Applied Computer Vision

Final Year Practical File(IoT-010)

Submitted to:-

DEI

Under the guidance:-

Miss. Sadhana Singh

Submitted by:

Dayal Nigam

(1803743)

DEPARTMENT OF PHYSICS & COMPUTER

SCIENCE FACULTY OF SCIENCE

DAYALBAGH EDUCATIONAL INSTITUTE

DAYALBAGH AGRA (UP)-282005

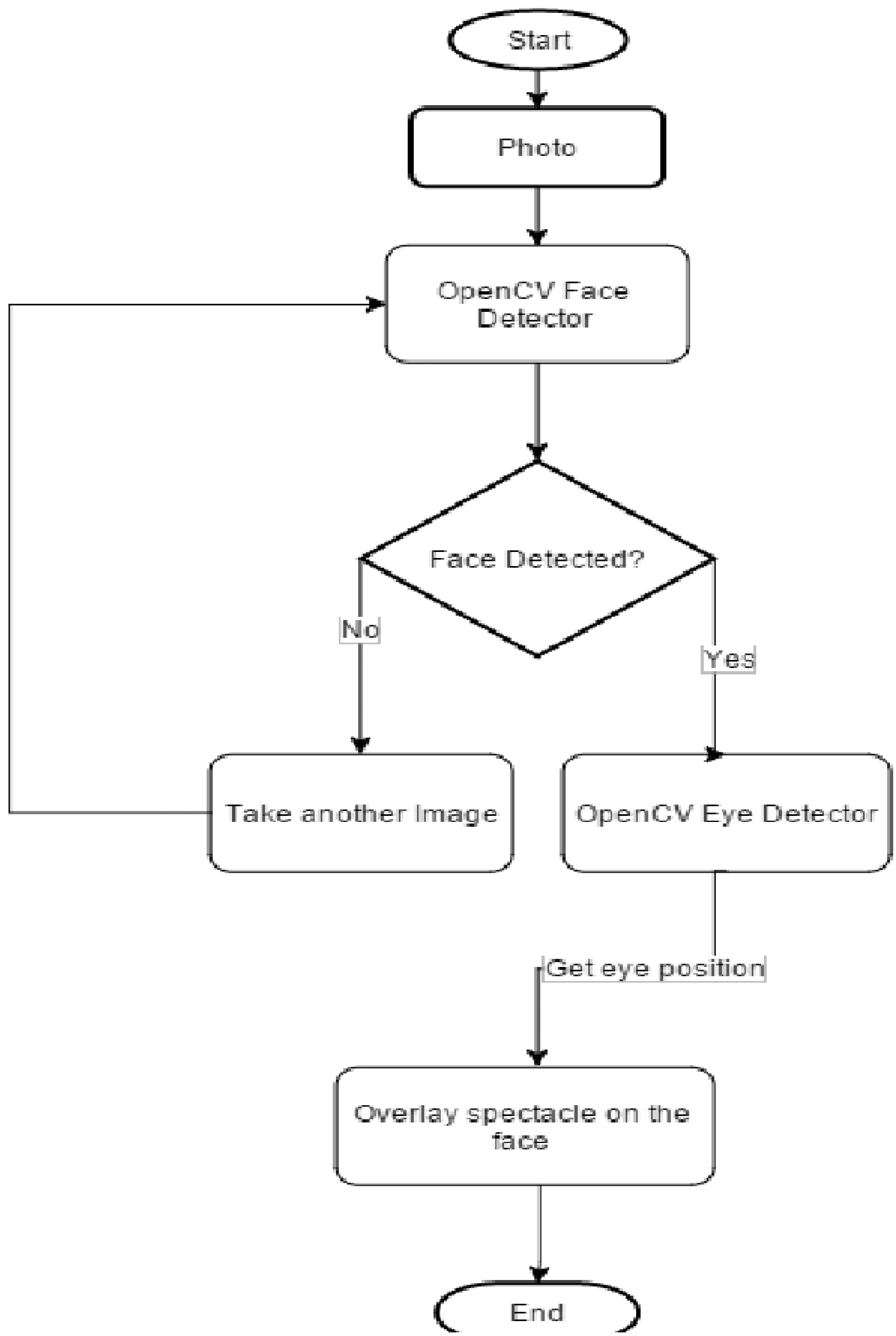
Table of Contents

Practical No: 1

Question :- Explain the computer vision in details with its significance, need and applications.

Theory Explanation :- The objective OpenCV Computer Vision is to made computer smart enough such that it will smartly recognize the objects nearby just like humans which was very complex type of vision available out there due to this vision humans can easily recognize the objects the living beings out there such computation involves complex neural computation at neuron level also these things involve high level computation which occurs inside human brain at neuron level within seconds to understand this computation we need high level understanding of human mind + neurons + computer science .

How does it works?
Computer vision is concerned with the automatic extraction, analysis and understanding of useful information from a single image or video. It basically includes theoretical and algorithmic basis to achieve automatic understanding. It is an interdisciplinary field that works for high level understanding from images or videos.


Start
Photo
OpenCV Face Detector
Face Detected?
No
Yes
Take another Image
OpenCV Eye Detector
Get eye position
Overlay spectacle on the face
End

Practical No: 2

Question :- Write a Program to create a Logo of your by drawing shapes and text on an image.

Theory Explanation :- The functions which are mainly used to change the image.

- **Resize:** img =cv2.resize(img_rgb,(1300,275))
- **Resize by Ratio:** new_img=cv2.resize(img_rgb,(0,0),img,w_ratio,h_ ratio)
- **Flip Image:** Flipped_img = cv2.flip(img, flag)
- **Rectangle:** cv2.rectangle(blank_img,pt1=(384,0),pt2=(510,128),color=(0,255,0),thickness=5)
- **Lines:** cv2.line(blank_img,pt1=(0,0),pt2=(511,511),color=(102, 255, 255),thickness=5)
- **Circles :** cv2.circle(img=blank_img,center=(100,100),radius=(255,0,0),thickness=5)
- **Text:** cv2.putText(blank_img,text='Hello',org=(10,500),fontFace=font.fontScale=4,color=(255,255,255),thickness=2,lineType=cv2.LINE_AA)

Code

```
import cv2
import numpy as np
img = cv2.imread('black.jpg',0)

##img = cv2.line(img,(270,180),(255,255),(0, 136, 254),5)

image = cv2.putText(img, 'DAYAL', (360,450), cv2.FONT_HERSHEY_SIMPLEX,
                   5, (255,136,136), 2, cv2.LINE_AA)
image = cv2.circle(img,(590,400),250,(255,136,136),2)
image = cv2.circle(img,(590,400),270,(255,136,136),2)
image = cv2.putText(img, '---*---', (420,270), cv2.FONT_HERSHEY_SIMPLEX,
                   2, (255,136,136), 2, cv2.LINE_AA)
image = cv2.putText(img, '---*---', (420,550), cv2.FONT_HERSHEY_SIMPLEX,
                   2, (255,136,136), 2, cv2.LINE_AA)



cv2.imshow('image',img)
cv2.imwrite('logo.png',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output

Practical No: 3

Question :- Apply different color spaces on image.

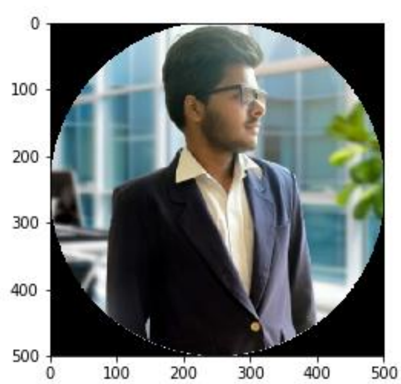
Theory Explanation :- The functions which are mainly used to change the color of the image.

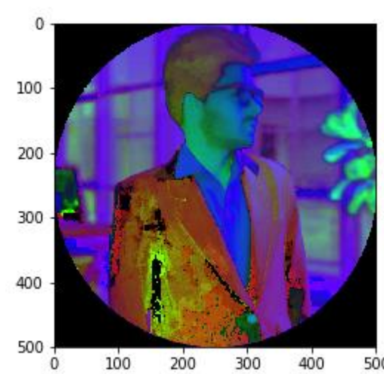
- **RGB color:** cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
- **HSV color:** cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
- Most popular colorspace is RGB , colors are modeled are as a combination of RED , GREEN ,BLUE.
- Some alternative models were developed in later
  HSL
  HSV
- More closely aligned the way human vision actually
  perceives color.

Code

```python
import cv2
import matplotlib.pyplot as plt
import numpy as np

img = cv2.imread('dayal.png')

img1 = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.imshow(img1)
plt.savefig('dayal1.png')

img2 = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)

plt.imshow(img2)
plt.savefig('dayal2.png')

img2 = cv2.cvtColor(img,cv2.COLOR_BGR2HLS)

plt.imshow(img2)
plt.savefig('dayal3.png')
```

Output





Practical No: 4

Question :- Implement image blending and pasting to create a new image.

Theory Explanation :- The functions which are mainly used to blend the image.

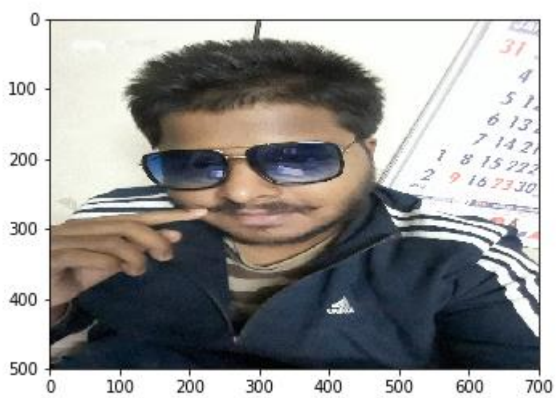
- **blended** = cv2.addWeighted(src1=img1,alpha=0.7,src2=img2,beta=0.3,gamma=0)
- Blending and pasting is used commonly to mix two images or overlay an image on top of another image.
- Image Blending means to mix or blend two images.
- Image pasting means to copy one image over other image.

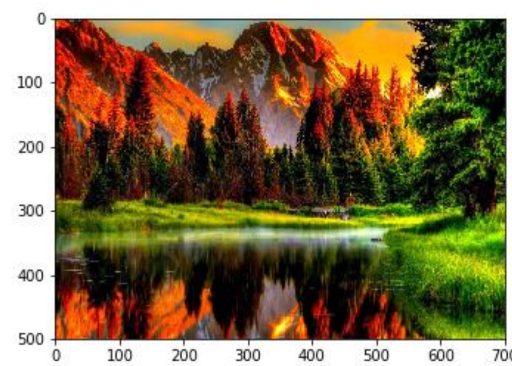
Code

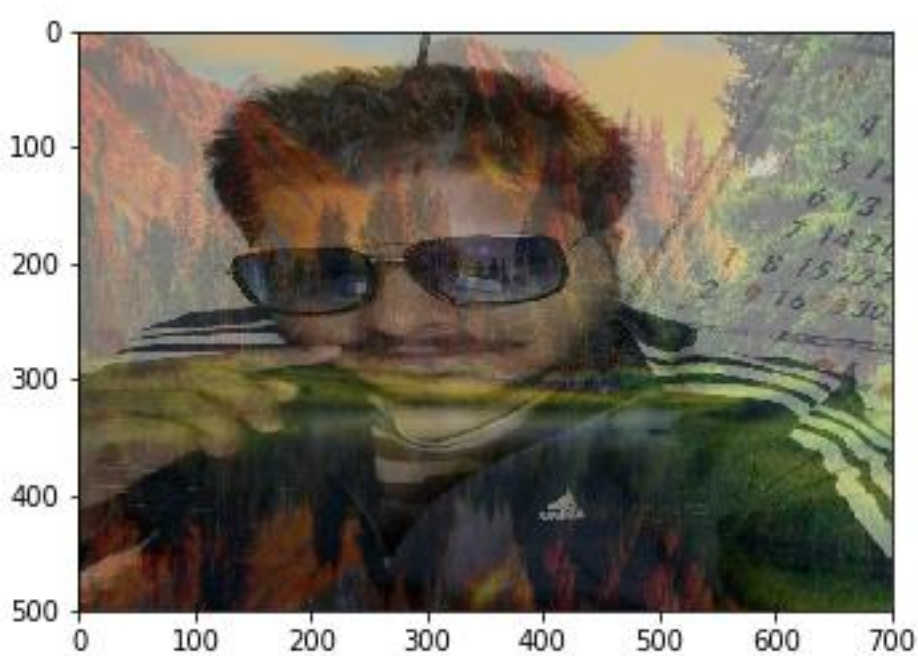
```python
import cv2
import matplotlib.pyplot as plt


# In[14]:


img1=cv2.imread('C:/Users/Dayal Nigam/Desktop/dayal
personal/.ipynb_checkpoints/dayal2.jpg')
img2=cv2.imread('C:/Users/Dayal Nigam/Desktop/dayal
personal/.ipynb_checkpoints/myfile.jpg')


# In[15]:


img1.shape
img2.shape


# In[16]:


img2 = cv2.resize(img2,(700,500))
img1 = cv2.resize(img1,(700,500))


# In[17]:


img1=cv2.cvtColor(img1,cv2.COLOR_BGR2RGB)
img2=cv2.cvtColor(img2,cv2.COLOR_BGR2RGB)
plt.imshow(img1)
plt.savefig('myfig.png')


# In[18]:


plt.imshow(img2)
plt.savefig('myfig-1.png')


# In[19]:


img2.shape


# In[20]:


```

```
b = cv2.addWeighted(src1=img1,alpha=0.5,src2=img2,beta=0.3,gamma=0.4)


# In[21]:


plt.imshow(b)
plt.savefig('myfig-2.png')
```

Output







Practical No: 5

Question :- Implement the different Image blur and smoothing techniques on an image

Theory Explanation :- The functions which are mainly used to blur and smoothing the image.

- Gaussian Blur – cv2.GaussianBlur(image,kernel_size = (5,5), sigma_value)

- Median Blur – cv2.medianBlur(image, Kernel_size)

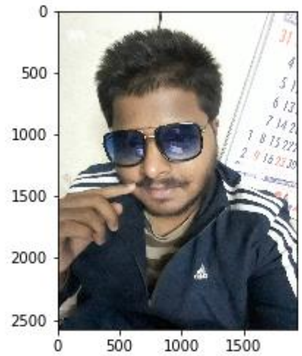
- Smoothing an image means to make the color transition from one side of an edge in the image to another smooth rather than sudden.
- To average out rapid changes in pixel intensity.
- The blur, or smoothing, of an image removes “outlier” pixels that may be noise in the image.
- It is an example of applying low pass filter.

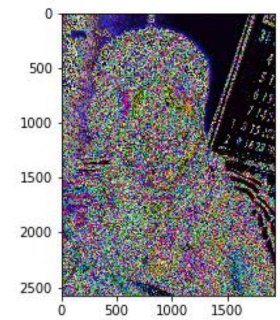
Code

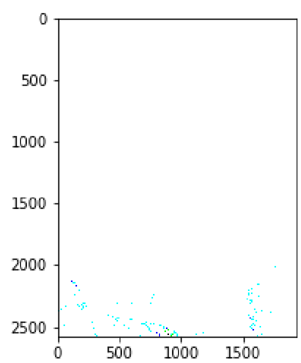
```
#!/usr/bin/env python
# coding: utf-8

# In[2]:


import cv2
import matplotlib.pyplot as plt
import numpy as np


# In[7]:


img1=cv2.imread('C:/Users/Dayal Nigam/Desktop/dayal personal/.ipynb_checkpoints/dayal2.jpg')
img1=cv2.cvtColor(img1,cv2.COLOR_BGR2RGB)
plt.imshow(img1)
plt.savefig('b1.png')


# In[9]:


gamma=2
effect_img=np.power(img1,gamma)
plt.imshow(effect_img)
plt.savefig('effect_img.png')


# In[17]:


gamma=0.8
effect_img=np.power(img1,gamma)
plt.imshow(effect_img)
plt.savefig('effect_img-1.png')


# In[16]:


gamma=1.6
effect_img1=np.power(img1,gamma)
plt.imshow(effect_img1)
plt.savefig('effect_img-1.png')


# In[57]:


img1=cv2.imread('C:/Users/Dayal Nigam/Desktop/dayal personal/.ipynb_checkpoints/dayal.png')
img1=cv2.cvtColor(img1,cv2.COLOR_BGR2RGB)
```

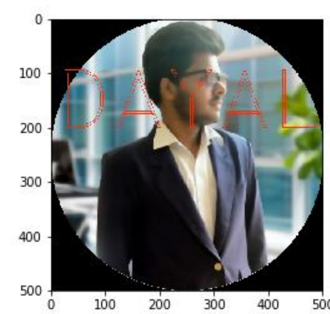
```python
font=cv2.FONT_HERSHEY_DUPLEX
img1=cv2.putText(img1,text='DAYAL',org=(5,200),fontFace=font,fontScale=5,color=(255, 45, 0))
plt.imshow(img1)
plt.savefig('text.png')


# In[58]:


blur=cv2.blur(img1,(5,5))
plt.imshow(blur)
plt.savefig('blur.png')


# In[56]:


blur=cv2.blur(img1,(7,7))
plt.imshow(blur)


# In[62]:


blur=cv2.GaussianBlur(img1,(3,3),0)
plt.imshow(blur)
plt.savefig('Gaussian-blur.png')


# In[64]:


blur2=cv2.medianBlur(img1,5)
plt.imshow(blur2)
plt.savefig('Gaussian-blur-1.png')


# In[65]:


blurr=cv2.bilateralFilter(img1,9,75,75)
plt.imshow(blurr)
plt.savefig('myblurr.png')


# In[ ]:
```

Output



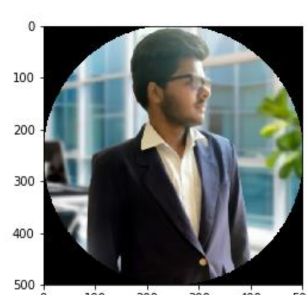






BilateralFilter

GaussianFilter



MedianFilter

Practical No: 6

Question :- Implement Erosion, Dilation, Opening and Closing morphological operators on an image

Theory Explanation :- The functions which are mainly used to Erosion ,Dilation ,Opening and closing the image.

- **erosion** = cv2.erode(img,kernel,iterations = 1)
- **dilation** = cv2.dilate(img,kernel,iterations = 1)
- **opening** = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
- **closing** = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
- **Dilation**: dilation makes an object more visible. It is an operation which adds maximum value of all pixels in neighbourhood

- **Erosion**: the resultant value of a pixel is the minimum value of all pixels in neighborhood. – Removes island and small objects.

- **Opening**: erode followed by dilation using the same structuring element.

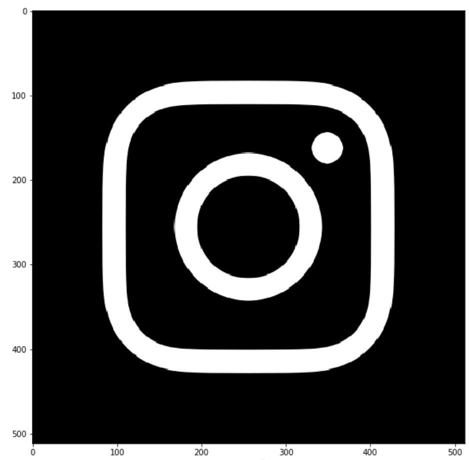
- **Closing**: dilation followed by erosion using the same structuring element.

Code

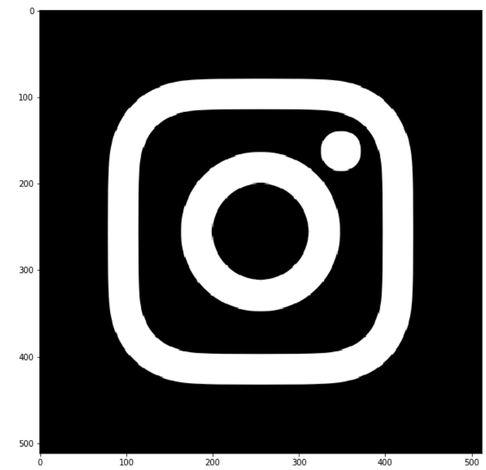
```python
#!/usr/bin/env python
# coding: utf-8

# In[2]:


import cv2
import numpy as np
import matplotlib.pyplot as plt


# # DAYAL NIGAM

# # Reading an Image

# In[3]:


img = cv2.imread("C:/Users/Dayal Nigam/Desktop/dayal
personal/.ipynb_checkpoints/insta.png")
plt.figure(figsize=(10,10))
plt.imshow(img)
plt.savefig('insta.png')


# # Erosion using Opencv

# In[5]:


kernel = np.ones((5,5),np.uint8)
erosion = cv2.erode(img,kernel,iterations = 1)
plt.figure(figsize=(10,10))
plt.imshow(erosion)
plt.savefig('insta-1.png')


# # dilation using opencv

# In[6]:


dilation = cv2.dilate(img,kernel,iterations = 1)
plt.figure(figsize=(10,10))
plt.imshow(dilation)
plt.savefig('insta-2.png')


# In[7]:


inversion=erosion-dilation
plt.imshow(inversion)
```

```python
plt.savefig('inver.png')


# In[ ]:





# # MorphologyEx MORPH_OPEN

# In[8]:


opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
plt.figure(figsize=(10,10))
plt.imshow(opening)
plt.savefig('insta-3.png')


# # MorphologyEx MORPH_CLOSE

# In[9]:


closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
plt.figure(figsize=(10,10))
plt.imshow(closing)
plt.savefig('instaa.png')


# # Gradient Descent using opencv

# In[10]:


gradient = cv2.morphologyEx(img, cv2.MORPH_GRADIENT, kernel)
plt.figure(figsize=(10,10))
plt.imshow(gradient)
plt.savefig('gradient.png')


# In[ ]:





# In[ ]:
```

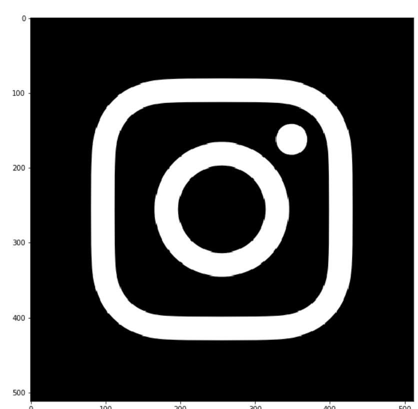
Output



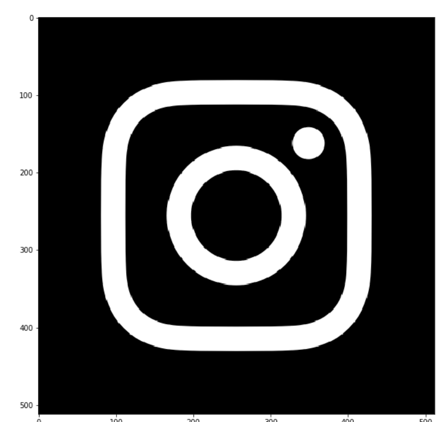
Erosion



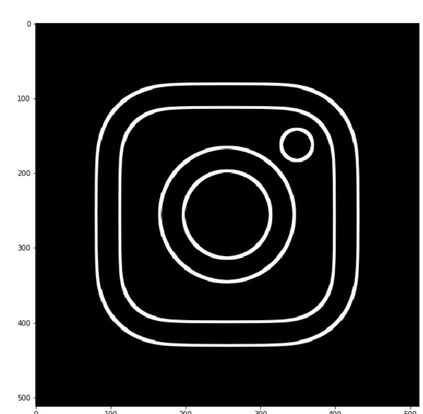
Dilation



OPENING



CLOSING



GRADIENT_DESCENT

Practical No: 7

Question :- Implement Canny Edge detection on an image to extract the edge.

Theory Explanation :- The functions which are mainly used to Canny Edge detection

- **sobelx** = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=7)
- **sobely** = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=7)
- **laplacian** = cv2.Laplacian(img,cv2.CV_64F)
- **blended** = cv2.addWeighted(src1=sobelx,alpha=0.5,src2=sobely,beta=0.5,gamma=0)
- **gradient** = cv2.morphologyEx(blended,cv2.MORPH_GRADIENT,kernel)
- **edges** = cv2.Canny(image=img, threshold1=127, threshold2=127)

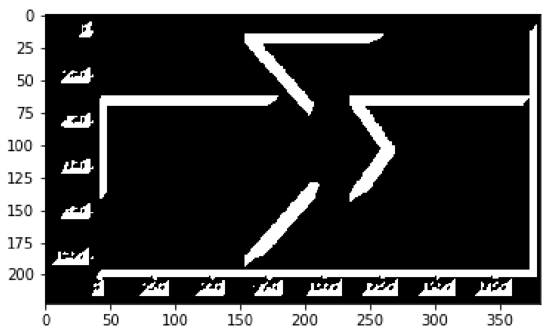
Code

```python
#!/usr/bin/env python
# coding: utf-8

# # Canny_Edge_Detection

# In[2]:


import numpy as np
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
import cv2


# In[6]:


img = cv2.imread("C:/Users/Dayal Nigam/Desktop/.ipynb_checkpoints/canny.png")
img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.imshow(img)
plt.xlabel('canny-img',fontsize=20)
plt.savefig('canny.png')


# In[8]:


sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=7)
sobely = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=7)
laplacian = cv2.Laplacian(img,cv2.CV_64F)


# In[9]:


plt.imshow(sobelx)
plt.xlabel('sobelx-img',fontsize=20)
plt.savefig('sobelx.png')


# In[10]:


plt.imshow(sobely)
plt.xlabel('sobely-img',fontsize=20)
plt.savefig('sobely.png')


# In[11]:


plt.imshow(laplacian)
```

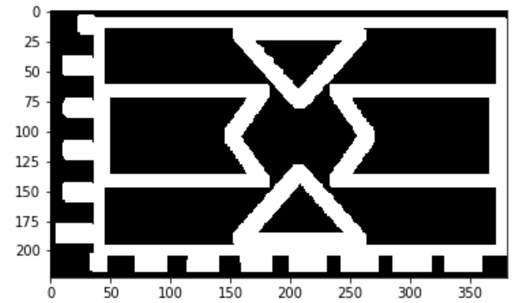
```python
plt.xlabel('laplacian-img',fontsize=20)
plt.savefig('laplacian.png')


# # Blending Image

# In[12]:


blended= cv2.addWeighted(src1=sobelx,alpha=0.5,src2=sobely,beta=0.5,gamma=0)
plt.imshow(blended)


# In[13]:


kernel = np.ones((4,4),np.uint8)
gradient = cv2.morphologyEx(blended,cv2.MORPH_GRADIENT,kernel)
plt.imshow(gradient)
plt.xlabel('gradient-img',fontsize=20)
plt.savefig('gradient.png')


# In[14]:


ret,th1 = cv2.threshold(gradient,200,255,cv2.THRESH_BINARY_INV)
plt.imshow(th1)
plt.xlabel('threshold-img',fontsize=20)
plt.savefig('threshold.png')


# In[ ]:





# In[15]:


edges = cv2.Canny(image=img, threshold1=127, threshold2=127)
plt.imshow(edges)
plt.xlabel('edges-img at th1=127 & th2=127',fontsize=20)
plt.savefig('edges.png')


# In[16]:


edges = cv2.Canny(image=img, threshold1=0, threshold2=255)
plt.imshow(edges)
plt.xlabel('edges-img at th1=0 & th2=255',fontsize=20)
plt.savefig('edges.png')


# In[14]:
```

```python


# Calculate the median pixel value
med_val = np.median(img)

# Lower bound is either 0 or 70% of the median value, whicever is higher
lower = int(max(0, 0.7* med_val))

# Upper bound is either 255 or 30% above the median value, whichever is lower
upper = int(min(255,1.3 * med_val))

edges = cv2.Canny(image=img, threshold1=lower , threshold2=upper)
plt.imshow(edges)


# In[15]:


blurred_img = cv2.blur(img,ksize=(5,5))
edges = cv2.Canny(image=blurred_img, threshold1=lower , threshold2=upper)
plt.imshow(edges)


# In[16]:


edges = cv2.Canny(image=blurred_img, threshold1=lower , threshold2=upper+50)

plt.imshow(edges)


# In[ ]:





# In[17]:


edges = cv2.Canny(image=img, threshold1=127, threshold2=127)
plt.imshow(edges)


# In[ ]:





# In[ ]:
```

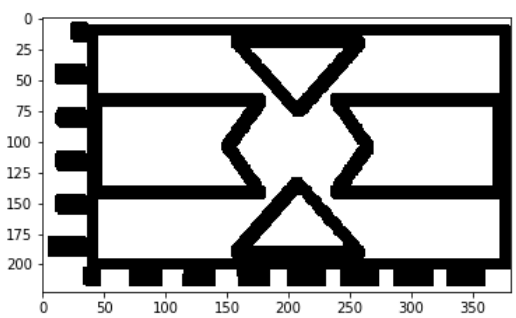
Output



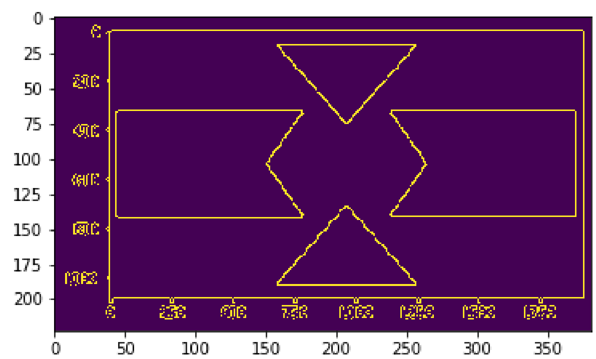
blended-img



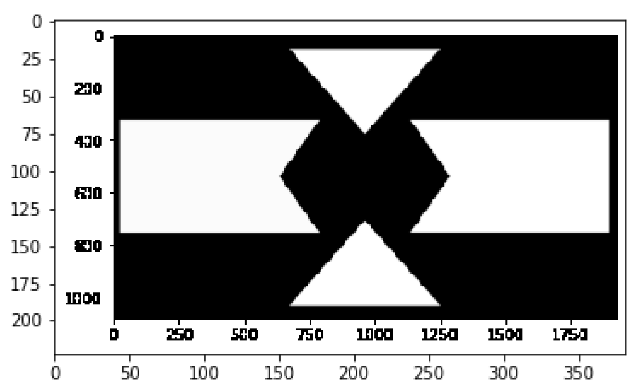
gradient-img



threshold-img



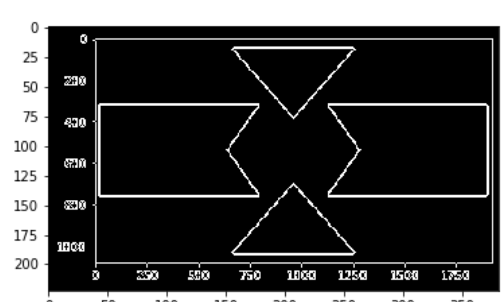
edges-img at th1=0 & th2=255



canny-img

sobelx-img

sobely-img



laplacian-img

Practical No: 8

Question :- Implement the template matching with OpenCV

Theory Explanation :- The functions which are mainly used to template matching.

- **Template** = cv2.matchTemplate(grey_img,template,cv2.TM_CCOEFF_NORMED)
  - Template matching is the simplest form of object detection.
  - It is a method of searching and finding the location of template image in a larger image.
  - It simply scans a larger image for a provided template by sliding the template target image across the larger image.

- cv2.TM_CCOEFF
- cv2.TM_CCOEFF_NORMED
- cv2.TM_CCORR
- cv2.TM_CCORR_NORMED
- cv2.TM_SQDIFF
- cv2.TM_SQDIFF_NORMED

Code

```python
#!/usr/bin/env python
# coding: utf-8

# In[24]:


import cv2
import numpy as np
import matplotlib.pyplot as plt


# # Dayal Nigam B.Voc(IoT) 1803743

# In[19]:


img = cv2.imread('C:/Users/Dayal Nigam/Desktop/dayal
personal/.ipynb_checkpoints/image.jpeg')
grey_img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
template = cv2.imread('C:/Users/Dayal Nigam/Desktop/dayal
personal/.ipynb_checkpoints/dayal face.jpeg',0)
plt.imshow(img)


# In[20]:


w,h = template.shape[::-1]
res = cv2.matchTemplate(grey_img,template,cv2.TM_CCOEFF_NORMED)
print(res)
threshold=0.9
loc=np.where(res>=threshold)
print(loc)


# In[21]:


for pt in zip(*loc[::-1]):
    cv2.rectangle(img,pt,(pt[0]+w,pt[1]+h),(0,0,255),2)


# In[23]:


plt.imshow(img)


# In[ ]:
```

```


# In[ ]:






# In[ ]:





# In[ ]:
```

Output

Practical No: 9

Question :- Implement corner detection and grid detection on an image.

Theory Explanation :- The functions which are mainly used to Corner detection

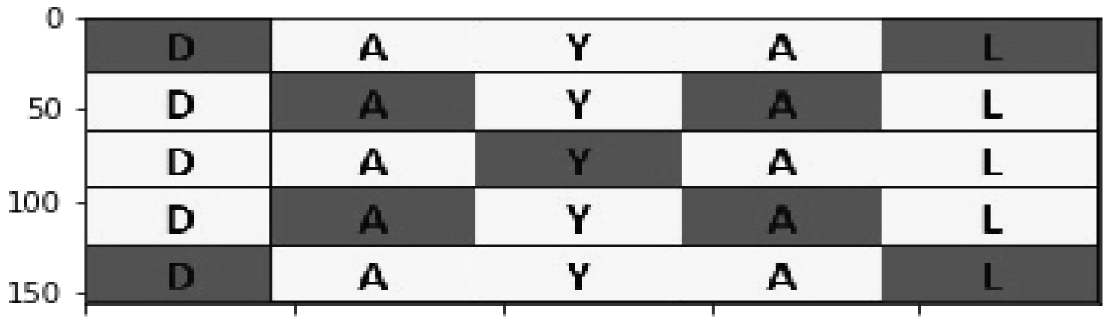
- A corner is a point whose local neighborhood stands in two dominant and different edge directions.
- A corner simply can be interpreted as the junction of two edges or an edge is a sudden change.
- corners are regions in the image with large variation in intensity in all the directions.

- **dst** = cv2.cornerHarris(src=gray,blockSize=2,ksize=3, k=0.04)
- **dst** = cv2.dilate(dst,None)

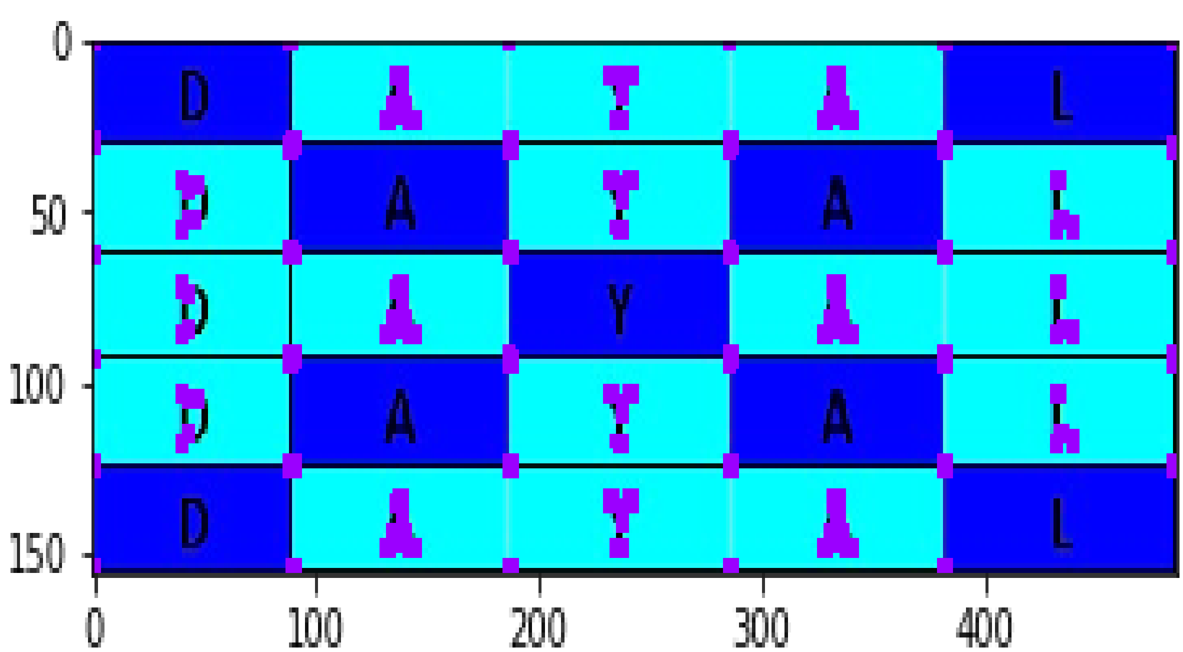
Code

```python
#!/usr/bin/env python
# coding: utf-8

# In[57]:


import cv2
import numpy as np
import matplotlib.pyplot as plt


# # Dayal Nigam

# # 1803743

# In[52]:


flat_chess1 = cv2.imread('C:/Users/Dayal Nigam/Desktop/dayal
personal/.ipynb_checkpoints/mat.jpeg')

flat_chess = cv2.cvtColor(flat_chess1,cv2.COLOR_BGR2GRAY)
plt.imshow(flat_chess,cmap='gray')


# In[53]:


gray = np.float32(flat_chess)


# In[54]:


dst = cv2.cornerHarris(src=gray,blockSize=2,ksize=3,k=0.04)


# In[56]:


dst = cv2.dilate(dst,None)
flat_chess1[dst>0.01*dst.max()]=[155,0,255]
plt.imshow(flat_chess1)


# In[ ]:






# In[ ]:
```

Output



flat_chess



edge detection i.e corner detection

Practical No: 10

Question :- Implement Feature matching with ORB and SIFT descriptors.

Theory Explanation :- The functions which are mainly used to ORB and SIFT descriptors.

- Template matching requires exact copy of an image.
- Feature matching extracts defining key features from an image such as corners, edges, and contour.
- Using a distance measure it finds all the matches in another image.
- So no need of exact template
- kp1, des1 = orb.detectAndCompute(im1,None)
- kp2, des2 = orb.detectAndCompute(im1,None)

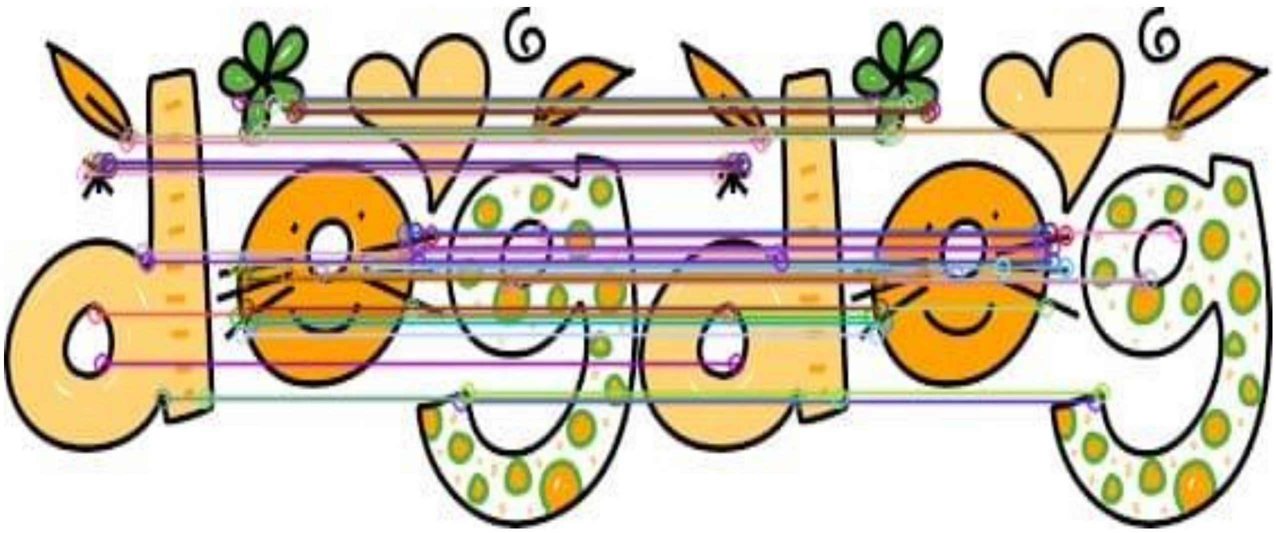
- bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

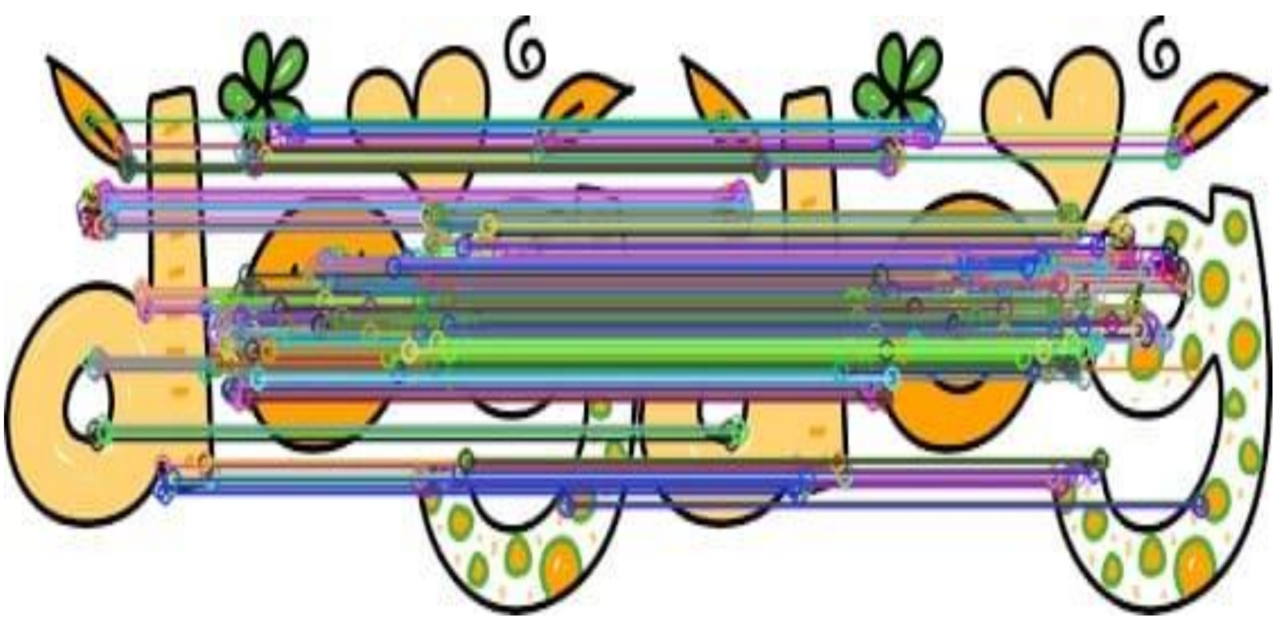
Code

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

def display(img,cmap='gray'):
        fig = plt.figure(figsize=(12,10))
        ax = fig.add_subplot(111)
        ax.imshow(img,cmap='gray')

im1 = cv2.imread('dogg.jpg')
im2 = cv2.imread('dogg.jpg')

# Initiate ORB detector
orb = cv2.ORB_create()
# find the keypoints and descriptors with ORB

kp1, des1 = orb.detectAndCompute(im1,None)
kp2, des2 = orb.detectAndCompute(im1,None)

# create BFMatcher object
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
# Match descriptors.
matches = bf.match(des1,des2)

# Sort them in the order of their distance.
matches = sorted(matches, key = lambda x:x.distance)
# Draw first 100 matches.
im_matches = cv2.drawMatches(im1,kp1,im2,kp2,matches[:50],None,flags=2)

# BFMatcher with default params
bf = cv2.BFMatcher()
matches = bf.knnMatch(des1,des2, k=2)
# finding good matches
good = []
for match1,match2 in matches:
        if match1.distance < 0.75*match2.distance:
                good.append([match1])

# cv2.drawMatchesKnn expects list of lists as matches.
sift_matches = cv2.drawMatchesKnn(im1,kp1,im2,kp2,good,None,flags=2)


cv2.imshow('detected circles',im_matches)
File3 = 'feature_ORB.png'
cv2.imwrite(File3, im_matches)
cv2.waitKey(0)
cv2.destroyAllWindows()

cv2.imshow('detected circles',sift_matches)
File3 = 'feature_SHIFT.png'
```

```
cv2.imwrite(File3, sift_matches)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output





Practical No: 11

Question :- Implement Watershed algorithm for image segmentation

Theory Explanation :- The functions which are mainly used to Watershed algorithm for image segmentation.

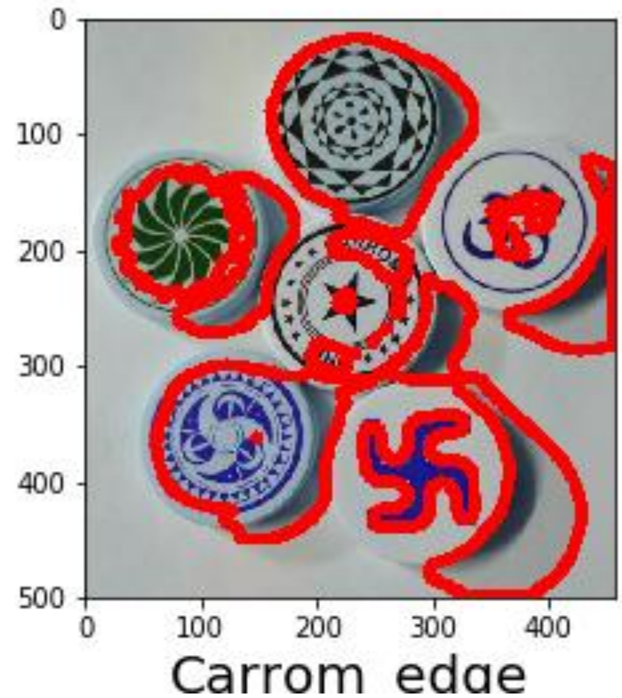
- **img** = cv2.medianBlur(carrom_img,25)
- **ret,thresh** = cv2.threshold(gray,160,255,cv2.THRESH_BINARY_INV)
- **contours,hierarchy**=cv2.findContours(thresh.copy(),cv2.RETR_CCOMP,cv2.CHAIN_APPROX_NONE)
- cv2.watershed(image, markers)
- **ret, markers** = cv2.connectedComponents(sure_fg)
- Convert image to Grayscale.
- Threshold the image.
- Remove the noise present in image with the help of morphological operators.
- Identify background and sure foreground.
  • Identify unknown regions.
- Label Markers of Sure Foreground
- Apply Watershed Algorithm to find Markers
- Find contours on markers.

Code

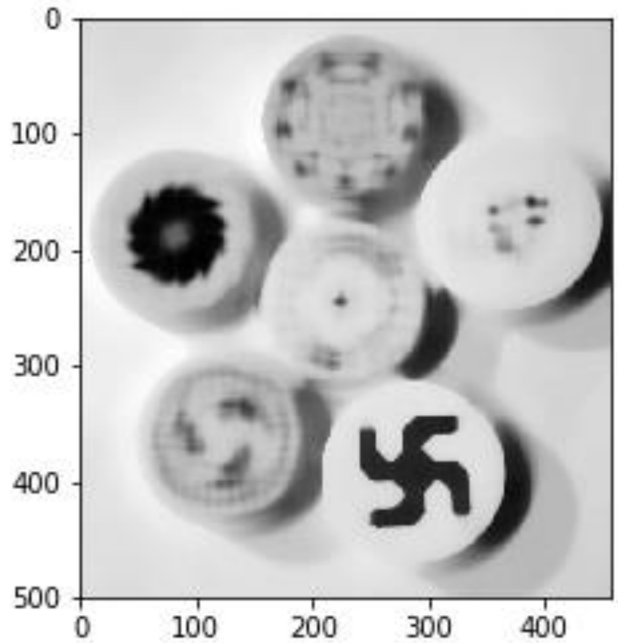
```
#!/usr/bin/env python
# coding: utf-8

# In[17]:


import numpy as np
import cv2
import matplotlib.pyplot as plt


# # Dayal Nigam 1803743

# In[3]:


carrom_img = cv2.imread('C:/Users/Dayal Nigam/Desktop/dayal 
personal/.ipynb_checkpoints/myimg.jpeg')


# In[6]:


plt.imshow(carrom_img)


# In[7]:


img = cv2.medianBlur(carrom_img,25)
plt.imshow(img)


# In[10]:


gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
plt.imshow(gray,cmap='gray')


# In[11]:


ret,thresh = cv2.threshold(gray,160,255,cv2.THRESH_BINARY_INV)


# In[12]:


plt.imshow(thresh)
```

```python
# In[13]:


contours,hierarchy=cv2.findContours(thresh.copy(),cv2.RETR_CCOMP,cv2.CHAIN
_APPROX_NONE)


# In[14]:


for i in range(len(contours)):
    if hierarchy[0][i][3]==-1:
        cv2.drawContours(carrom_img,contours,i,(255,0,0),10)


# In[15]:


plt.imshow(carrom_img)


# In[ ]:
```

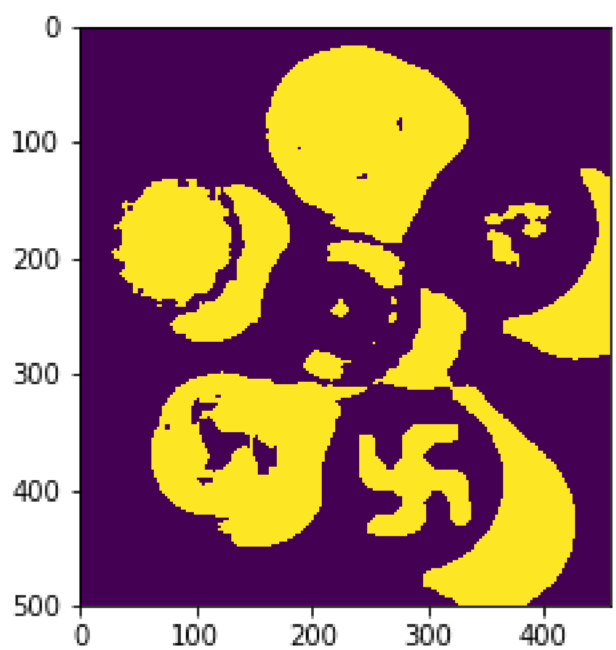
Output


0
100
200
300
400
500
0
100
200
300
400
Carrom edge


0
100
200
300
400
500
0
100
200
300
400
carrom img



gray img



Threshold img