EX NO: 1	**OPENCV INSTALLATION - PYTHON**
DATE :	

AIM:

To install a opencv in python platform and working with python.

PROCEDURE:

Step1: Open the Command line(search for cmd in the Run dialog(+ R). Now run the following **command:**

```
python --version
```

If Python is already installed, it will generate a message with the Python version available.

Step2: Next, check if PIP is already installed on your system, just go to the command line and executethe following command:

```
pip -V
```

Step3: OpenCV can be directly downloaded and installed with the use of pip (package manager). To install OpenCV, just go to the command-line and type the following command:

```
pip install opencv-python
```

Step4: Type the command in the Terminal and proceed

Step5: pip install opencv-python

Step6: Collecting Information and downloading data:

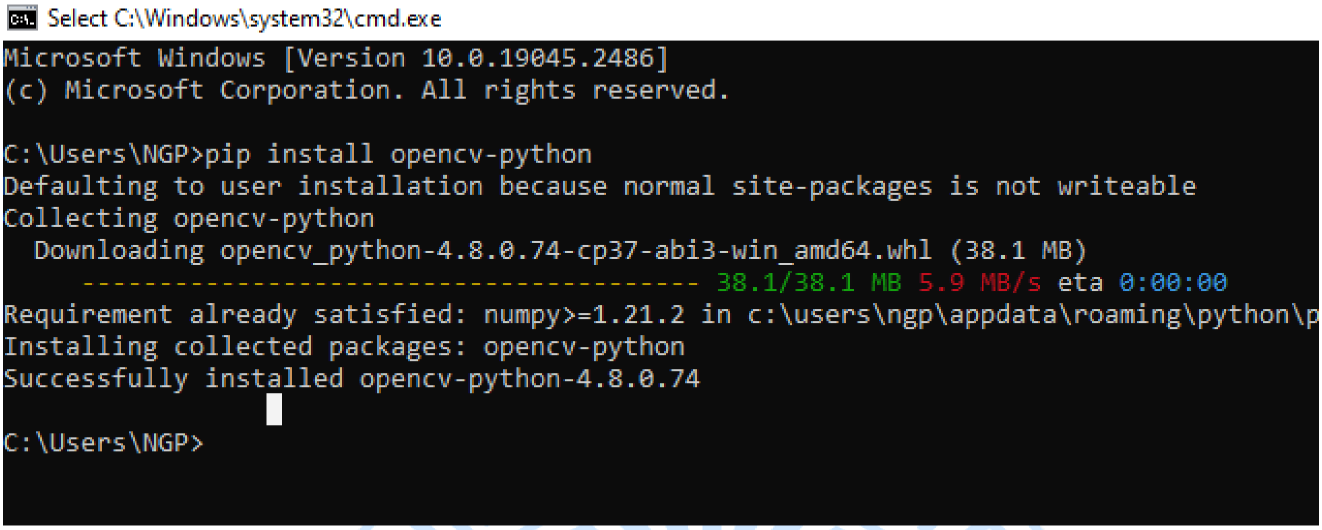
Step7: Installing Packages:

Step8: Finished Installation:

Step9: To check if OpenCV is correctly installed, just run the following commands to perform a version check:

```
Python
 import cv2
print(cv2._version_)
```

OUTPUT:



```
Microsoft Windows [Version 10.0.19045.2486]
(c) Microsoft Corporation. All rights reserved.

C:\Users\NGP>pip install opencv-python
Defaulting to user installation because normal site-packages is not writeable
Collecting opencv-python
  Downloading opencv_python-4.8.0.74-cp37-abi3-win_amd64.whl (38.1 MB)
     ---------------------------------------- 38.1/38.1 MB 5.9 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.21.2 in c:\users\ngp\appdata\roaming\python\p
Installing collected packages: opencv-python
Successfully installed opencv-python-4.8.0.74

C:\Users\NGP>
```

RESULT:

Thus, the installation of opencv in python platform and working with python was executed and output is verified successfully.

EX NO: 2	BASIC IMAGE PROCESSING ,LOADING IMAGE ,CROPPING ,RESIZING, THRESHOLDING CONTOUR ANALYSIS ,BLOB DETECTION
DATE	

AIM:

To write a python code to perform basic image processing such as loading image , cropping resizing , thresholding contour analysis ,blob detection.

ALGORITHM:

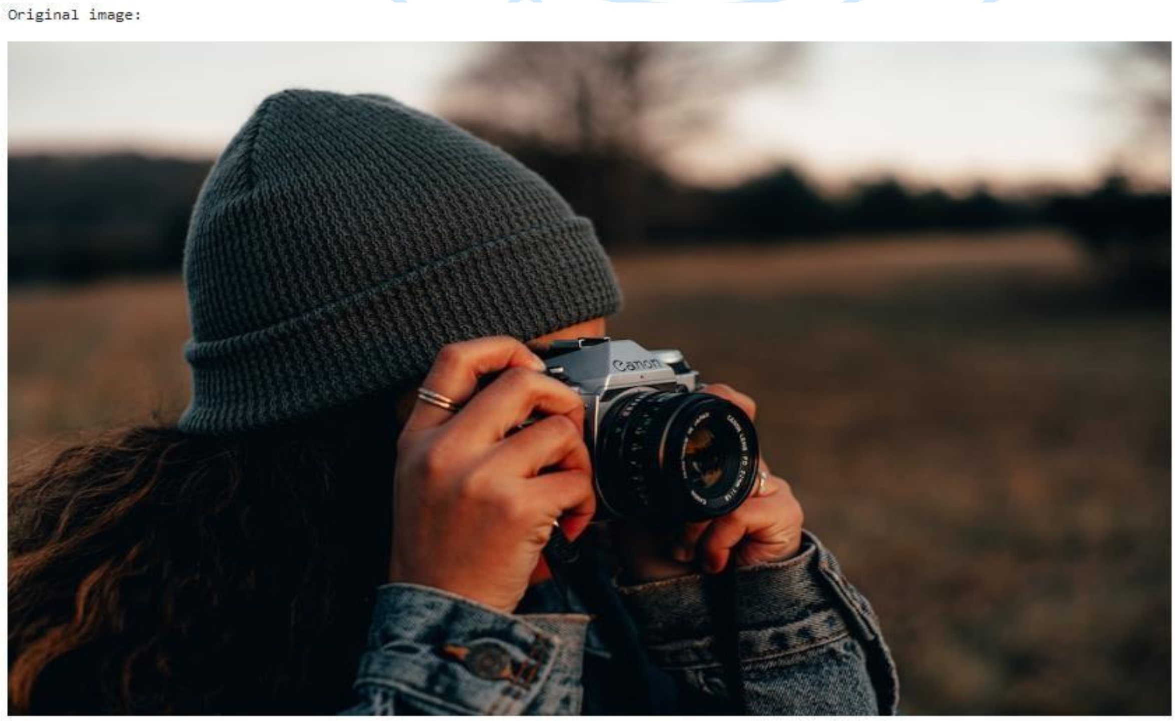
Step1: Import Necessary packages.
Step2:Load the image using cv2.imread()
Step3:Using plt.imshow() print the original image
Step4:By defining width and height crop the image and display the cropped image.
Step5:Using resize() print ‘Half’,’Bigger’,’Interpolation Nearest’ views to original image.and resized images.
Step6:Print the Threshold analysis of the image
Step4:Print the contour analysis view of the image.
Step5:Load an image of different shapes for Blob detection.
Step6:Using SimpleBlobDetector_create() find the number of circles in the image and print the count.
Step7: display all the images.
Step7: stop the program.

PROGRAM:

LOADING IMAGES:

```
from PIL import Image
print("Original image:\n")
img = Image.open('/content/cvimage.jpg')
img.show()
```

OUTPUT:



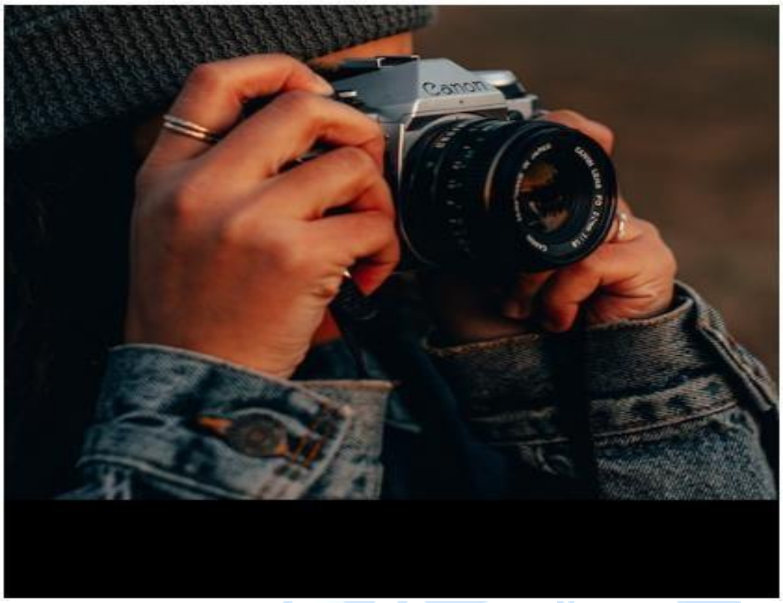
```
Original image:
```

CROPPING:

```
box = (250, 250, 750, 750)
img2 = img.crop(box)
img2.save('myimage_cropped.jpg')
print("\n\ncropped image is:\n")
img2.show()
```

OUTPUT:

```
cropped image is:
```

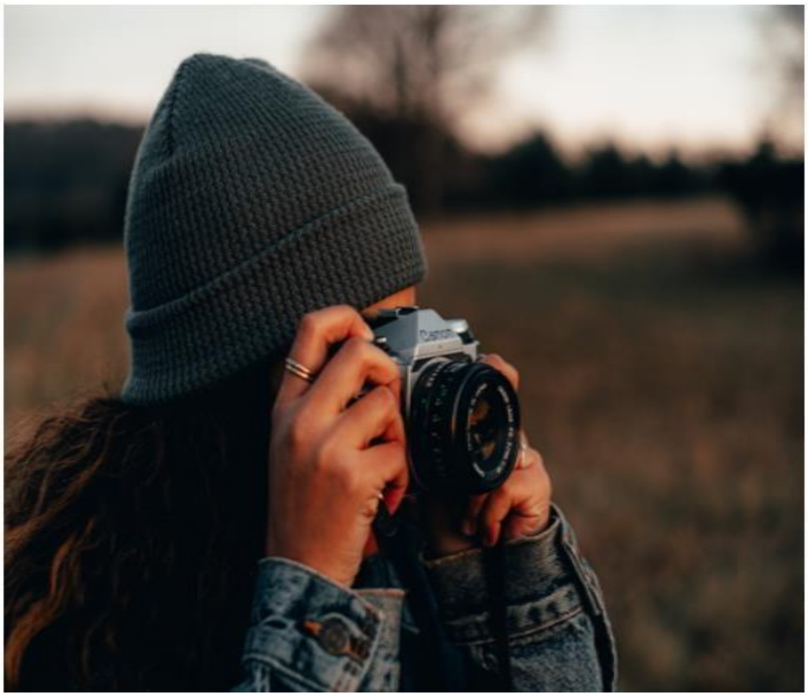


RESIZING:

```
print("\n\nResized image:\n")
img3=img.resize((400,500))
img3.save('resizedimg.jpg')
img3.show()
```

OUTPUT:

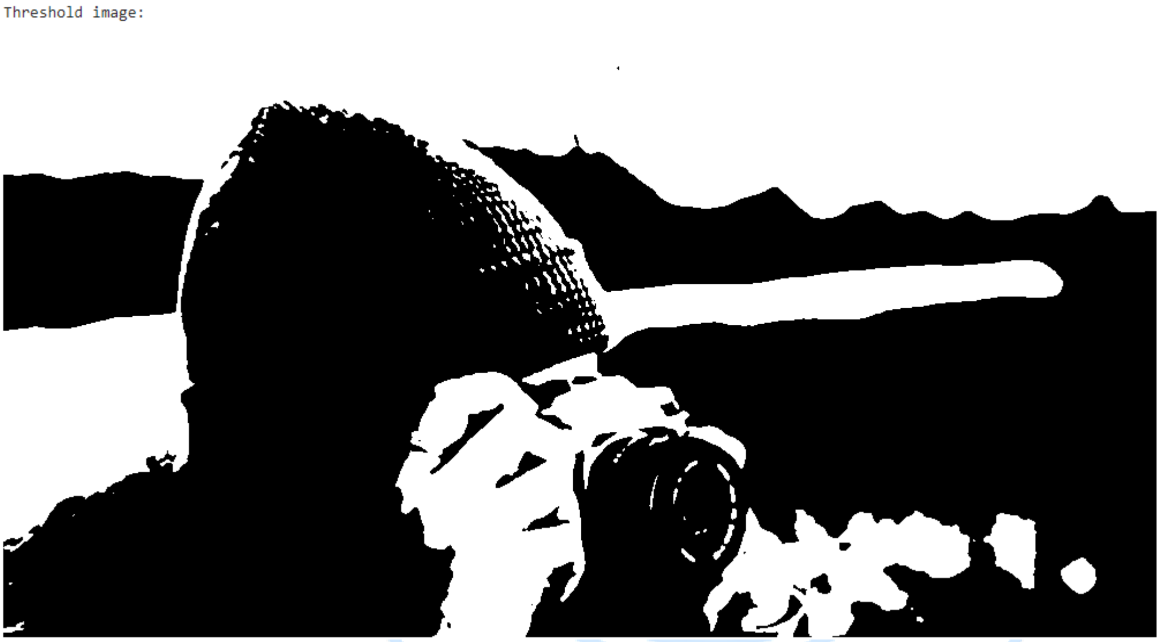
```
Resized image:
```



THRESHOLDING:

```
import cv2
from google.colab.patches import cv2_imshow
image = cv2.imread("cvimage.jpg")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
blurred = cv2.GaussianBlur(gray, (7, 7), 0)
ret, thresh_binary = cv2.threshold(blurred, 60, 255, cv2.THRESH_BINARY)
cv2_imshow(image)
print("\n\nThreshold image:\n")
cv2_imshow(thresh_binary)
```

OUTPUT:



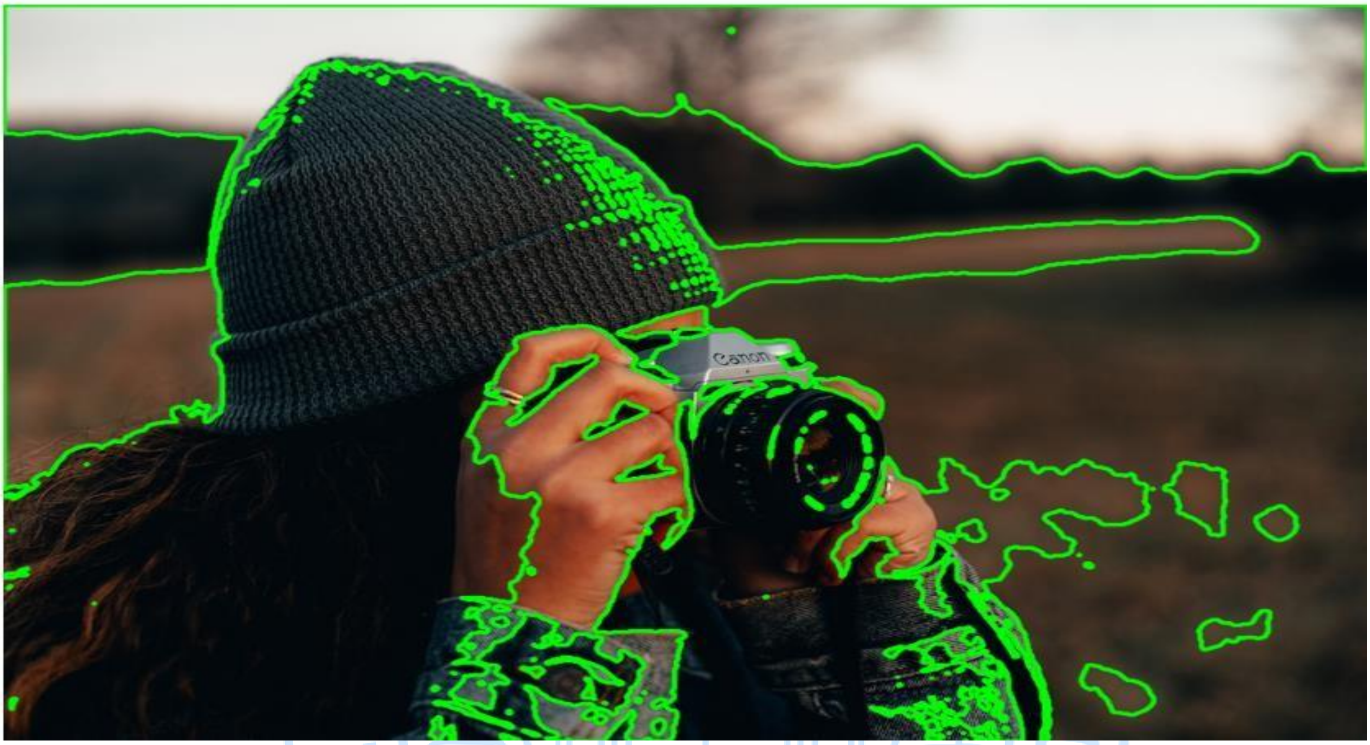
```
Threshold image:
```

CONTOUR ANALYSIS:

```
print("\n\nContour analysis:\n")
contours,hierarchy=cv2.findContours(image=thresh_binary,mode=
cv2.RETR_TREE,method=cv2.CHAIN_APPROX_NONE)
contour_image=image.copy()
cv2.drawContours(image=contour_image,contours=contours,conto
urIdx=-1,color=(0,250,0),thickness=2,lineType=cv2.LINE_AA)
cv2_imshow(contour_image)
```

OUTPUT:

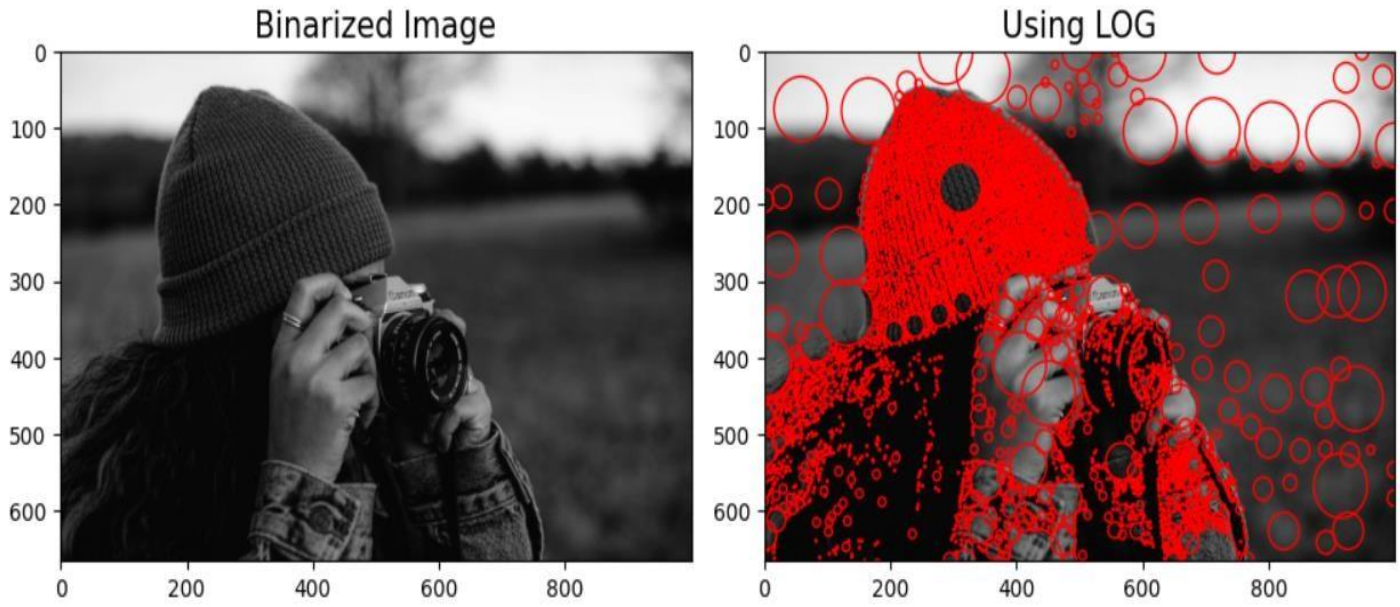
```
Contour analysis:
```



BLOB DECTECTION:

```
import cv2
from skimage.io import imshow, imread
from skimage.color import rgb2gray
import matplotlib.pyplot as plt
import numpy as np
from skimage.feature import blob_dog,
blob_log, blob_doh
sample = imread('/content/cvimage.png')
sample_g = rgb2gray(sample)
fig, ax = plt.subplots(1,2,figsize=(10,5))
ax[0].set_title('Binarized Image',fontsize=15)
ax[0].imshow(sample_g,cmap='gray')
blobs = blob_log(sample_g, max_sigma=30,
threshold=0.01)
ax[1].imshow(sample_g, cmap='gray')
for blob in blobs:
    y, x, area = blob
    ax[1].add_patch(plt.Circle((x, y),
area*np.sqrt(2), color='r',fill=False))
ax[1].set_title('Using LOG',fontsize=15)
plt.tight_layout()
plt.show()
```

OUTPUT:



RESULT:

Thus the python program to perform various operations on image using open-CV was verified and executed successfully.

EX NO: 3 DATE:	Image Annotation – Drawing lines, text circles, rectangle, ellipseon image

AIM:

To write a python code to perform a image annotation in drawing lines, text circle, rectangle,ellipse on image.

ALGORITHM:

Step1: Start the program.

Step2: Load the image using in read() function

Step3: Using the line in the image by line (image line, point A, point B)

Step4: Draw the circle in the image

Step5: Draw the rectangle in the image using rectangle function

Step6: Draw the eclipse in the image

Step7: Stop the program

PROGRAM:

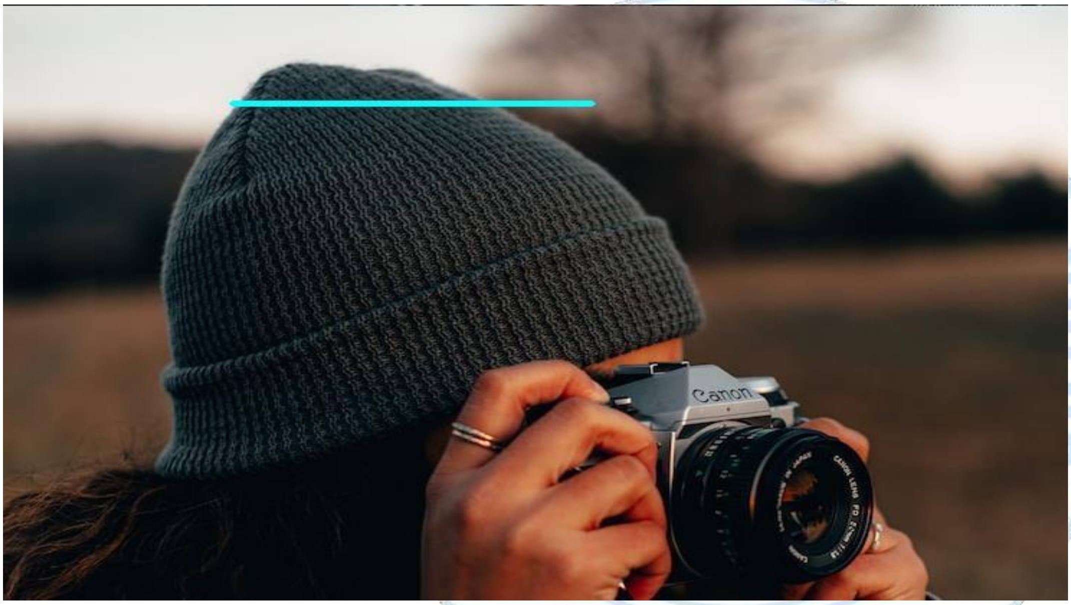
```
from google.colab.patches import cv2_imshow
import cv2
img= cv2.imread("/content/cvimage.png")
cv2_imshow(img)
cv2.waitKey()
cv2.destroyAllWindows()
```

Original Image:

Line Image:

```
imageLine = img.copy()
pointA = (200,80)
pointB = (450,80)
cv2.line(imageLine, pointA, pointB, (255, 255, 0), thickness=3)
cv2_imshow(imageLine)
cv2.waitKey(0)
```

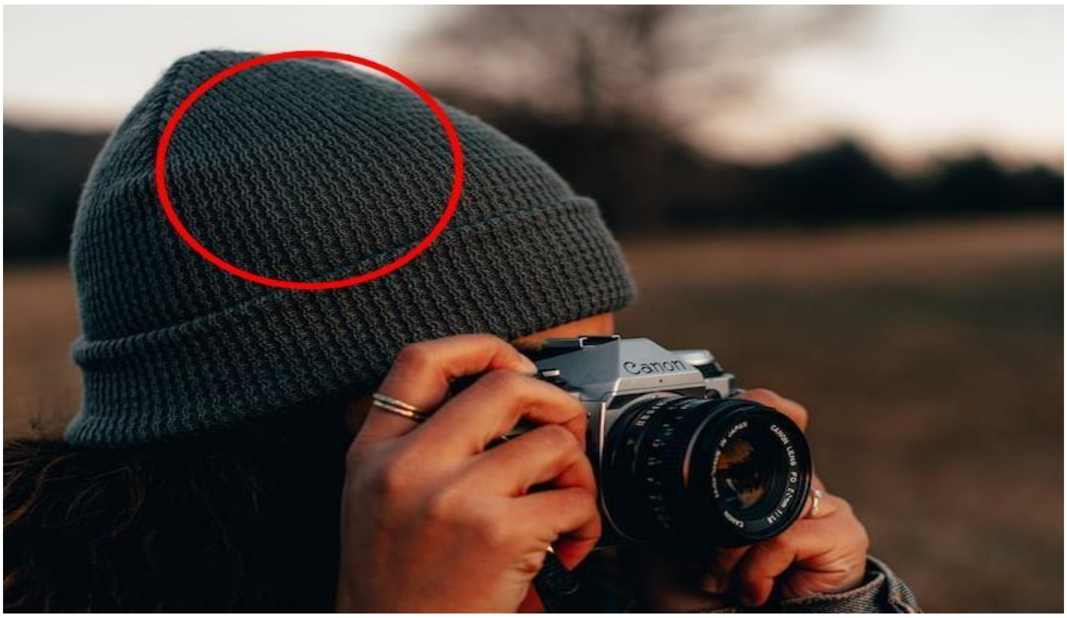
OUTPUT:



Circle Image:

```
imageCircle = img.copy()
circle_center = (310,150)
radius =100
cv2.circle(imageCircle, circle_center, radius, (0, 0, 255), thickness=3, lineType=cv2.LINE_AA)
cv2_imshow(imageCircle)
cv2.waitKey(0)
```

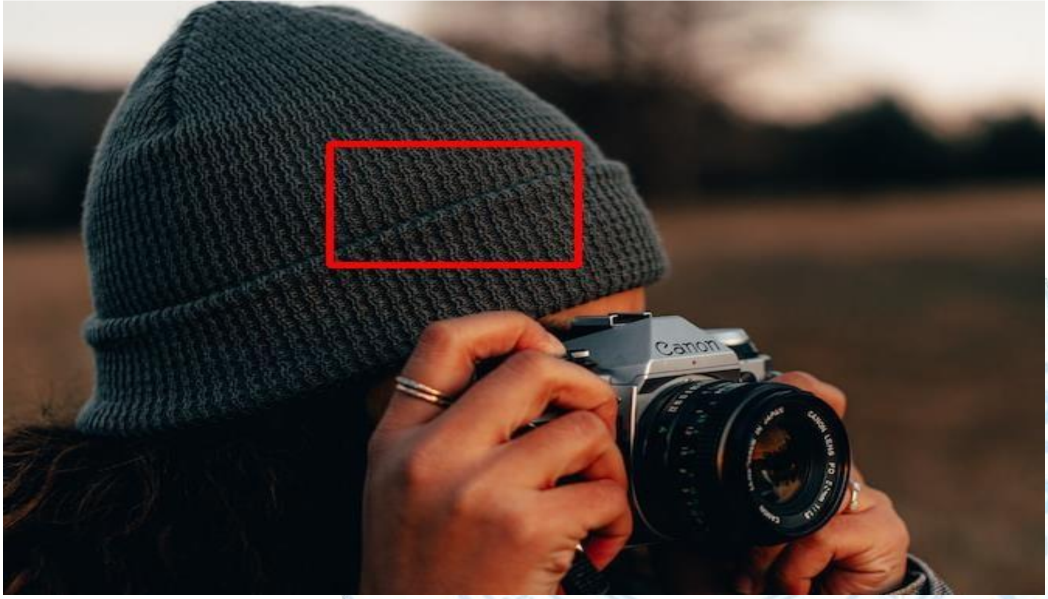
Output:



Rectangle image:

```
imageRectangle = img.copy()
start_point =(310,160)
end_point =(470,255)
cv2.rectangle(imageRectangle, start_point, end_point, (0, 0, 255), thickness= 3, lineType=cv2.LINE_8)
cv2_imshow(imageRectangle)
cv2.waitKey(0)
```

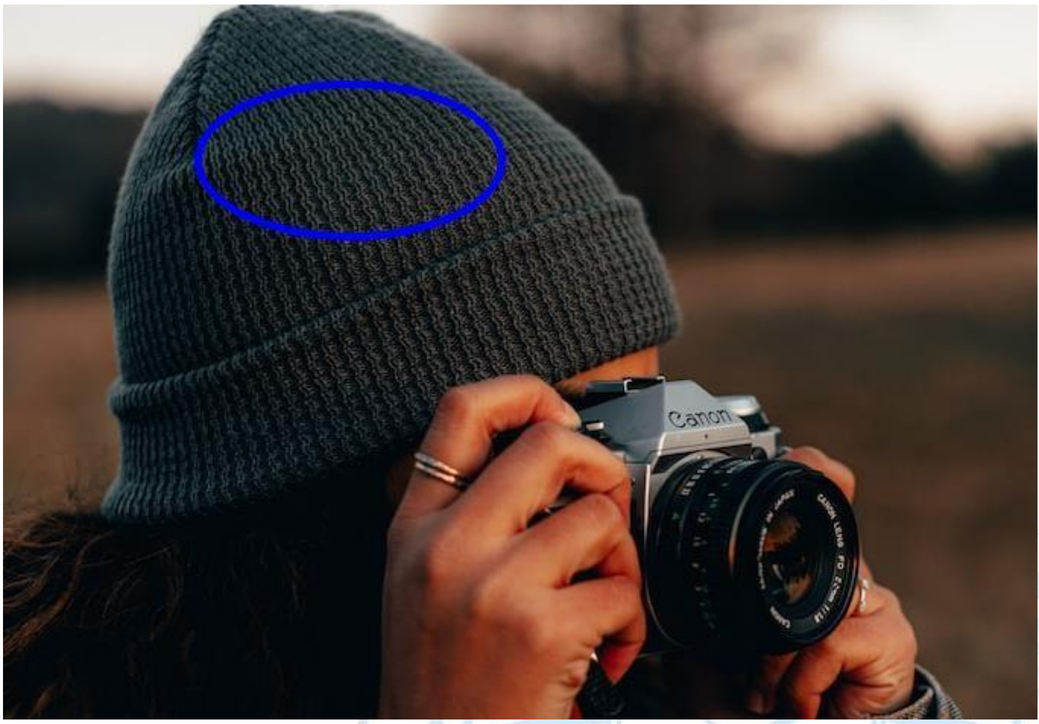
Output:



Ellipse image:

```
imageEllipse = img.copy()
ellipse_center = (310,150)
axis1 = (100,50)
axis2 = (125,50)
cv2.ellipse(imageEllipse, ellipse_center, axis1, 0, 0, 360, (255, 0, 0), thickness=3)
cv2_imshow(imageEllipse)
cv2.waitKey(0)
```

Output:



Result:

Thus the python program to perform various operations on image using open-CV wasverified and executed successfully.

EX NO: 4	IMAGE ENHANCEMENT
DATE :	

AIM:

To perform Image enhancement for understanding color spaces, color space conversion, histogram equalization, convolution, image smoothing, gradients, edge detection.

ALGORITHM:

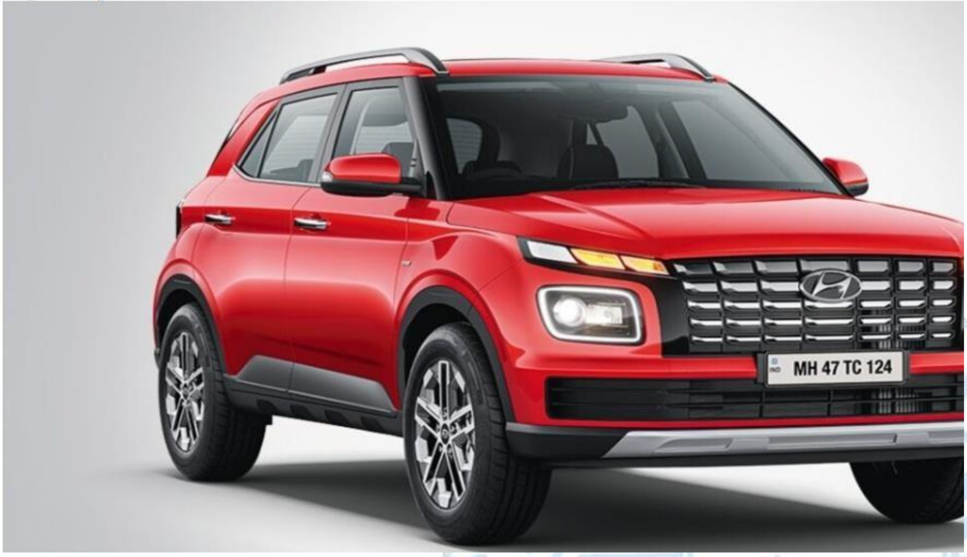
Step 1 : Start the program
Step 2 : Import cv2
Step 3 : Load the image
Step 4 : Convert image from BGR to grayscale
Step 5 : Convert image from BGR to HSV
Step 6 : Apply histogram equalization to enhance contrast
Step 7 Apply Gaussian blur for image smoothing
Step 8 : Calculate gradients using Sobel operators
Step 9 : Draw an ellipse
Step 10 : Calculate magnitude and direction of gradients
Step 11 : Apply Canny edge detection
Step 12 : Display the results
Step 13 : Stop the program

PROGRAM:

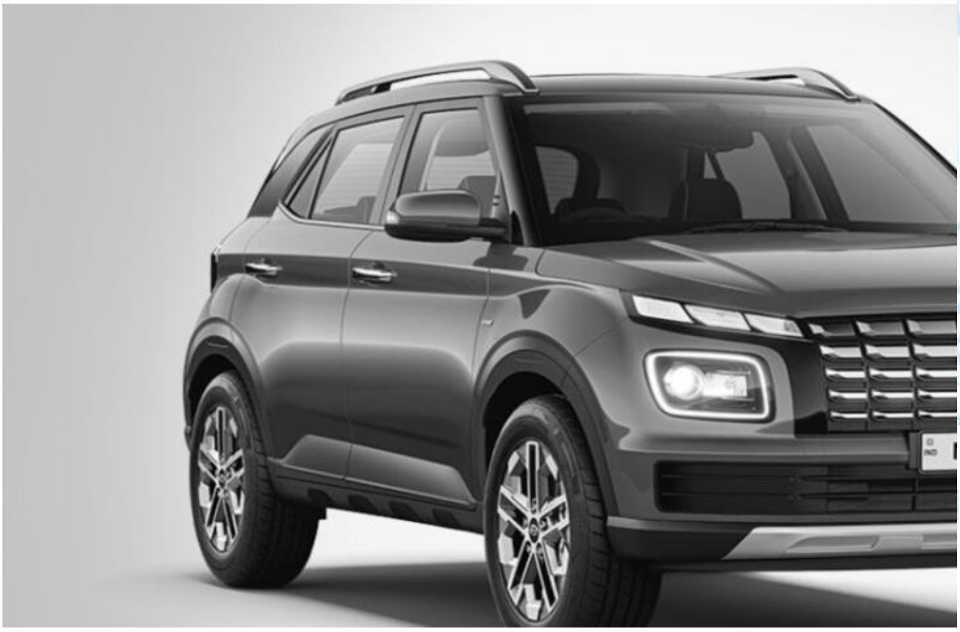
```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
image = cv2.imread('/content/cv.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
equalized = cv2.equalizeHist(gray)
blurred = cv2.GaussianBlur(equalized, (5, 5), 0)
grad_x = cv2.Sobel(blurred, cv2.CV_64F, 1, 0, ksize=3)
grad_y = cv2.Sobel(blurred, cv2.CV_64F, 0, 1, ksize=3)
mag = np.sqrt(grad_x**2 + grad_y**2)
angle = np.arctan2(grad_y, grad_x)
edges = cv2.Canny(blurred, 100, 200)
print("Original image")
cv2_imshow(image)
print("\nConversion of BGR to GRAY")
cv2_imshow(gray)
print("\nConversion of BGR to HSV")
cv2_imshow(hsv)
print("\nHistogram equalization")
cv2_imshow(equalized)
print("\nSmoothing")
cv2_imshow(blurred)
print("\nGradient")
cv2_imshow(mag.astype(np.uint8))
print("\nEdge detection")
cv2_imshow(edges)
```

OUTPUT:

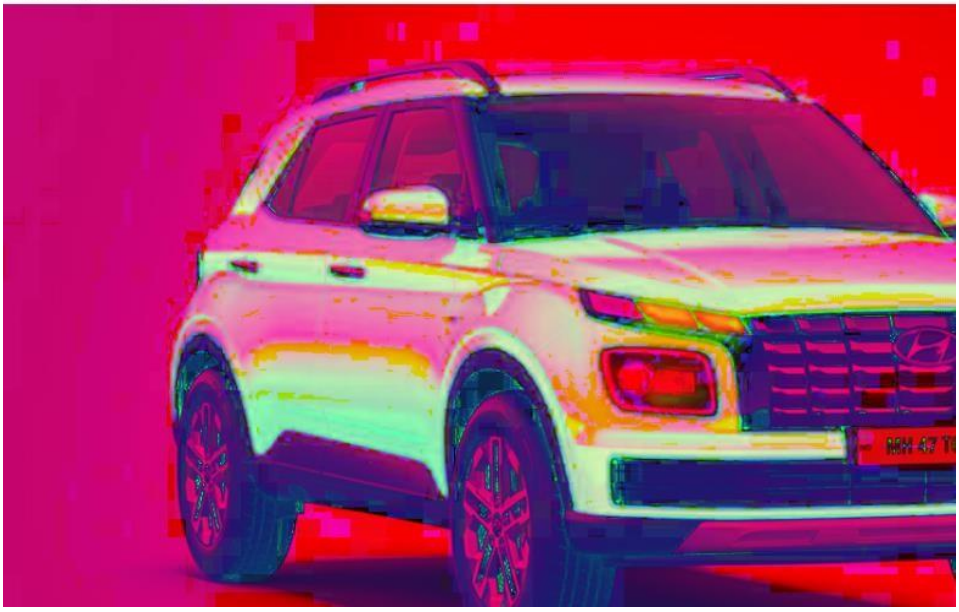
Original image

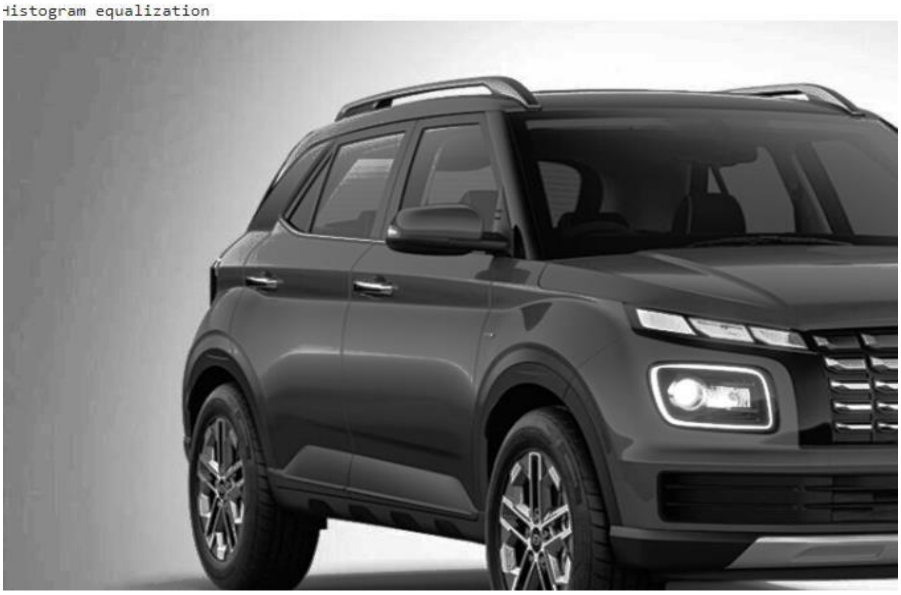


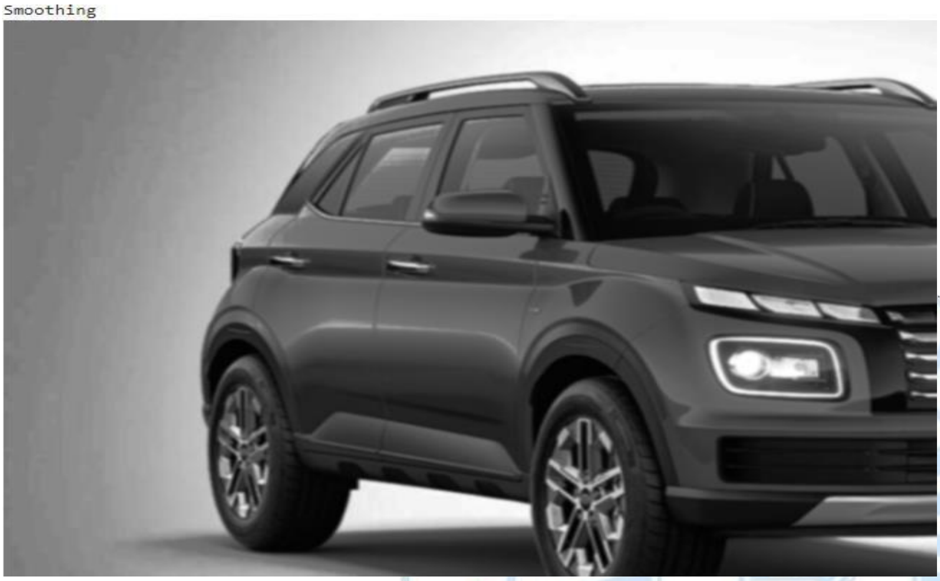
Conversion of BGR to GRAY

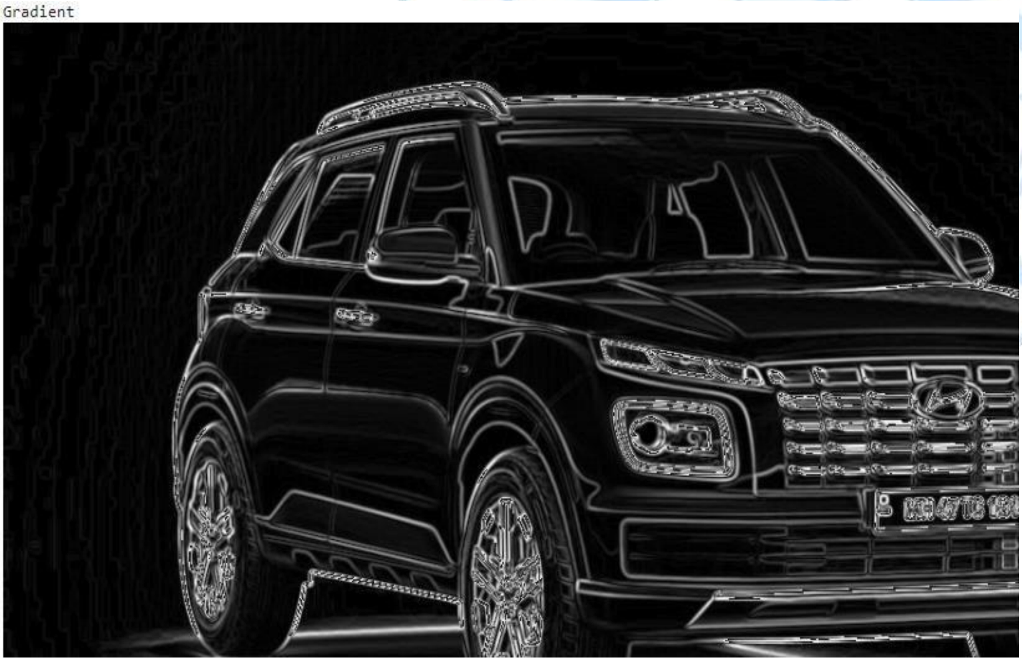


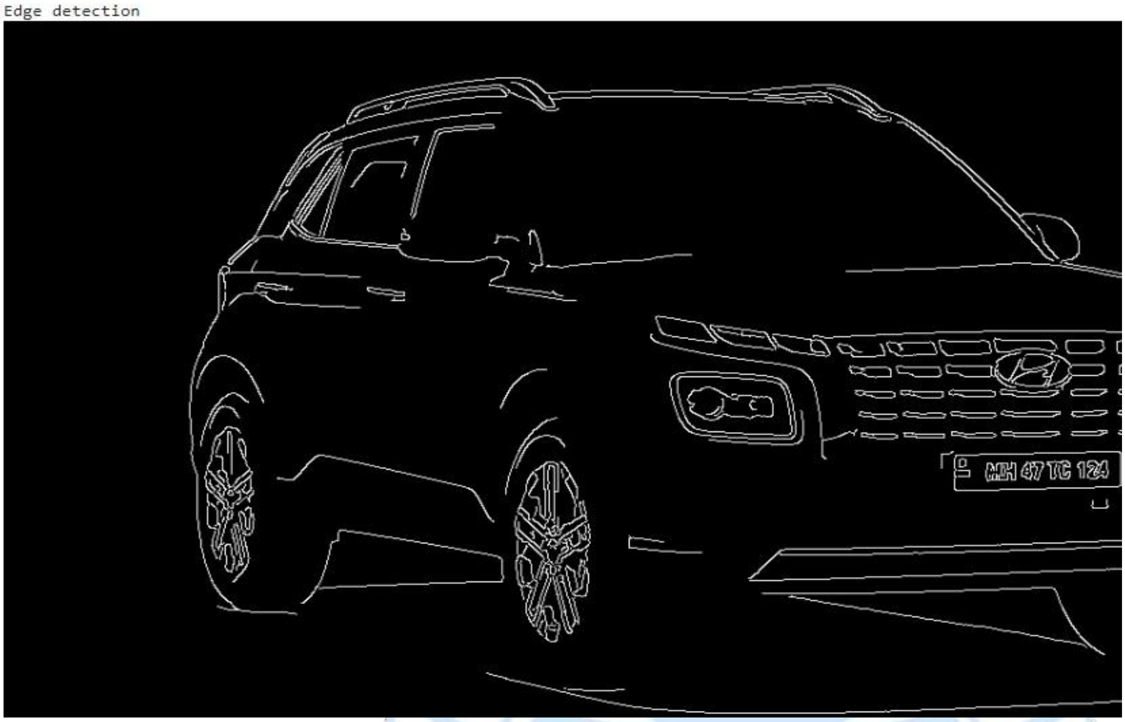
Conversion of BGR to HSV




Histogram equalization


Smoothing


Gradient



Edge detection

RESULT:

Thus the image enhancement has been done successfully and the output is verified.

EX NO: 5	IMAGE FEATURES AND IMAGE ALIGNMENT
DATE :	

AIM:

To perform image features and image alignment image transformation fourier, hough, extract ORB image features,features matching,features matching cloning,feature matching based on image alignment.

ALGORITHM:

Step1:Start the program
Step2:import cv2 and matplotlib
Step3:Load the image
Step4:Display the fourierbimagebof the image
Step5:Display the hough edged image
Step6:Display the ORB matching images using matplotlib
Step7:Display the cloned image
Step8:display the featured image
Step9:Stop the program

PROGRAM:

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
img = cv2.imread('/content/cv.jpg')
cv2_imshow(img)
print("\n\t\t\t\t\t\t Original Image\n")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#fourier
fourier = cv2.dft(np.float32(gray), flags=cv2.DFT_COMPLEX_OUTPUT)
fourier_shift = np.fft.fftshift(fourier)
magnitude = 20*np.log(cv2.magnitude(fourier_shift[:,:,0],fourier_shift[:,:,1]))
magnitude = cv2.normalize(magnitude, None, 0, 255, cv2.NORM_MINMAX,
cv2.CV_8UC1)
cv2_imshow(magnitude)
print("\n\t\t\t\t\t\t Fourier Image\n")
#hough
edges = cv2.Canny(gray,50,200,apertureSize = 3)
minLineLength = 10
maxLineGap = 5
lines = cv2.HoughLinesP(edges,1,np.pi/180,50,minLineLength,maxLineGap)
for line in lines:
for x1,y1,x2,y2 in line:
cv2.line(img,(x1,y1),(x2,y2),(0,255,0),2)
cv2_imshow(img)
cv2_imshow(edges)
print("\n\t\t\t\t\t\t Hough Edged Image\n")
cv2.waitKey(0)
cv2.destroyAllWindows()
```

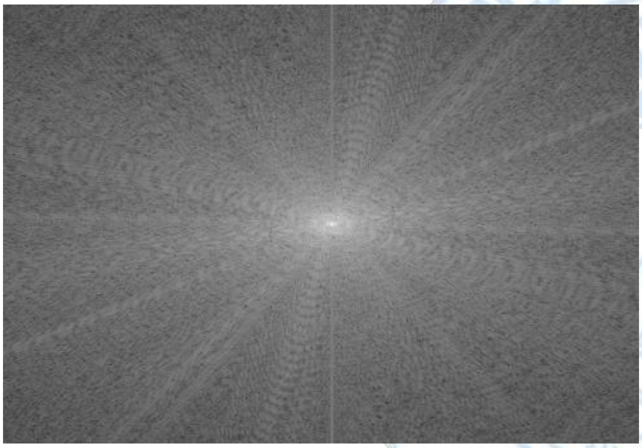
```
#ORB Matching
query_img = cv2.imread('/content/cv.jpg')
train_img = cv2.imread('/content/cv.jpg')
query_img_bw = cv2.cvtColor(query_img,cv2.COLOR_BGR2GRAY)
train_img_bw = cv2.cvtColor(train_img, cv2.COLOR_BGR2GRAY)
orb = cv2.ORB_create()
queryKeypoints, queryDescriptors = orb.detectAndCompute(query_img_bw,None)
trainKeypoints, trainDescriptors = orb.detectAndCompute(train_img_bw,None)
matcher = cv2.BFMatcher()
matches = matcher.match(queryDescriptors,trainDescriptors)
final_img = cv2.drawMatches(query_img, queryKeypoints,
train_img, trainKeypoints, matches[:20],None)
final_img = cv2.resize(final_img, (1000,650))
cv2_imshow(final_img)
print("\n\t\t\t\t\t\t ORB Matching Image\n")
cv2.waitKey(3000)
#cloning
im = cv2.imread("/content/cv.jpg")
obj = cv2.imread("/content/cv.jpg")
obj = cv2.resize(obj, (im.shape[1], im.shape[0]))
mask = 255 * np.ones(obj.shape, obj.dtype)
center = (im.shape[1] // 2, im.shape[0] // 2)
mixed_clone = cv2.seamlessClone(obj, im, mask, center, cv2.MIXED_CLONE)
cv2_imshow(mixed_clone)
print("\n\t\t\t\t\t\t\t Cloned Image\n")
#alignment match
img1_color = cv2.imread("/content/cv.jpg") # Image to be aligned.
img2_color = cv2.imread("/content/cv.jpg") # Reference image.
img1 = cv2.cvtColor(img1_color, cv2.COLOR_BGR2GRAY)
img2 = cv2.cvtColor(img2_color, cv2.COLOR_BGR2GRAY)
height, width = img2.shape
orb_detector = cv2.ORB_create(5000)
kp1, d1 = orb_detector.detectAndCompute(img1, None)
kp2, d2 = orb_detector.detectAndCompute(img2, None)
matcher = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck = True)
matches = matcher.match(d1, d2)
matches = sorted(matches, key=lambda x: x.distance)
matches = matches[:int(len(matches)*0.9)]
no_of_matches = len(matches)
p1 = np.zeros((no_of_matches, 2))
p2 = np.zeros((no_of_matches, 2))
for i in range(len(matches)):
p1[i, :] = kp1[matches[i].queryIdx].pt
p2[i, :] = kp2[matches[i].trainIdx].pt
homography, mask = cv2.findHomography(p1, p2, cv2.RANSAC)
transformed_img = cv2.warpPerspective(img1_color , homography, (width, height))
cv2_imshow (transformed_img)
```

```
print("\n\t\t\t\t\t\t Feature matching Image\n")
```

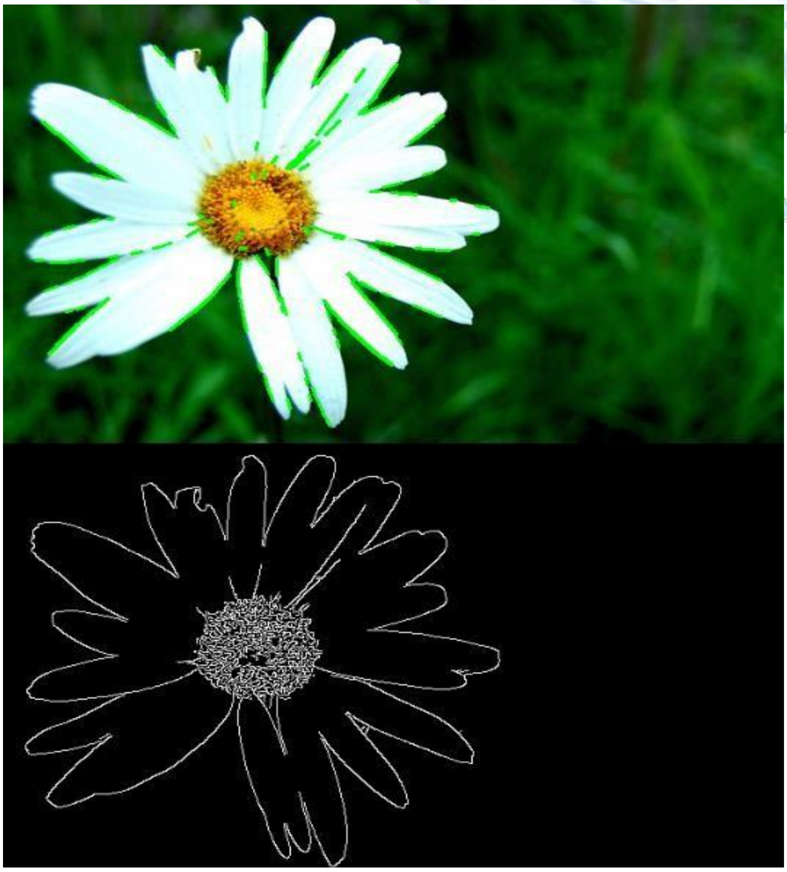
OUTPUT:

Original image:

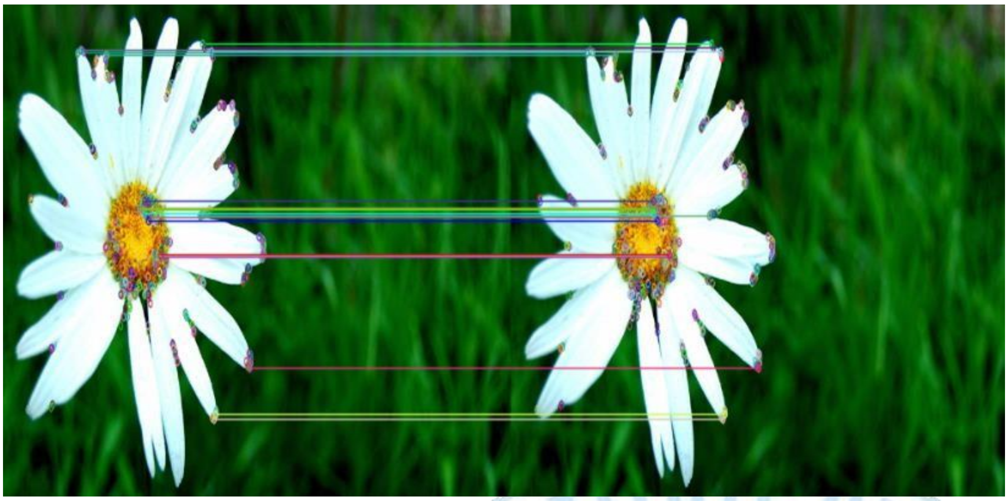
Fourier:



Hough edge images:



ORB matching image :



Cloned image:

Feature matching:

RESULT:

Thus the program for image features and image alignment was executed and the output verified successfully.

EX NO: 6	IMAGE SEGMENTATION USING GRAPHCUT/GRABCUT
DATE :	

AIM:

To write a program to implement the image segmentation using graphcut and grabcut.

ALGORITHM:

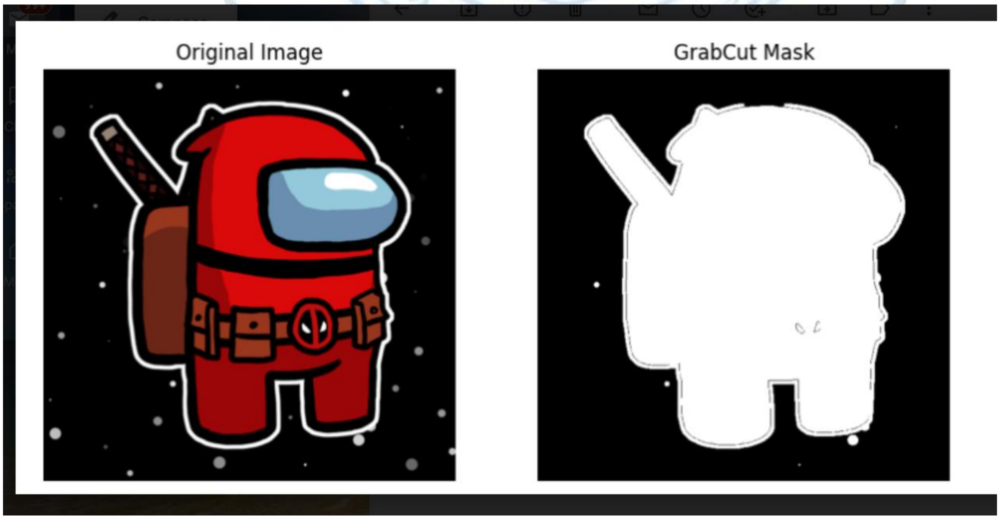
Step 1: Start the program.
Step 2: Load the image.
Step 3: Create a mask for the foreground and background.
Step 4: Define the rectangle enclosing the object of interest.
Step 5: Initialize the foreground and background models.
Step 6: Apply GraphCut algorithm.
Step 7: Modify the mask to create a binary mask for the foreground and background.
Step 8: Apply the mask to the original image.
Step 9: Reset the mask for GrabCut.
Step 10: Apply GrabCut algorithm.
Step 11: Modify the mask to create a binary mask for the foreground and background.
Step 12: Apply the mask to the original image.
Step 13: Display the original image, GraphCut segmented image, and GrabCut segmented image.
Step 14: Stop the program.

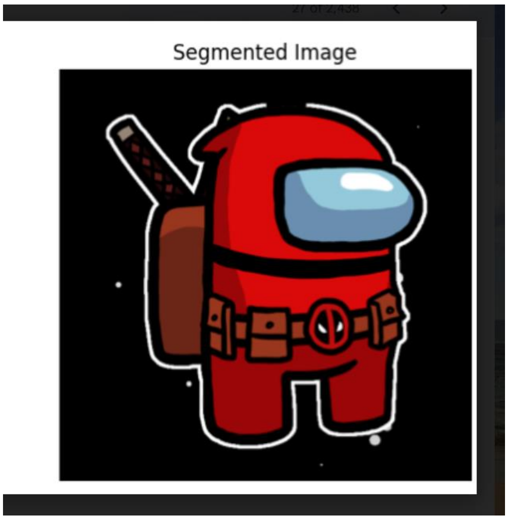
PROGRAM:

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

# Directly specify the input image path
image_path = "/content/asd.png"  # Replace with the actual image path

# Load the input image
image = cv2.imread(image_path)

# Initialize the mask with an all-background mask
mask = np.zeros(image.shape[:2], np.uint8)

# Define a rectangle around the object of interest
rect = (50, 50, 500, 600)  # (x, y, width, height)

# Initialize GrabCut with a bounding rectangle
cv2.grabCut(image, mask, rect, None, None, 5, cv2.GC_INIT_WITH_RECT)

# Create a mask where all definite background and probable background pixels are marked as 0, others as 1
mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')

# Multiply the original image with the new mask to get the segmented image
segmented = image * mask2[:, :, np.newaxis]

# Plot the original image, GrabCut mask, and segmented image side by side
fig, axes = plt.subplots(1, 3, figsize=(15, 5))
axes[0].imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
```

```
axes[0].set_title("Original Image")
axes[0].axis('off')

axes[1].imshow(mask2, cmap='gray')
axes[1].set_title("GrabCut Mask")
axes[1].axis('off')

axes[2].imshow(cv2.cvtColor(segmented, cv2.COLOR_BGR2RGB))
axes[2].set_title("Segmented Image")
axes[2].axis('off')

plt.show()
```

OUTPUT:





RESULT:

Thus, the above program to implement the image segmentation using graphcut and grabcut has been executed and the output is verified successfully

EX NO: 7	**CAMERA CALIBRATION WITH CIRCULAR GRID**
DATE :	

AIM:

To write a program for camera calibration with circular grid.

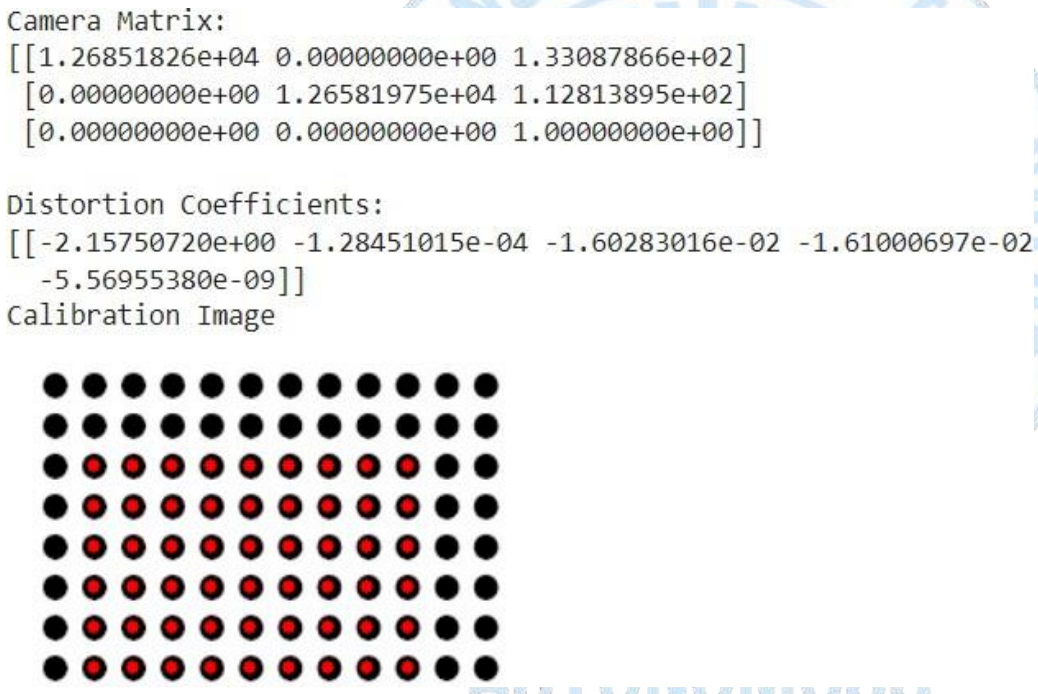
ALGORITHM:

Step 1: Start the program.
Step 2: Define the number of grid corners in the calibration pattern.
Step 3: Load the input image.
Step 4: Convert the image to grayscale.
Step 5: Generate the grid points in the real world (assuming a square grid).
Step 6: Find the circular grid corners.
Step 7: If corners are found, add them to the object and image points lists.
Step 8: Perform camera calibration.
Step 9: Save the calibration parameters to a file (you can use them later).
Step 10: Print the camera matrix and distortion coefficients.
Step 11: Draw circles at the detected corner positions.
Step 12: Display the input image with detected corners (for verification).
Step 13: Stop the program.

PROGRAM:

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
num_rows = 6
num_cols = 9
input_image_path = '/content/chess.png'
image = cv2.imread(input_image_path)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
grid_size = 1.0 # Size of each square in your calibration grid (e.g., 1 inch)
objp = np.zeros((num_rows * num_cols, 3), np.float32)
objp[:, :2] = np.mgrid[0:num_cols, 0:num_rows].T.reshape(-1, 2) * grid_size
ret, corners = cv2.findCirclesGrid(
    gray, (num_cols, num_rows), None, cv2.CALIB_CB_SYMMETRIC_GRID
)
if ret:
    obj_points = [objp]
    img_points = [corners]
    ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(obj_points, img_points, gray.shape[::-1], None, None)
    calibration_data = {
        "camera_matrix": mtx,
        "distortion_coefficients": dist,
    }
    np.save("camera_calibration.npy", calibration_data)
    print("Camera Matrix:")
    print(mtx)
    print("\nDistortion Coefficients:")
    print(dist)
       for corner in corners:
        cv2.circle(image, (int(corner[0][0]), int(corner[0][1])), 3, (0, 0, 255), -1)
    print("Calibration Image")
    cv2_imshow(image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
else:
    print("Corners not found. Calibration failed.")
```

OUTPUT:



```
Camera Matrix:
[[1.26851826e+04 0.00000000e+00 1.33087866e+02]
 [0.00000000e+00 1.26581975e+04 1.12813895e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]

Distortion Coefficients:
[[-2.15750720e+00 -1.28451015e-04 -1.60283016e-02 -1.61000697e-02
  -5.56955380e-09]]
Calibration Image
```

RESULT:

Thus, the above program to implement camera calibration with circular grid has been executed successfully and the output is verified.