title("Prewitt - horizontal")

plt.figure(4)

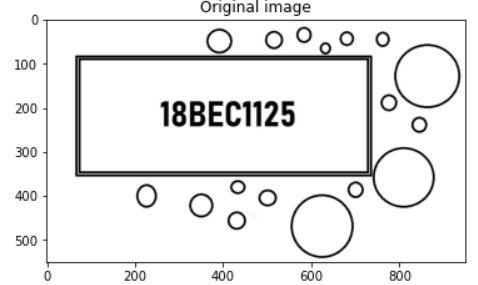
plt.imshow(sharpened\_img3,cmap='gray')

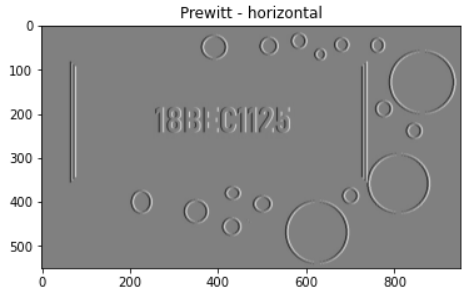
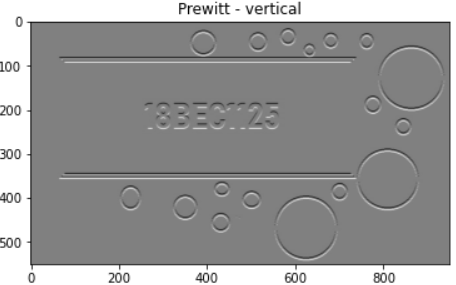
plt.title("Sobel - vertical")

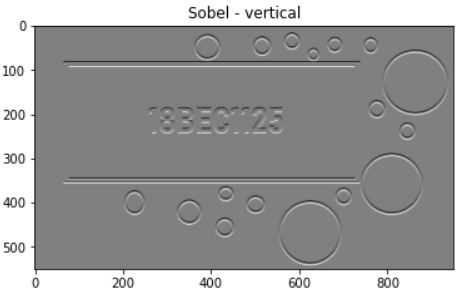
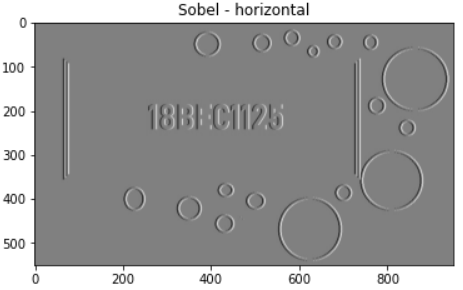
plt.figure(5)

plt.imshow(sharpened\_img4,cmap='gray')

plt.title("Sobel - horizontal")



1. **Line Detection**

#HOUGH LINE DETECTION

#HOUGH TRANSFORM – LINE DETECTION

import cv2

import numpy as np

import matplotlib.pyplot as plt

a = cv2.imread('C:/Users/WELCOME/Downloads/capture.jpg')

d=2

a\_gauss=cv2.GaussianBlur(a, (2\*d+1, 2\*d+1), -1)[d:-d,d:-d]

img= cv2.cvtColor(a\_gauss, cv2.COLOR\_BGR2GRAY)

edges = cv2.Canny(img,50,150,apertureSize = 3)

plt.figure(1)

plt.imshow(cv2.cvtColor(edges, cv2.COLOR\_BGR2RGB))

lines = cv2.HoughLines(edges,1,np.pi/180,100)

for i in range(4):

for rho,theta in lines[i]:

a = np.cos(theta)

b = np.sin(theta)

x0 = a\*rho

y0 = b\*rho

x1 = int(x0 + 1000\*(-b))

y1 = int(y0 + 1000\*(a))

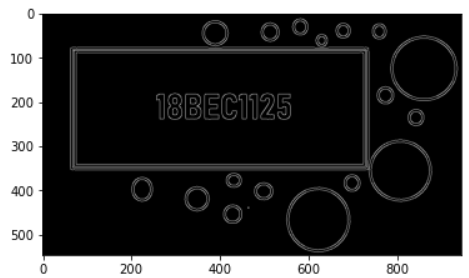
x2 = int(x0 - 1000\*(-b))

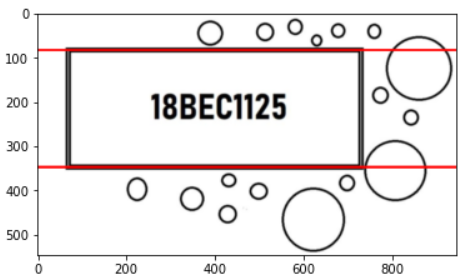
y2 = int(y0 - 1000\*(a))

cv2.line(a\_gauss,(x1,y1),(x2,y2),(0,0,255),2)

plt.figure(2)

plt.imshow(cv2.cvtColor(a\_gauss, cv2.COLOR\_BGR2RGB))





1. **Circle detection**

import cv2

import numpy as np

import matplotlib.pyplot as plt

img = cv2.imread('C:/Users/WELCOME/Downloads/capture1.jpg')

plt.imshow(img,cmap='gray')

plt.figure()

d=2

gimg=cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

gimg = cv2.GaussianBlur(gimg, (2\*d+1, 2\*d+1), -1)[d:-d,d:-d]

#cimg = cv2.cvtColor(img,cv2.COLOR\_GRAY2BGR)

circles = cv2.HoughCircles(gimg,cv2.HOUGH\_GRADIENT,1,170,param1=50,param2=30,minRadius=0,maxRadius=0)

circles = np.uint16(np.around(circles))

for i in circles[0,:]:

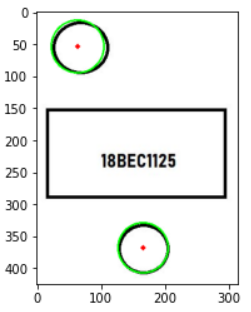
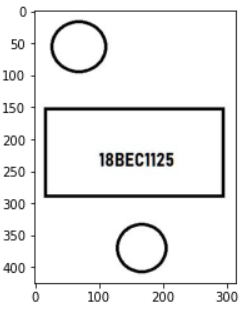
# draw the outer circle

cv2.circle(img,(i[0],i[1]),i[2],(0,255,0),2)

# draw the center of the circle

cv2.circle(img,(i[0],i[1]),2,(0,0,255),3)

plt.imshow(cv2.cvtColor(img, cv2.COLOR\_BGR2RGB))



1. **Boundary Extraction**

import cv2

import numpy as np

import matplotlib.pyplot as plt

img = cv2.imread('C:/Users/WELCOME/Downloads/capture.jpg',0)

plt.figure()

plt.title('Original Image')

plt.imshow(img,cmap='gray')

kernel = np.ones((5,5),np.uint8)

img\_eroded = cv2.erode(img,kernel,iterations =1)

img\_dilated = cv2.dilate(img,kernel,iterations = 1)

img\_opened = cv2.morphologyEx(img, cv2.MORPH\_OPEN, kernel)

img\_closed = cv2.morphologyEx(img, cv2.MORPH\_CLOSE, kernel)

plt.figure()

plt.imshow(img\_eroded,cmap='gray')

plt.title('Erosion')

plt.figure()

plt.imshow(img\_dilated,cmap='gray')

plt.title('Dilation')

plt.figure()

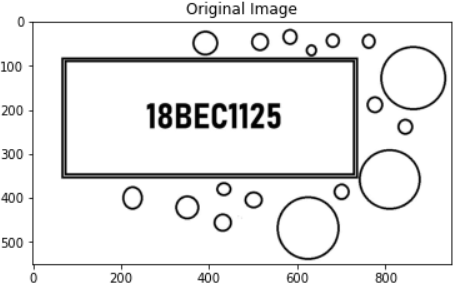
plt.imshow(img\_opened,cmap='gray')

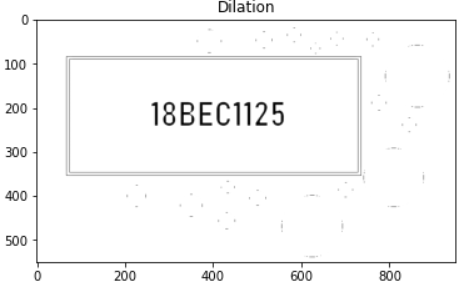
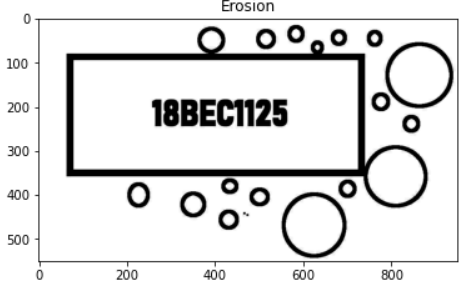
plt.title('Opening')

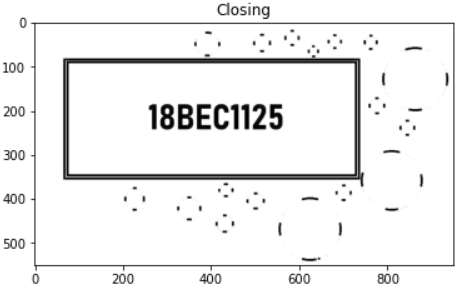
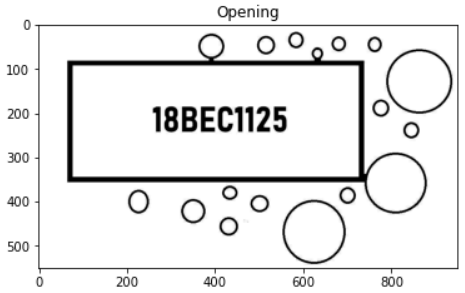
plt.figure()

plt.imshow(img\_closed,cmap='gray')

plt.title('Closing')







img\_boundary1 = np.subtract(img,img\_eroded)

plt.figure()

plt.imshow(img\_boundary1,cmap='gray')

plt.title('img\_boundary1')

img\_boundary2 = np.subtract(img\_dilated,img)

plt.figure()

plt.imshow(img\_boundary2,cmap='gray')

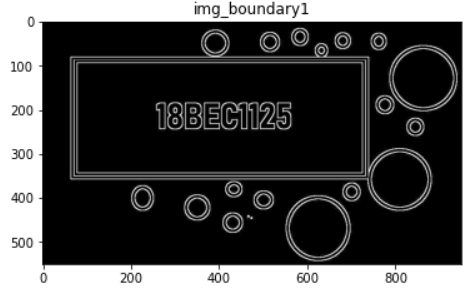
plt.title('img\_boundary2')

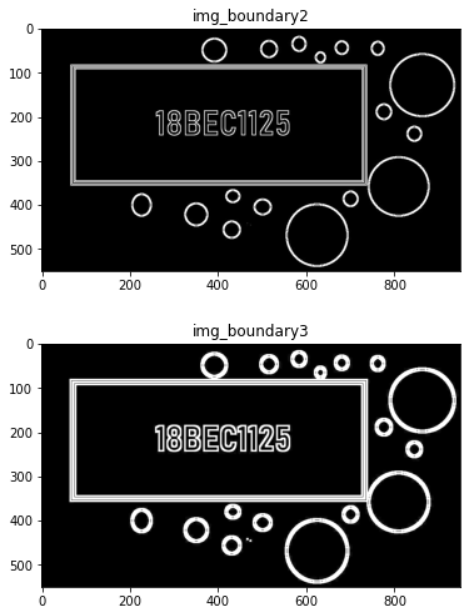
img\_boundary3= np.subtract(img\_dilated,img\_eroded)

plt.figure()

plt.imshow(img\_boundary3,cmap='gray')

plt.title('img\_boundary3')





1. Create an image containing 4 different shapes each of different colors (red, green, blue, pink, etc.) and apply color-based clustering on it.

**CODE:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

img = cv2.imread('C:/Users/WELCOME/Downloads/capture2.jpg')

hsv = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

plt.figure()

plt.title('Original Image')

plt.imshow(hsv)

lower\_blue = np.array([0,20,50])

upper\_blue = np.array([150,170,255])

lower\_green = np.array([20, 100, 50])

upper\_green = np.array([130, 240, 150])

lower\_red = np.array([150, 0, 20])

upper\_red = np.array([255, 130, 120])

lower\_pink = np.array([150, 150, 150])

upper\_pink = np.array([255, 200, 210])

# Threshold with inRange() get only specific colors

mask\_blue = cv2.inRange(hsv, lower\_blue, upper\_blue)

mask\_green = cv2.inRange(hsv, lower\_green, upper\_green)

mask\_red = cv2.inRange(hsv, lower\_red, upper\_red)

mask\_pink = cv2.inRange(hsv, lower\_pink, upper\_pink)

# Perform bitwise operation with the masks and original image

res\_blue = cv2.bitwise\_and(img,img, mask= mask\_blue)

res\_green = cv2.bitwise\_and(img,img, mask= mask\_green)

res\_red = cv2.bitwise\_and(img,img, mask= mask\_red)

res\_pink = cv2.bitwise\_and(img,img, mask= mask\_pink)

#hsv\_red[res\_red != 0] = hsv

res\_red=cv2.cvtColor(res\_red, cv2.COLOR\_BGR2RGB)

res\_blue=cv2.cvtColor(res\_blue, cv2.COLOR\_BGR2RGB)

res\_pink=cv2.cvtColor(res\_pink, cv2.COLOR\_BGR2RGB)

#plt.imshow(cv2.cvtColor(hsv\_red, cv2.COLOR\_HSV2RGB))

plt.figure()

plt.imshow(res\_blue)

plt.title('BLUE')

plt.figure()

plt.imshow(res\_red)

plt.title('RED')

plt.figure()

plt.imshow(res\_green)

plt.title('GREEN')

plt.figure()

plt.imshow(res\_pink)

plt.title('PINK')

