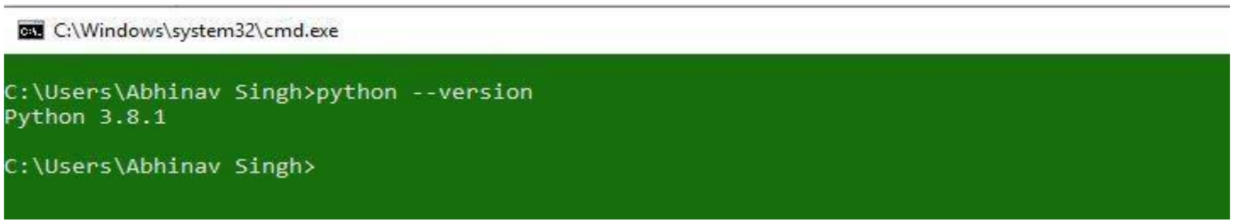
EX.NO: 1
DATE:

OPENCV INSTALLATION AND WORKING WITH PYTHON

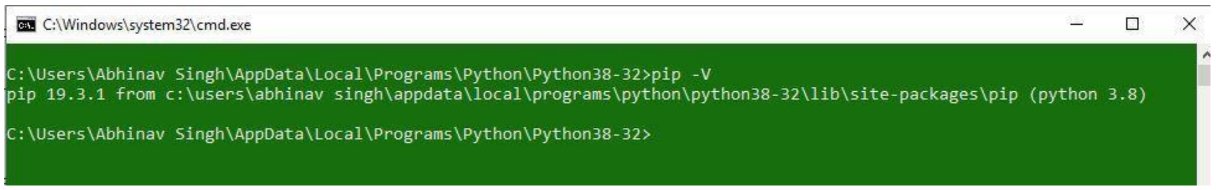
OpenCV is the huge open-source library for computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today‟s systems. By using it, one can process images and videos to identify objects, faces, or even the handwriting of a human. When it integrated with various libraries, such as Numpy, python is capable of processing the OpenCV array structure for analysis. To Identify image patterns and its various features we use vector space and perform mathematical operations on these features. To install OpenCV, one must have Python and PIP, preinstalled on their system. To check if your system already contains Python, go through the following instructions: 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.



```
C:\Users\Abhinav Singh>python --version
Python 3.8.1

C:\Users\Abhinav Singh>
```

PIP is a package management system used to install and manage software packages/libraries written in Python. These files are stored in a large "on-line repository" termed as Python Package Index (PyPI). To check if PIP is already installed on your system, just go to the command line and execute the following command:
pip -V



```
C:\Users\Abhinav Singh\AppData\Local\Programs\Python\Python38-32>pip -V
pip 19.3.1 from c:\users\abhinav singh\appdata\local\programs\python\python38-32\lib\site-packages\pip (python 3.8)

C:\Users\Abhinav Singh\AppData\Local\Programs\Python\Python38-32>
```

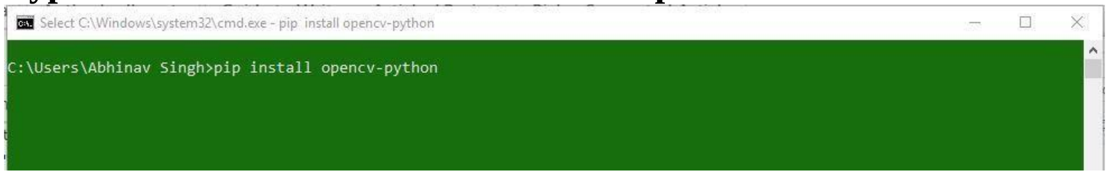
Downloading and Installing OpenCV:

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

Beginning with the installation:

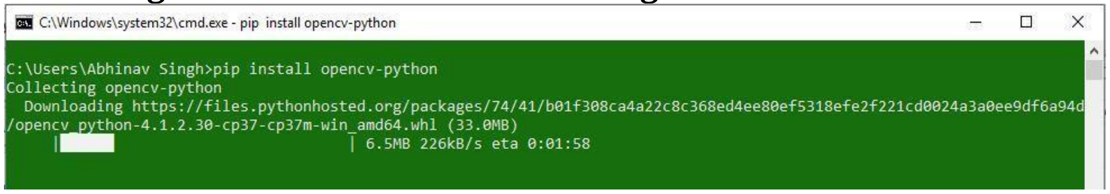
- **Type the command in the Terminal and proceed:**



Select C:\Windows\system32\cmd.exe - pip install opencv-python

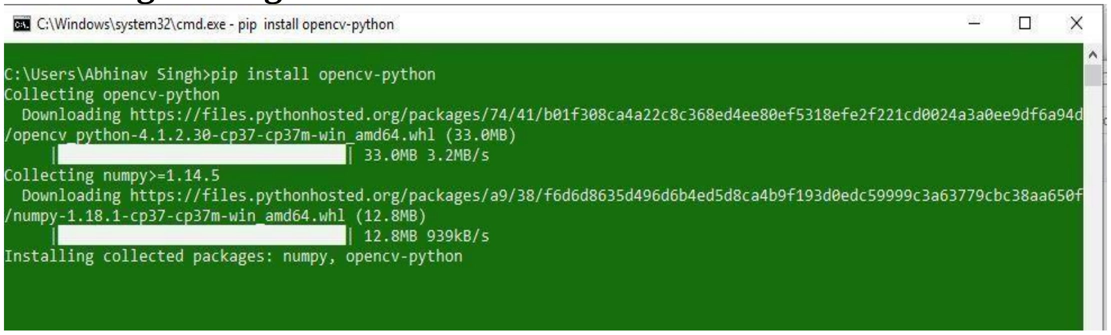
```
C:\Users\Abhinav Singh>pip install opencv-python
```

- **Collecting Information and downloading data:**



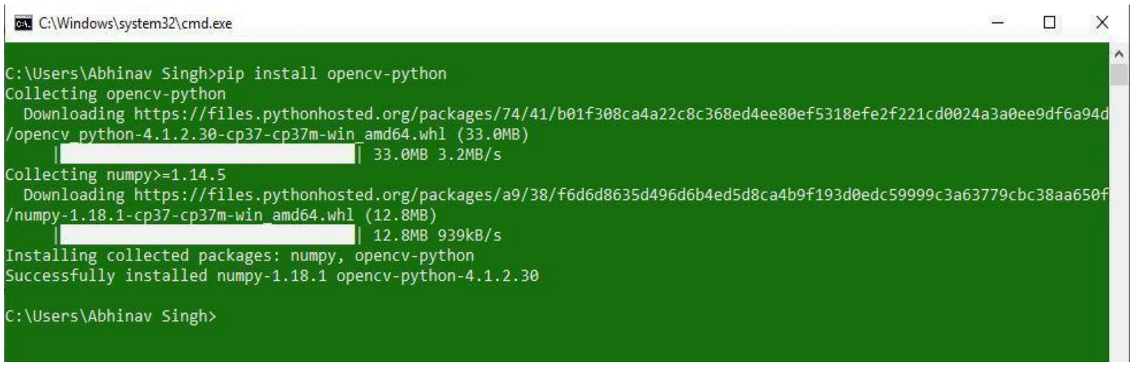
```
C:\Users\Abhinav Singh>pip install opencv-python
Collecting opencv-python
  Downloading https://files.pythonhosted.org/packages/74/41/b01f308ca4a22c8c368ed4ee80ef5318efe2f221cd0024a3a0ee9df6a94d
/opencv_python-4.1.2.30-cp37-cp37m-win_amd64.whl (33.0MB)
     |███                              | 6.5MB 226kB/s eta 0:01:58
```

- **Installing Packages:**



```
C:\Users\Abhinav Singh>pip install opencv-python
Collecting opencv-python
  Downloading https://files.pythonhosted.org/packages/74/41/b01f308ca4a22c8c368ed4ee80ef5318efe2f221cd0024a3a0ee9df6a94d
/opencv_python-4.1.2.30-cp37-cp37m-win_amd64.whl (33.0MB)
     |████████████████████████████████| 33.0MB 3.2MB/s
Collecting numpy>=1.14.5
  Downloading https://files.pythonhosted.org/packages/a9/38/f6d6d8635d496d6b4ed5d8ca4b9f193d0edc59999c3a63779cbc38aa650f
/numpy-1.18.1-cp37-cp37m-win_amd64.whl (12.8MB)
     |████████████████████████████████| 12.8MB 939kB/s
Installing collected packages: numpy, opencv-python
```

- **Finished Installation:**



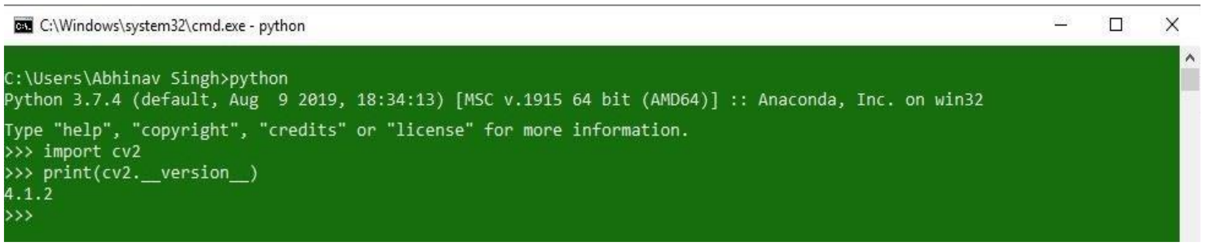
```
C:\Users\Abhinav Singh>pip install opencv-python
Collecting opencv-python
  Downloading https://files.pythonhosted.org/packages/74/41/b01f308ca4a22c8c368ed4ee80ef5318efe2f221cd0024a3a0ee9df6a94d
/opencv_python-4.1.2.30-cp37-cp37m-win_amd64.whl (33.0MB)
     |████████████████████████████████| 33.0MB 3.2MB/s
Collecting numpy>=1.14.5
  Downloading https://files.pythonhosted.org/packages/a9/38/f6d6d8635d496d6b4ed5d8ca4b9f193d0edc59999c3a63779cbc38aa650f
/numpy-1.18.1-cp37-cp37m-win_amd64.whl (12.8MB)
     |████████████████████████████████| 12.8MB 939kB/s
Installing collected packages: numpy, opencv-python
Successfully installed numpy-1.18.1 opencv-python-4.1.2.30

C:\Users\Abhinav Singh>
```

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

python

>>>import cv2

>>>print(cv2.__version__)



```
C:\Users\Abhinav Singh>python
Python 3.7.4 (default, Aug  9 2019, 18:34:13) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> print(cv2.__version__)
4.1.2
>>>
```

RESULT:

Thus the Open CV installation have been executed successfully and the output got verified.

EX.NO: 2 **Basic Image Processing - loading images, Cropping,**
DATE: **Resizing, Thresholding, Contour analysis, Bolb detection**

AIM:

To write a program about the basic image processing using loading images, Cropping, Resizing, Thresholding, Contour analysis, Bolb detection

ALGORITHM:

Step:1-Importing the image via image acquisition tools;

Step:2-Analysing and manipulating the image;

Step:3-Output in which result can be altered image or a report which is based on analysing that image.

PROGRAM:

```
import cv2

import numpy as np

from google.colab import files

from matplotlib import pyplot as plt
```

UPLOAD AN IMAGE

```
uploaded = files.upload()
```

LOAD THE UPLOADED IMAGE

```
image = cv2.imread(„your_uploaded_image.jpg‟)

convert BGR image to RGB for displaying with Matplotlib

image_rgb = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
```

DISPLAY THE IMAGE

Plt.imshow(image_rgb)

Plt.axis(„off‟)

Plt.show()

CROPPING IMAGES

x1, y1, x2, y2 = 100, 100, 300, 300

cropped_image = image[y1:y2, x1:x2]

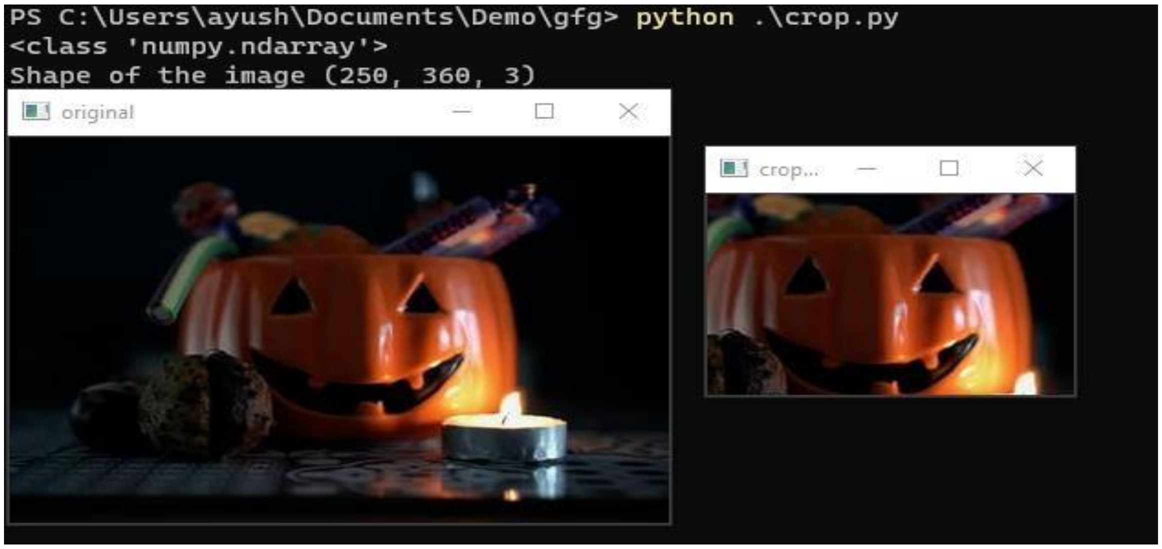
DISPLAY THE CROPPED REGION

Cropped_rgb = cv2.cvtColor(cropped_image,cv2.COLOR_BGR2RGB)

Plt.imshow(cropped_rgb)

Plt.axis(„off‟)

OUTPUT:

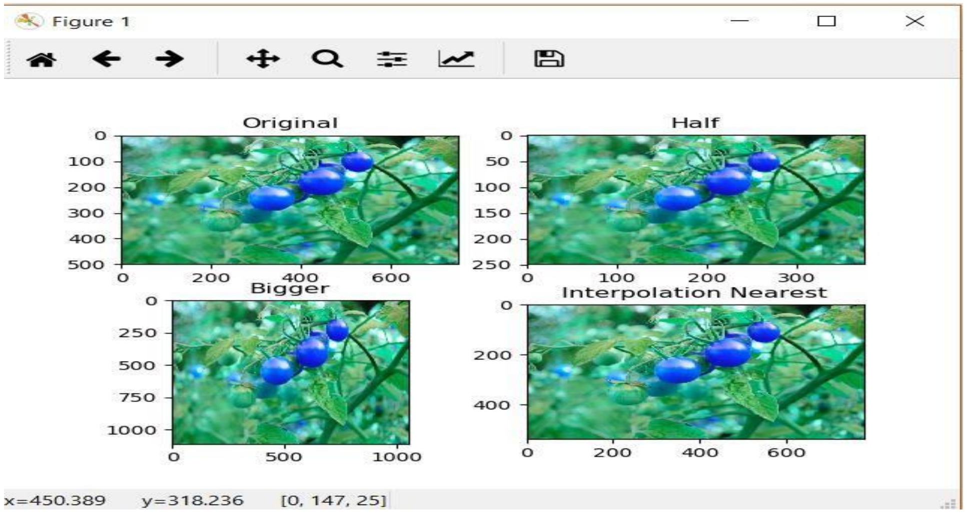


```
PS C:\Users\ayush\Documents\Demo\gfg> python .\crop.py
<class 'numpy.ndarray'>
Shape of the image (250, 360, 3)
```

RESIZING IMAGES

```
import cv2

import numpy as np

import matplotlib.pyplot as plt

image = cv2.imread(r"D:\sims\eb\sim21\EB-ML-06-10-2022-Test-Output-
15\PERFORATION\Overkill\Fail\Blister 1 2022-03-12 12-59-43.859 T0 M0
G0 3 PERFORATION Mono.bmp", 1)

# Loading the image

half = cv2.resize(image, (0, 0), fx = 0.1, fy = 0.1)

bigger = cv2.resize(image, (1050, 1610))

stretch_near = cv2.resize(image, (780, 540),

                               interpolation = cv2.INTER_LINEAR)

Titles =["Original", "Half", "Bigger", "Interpolation Nearest"]

images =[image, half, bigger, stretch_near]

count = 4

for i in range(count):

        plt.subplot(2, 2, i + 1)

        plt.title(Titles[i])

        plt.imshow(images[i])

plt.show()
```

OUTPUT:

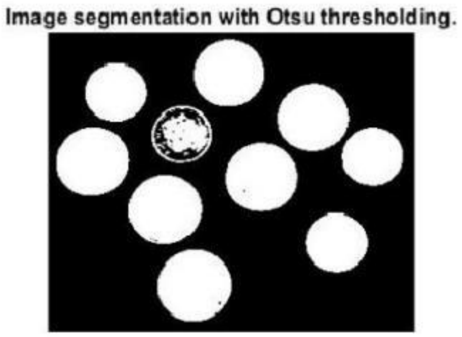


x=450.389 y=318.236 [0, 147, 25]

THRESHOLDING

```
% Matlab program to perform Otsu's thresholding

image=(imread("coins.jpg"));

figure(1);

imshow(image);

title("Original image.");

[counts,x] = imhist(image,16);

thresh= otsuthresh(counts);

otsu=imbinarize(image,thresh);

figure(2);

imshow(otsu);

title("Image segmentation with Otsu thresholding.");
```

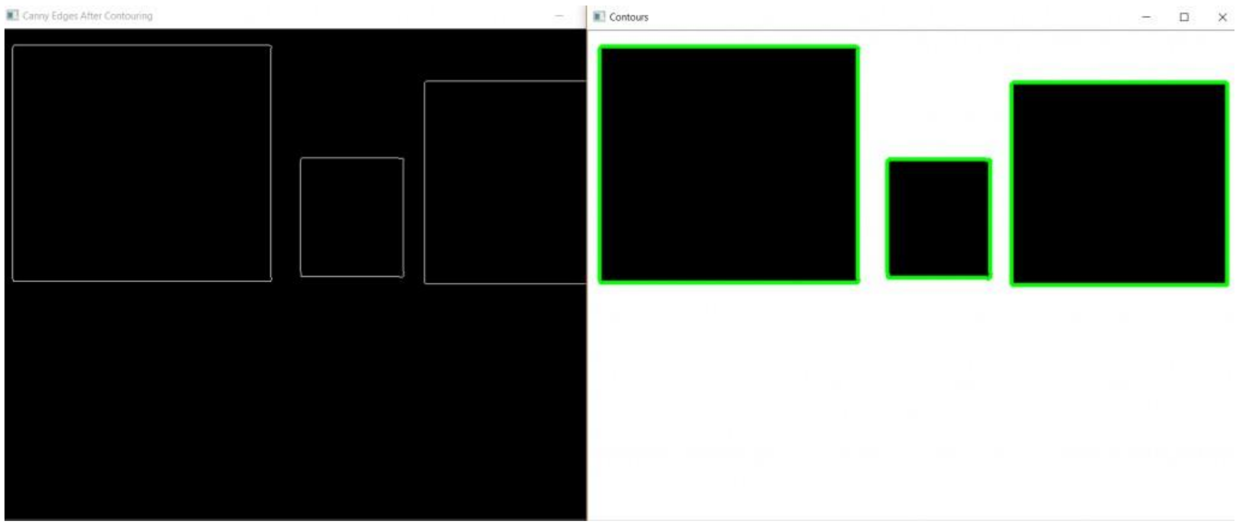
OUTPUT:



CONTOUR ANALYSIS

```
import cv2

import numpy as np

# Let's load a simple image with 3 black squares

image = cv2.imread('C://Users//gfg//shapes.jpg')

cv2.waitKey(0)

# Grayscale

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Find Canny edges

edged = cv2.Canny(gray, 30, 200)

cv2.waitKey(0)

# Finding Contours

# Use a copy of the image e.g. edged.copy()

# since findContours alters the image

contours, hierarchy = cv2.findContours(edged,
        cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

cv2.imshow('Canny Edges After Contouring', edged)

cv2.waitKey(0)

print("Number of Contours found = " + str(len(contours)))

# Draw all contours

# -1 signifies drawing all contours

cv2.drawContours(image, contours, -1, (0, 255, 0), 3)

cv2.imshow('Contours', image)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

OUTPUT :

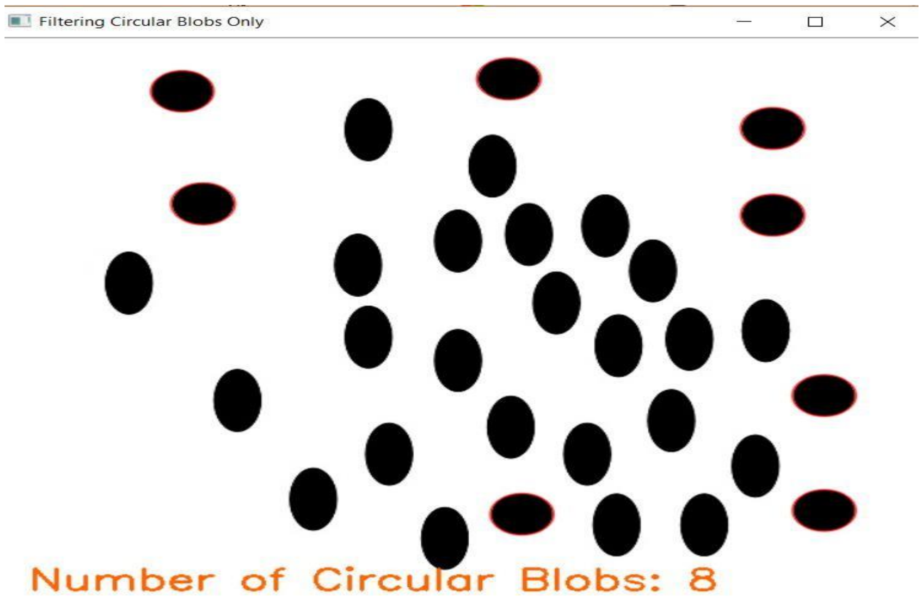


BOLB DETECTION

```
import cv2

import numpy as np

# Load image

image = cv2.imread('C://gfg//images//blobs.jpg', 0)

# Set our filtering parameters

# Initialize parameter setting using cv2.SimpleBlobDetector

params = cv2.SimpleBlobDetector_Params()

# Set Area filtering parameters

params.filterByArea = True

params.minArea = 100

# Set Circularity filtering parameters

params.filterByCircularity = True

params.minCircularity = 0.9

# Set Convexity filtering parameters

params.filterByConvexity = True

params.minConvexity = 0.2

# Set inertia filtering parameters

params.filterByInertia = True

params.minInertiaRatio = 0.01

# Create a detector with the parameters
```

```
detector = cv2.SimpleBlobDetector_create(params)

# Detect blobs

keypoints = detector.detect(image)

# Draw blobs on our image as red circles

blank = np.zeros((1, 1))

blobs = cv2.drawKeypoints(image, keypoints, blank, (0, 0, 255),

          cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
number_of_blobs = len(keypoints)

text = "Number of Circular Blobs: " + str(len(keypoints))

cv2.putText(blobs, text, (20, 550),

                              cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 100, 255), 2)

# Show blobs

cv2.imshow("Filtering Circular Blobs Only", blobs)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

OUTPUT:



Number of Circular Blobs: 8

RESULT:

Thus the Basic Image Processing - loading images, Cropping, Resizing, Thresholding, Contour analysis, Bolb detection have been executed successfully and the output got verified.

EX.NO: 3 **Image Annotation – Drawing lines, text circle, rectangle,**
DATE: **ellipse on images**

AIM:

To write a program about Image Annotation – Drawing lines, text circle, rectangle, ellipse on images

ALGORITHM:

1. Preparing the image dataset.
2. Specifying object classes that annotators will use to label images.
3. Assigning labels to images.
4. Marking objects within each image by drawing bounding boxes.
5. Selecting object class labels for each box.

PROGRAM:

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import cv2
# Load an image using OpenCV
image = cv2.imread("/parrot.jpg")
# Create a Matplotlib figure and axis
fig, ax = plt.subplots(1)
ax.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
# Draw a rectangle
rect = patches.Rectangle((100, 100), 200, 150, linewidth=2, edgecolor='r',
facecolor='none')
ax.add_patch(rect)
# Add text annotation
ax.text(50, 50, "EXAMPLE ANNOTATION", fontsize=12, color='b')
# Show the annotated image
plt.show()
```

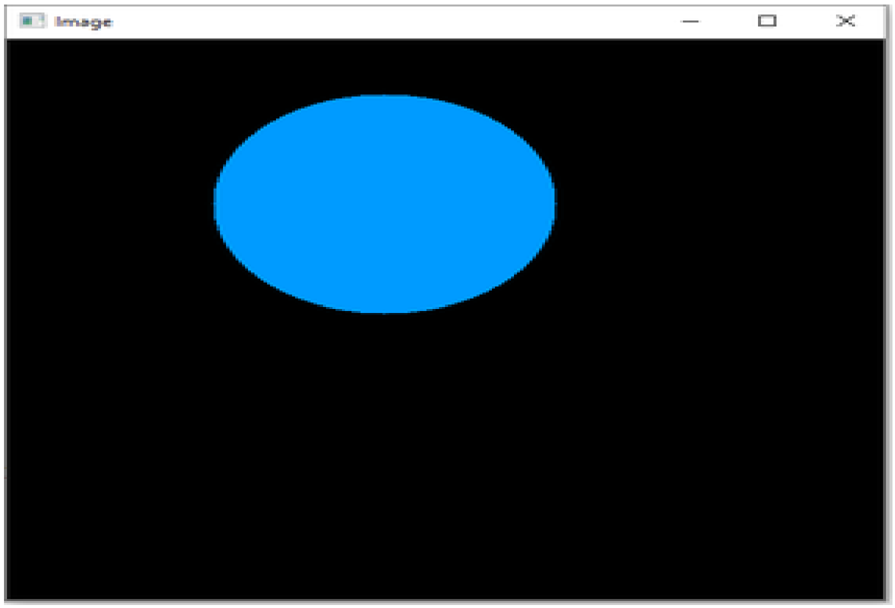
DRAW LINES

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
img=mpimg.imread('/parrot.jpg')
plt.figure(figsize=(8,8))
plt.imshow(img)
plt.title('annotated image')
plt.axis('off')
line1= (100,100,300,300)
line2=(200,200,400,400)
plt.plot([line1[0],line1[2]],[line1[1],line1[3]],color ='red',linewidth=2)
plt.plot([line2[0],line2[2]],[line2[1],line2[3]],color ='blue',linewidth=2)
plt.show()
```

TEXT CIRCLES

```
import cv2
import numpy as np
# Reading an image in default mode
Img = np.zeros((512, 512, 3), np.uint8)
# Window name in which image is displayed
window_name = 'Image'
# Center coordinates
center_coordinates = (220, 150)
# Radius of circle
radius = 100
# Red color in BGR
color = (255, 133, 233)
# Line thickness of -1 px
thickness = -1
# Using cv2.circle() method
# Draw a circle of red color of thickness -1 px
image = cv2.circle(Img, center_coordinates, radius, color, thickness)
# Displaying the image
cv2.imshow(window_name, image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



RESULT:

Thus the Image Annotation – Drawing lines, text circle, rectangle, ellipse on images have been executed successfully and the output got verified

EX.NO: 4 **Image Enhancement - Understanding Color spaces, color space conversion, Histogram equialization, Convolution, Image smoothing, Gradients, Edge Detection**

DATE:

AIM:

To write a program about Image Enhancement - Understanding Color spaces, color space conversion, Histogram equialization, Convolution, Image smoothing, Gradients, Edge Detection

ALGORITHM:

1. **Visualization:** The objects which are not visible, they are observed.
2. **Image sharpening and restoration:** It is used for better image resolution.
3. **Image retrieval:** An image of interest can be seen
4. **Measurement of pattern:** In an image, all the objects are measured.
5. **Image Recognition:** Each object in an image can be distinguished.

PROGRAM:

```
import matplotlib.pyplot as plt
import cv2
# Load an image using OpenCV
image = cv2.imread("your_image.jpg")
# Convert the image from BGR to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Convert the image from BGR to HSV color space
hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
# Extract and enhance the saturation channel
hsv_image[:, :, 1] = hsv_image[:, :, 1] * 1.5  # Increase saturation by a factor
# Convert back to BGR color space
```

```
enhanced_image = cv2.cvtColor(hsv_image, cv2.COLOR_HSV2BGR)
# Display the original and enhanced images using Matplotlib
plt.figure(figsize=(12, 6))
# Original Image
plt.subplot(131)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title("Original Image")
plt.axis('off')
# Grayscale Image
plt.subplot(132)
plt.imshow(gray_image, cmap='gray')
plt.title("Grayscale Image")
plt.axis('off')
# Enhanced Image
plt.subplot(133)
plt.imshow(cv2.cvtColor(enhanced_image, cv2.COLOR_BGR2RGB))
plt.title("Enhanced Image")
plt.axis('off')
plt.show()
```

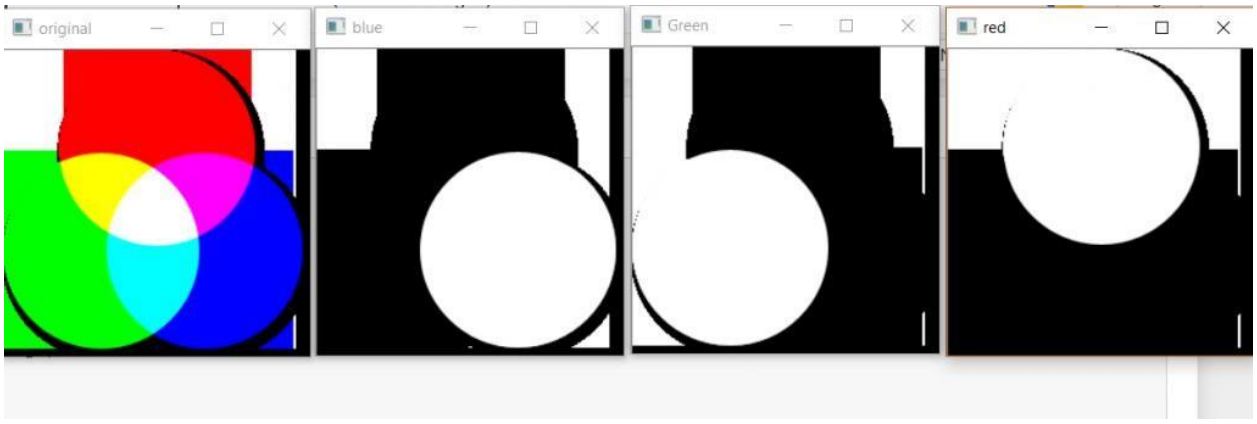
UNDERSTANDING COLOR SPACES

PROGRAM:

```
import cv2
image = cv2.imread('C://Users//Gfg//rgb.png')
B, G, R = cv2.split(image)
# Corresponding channels are separated
cv2.imshow("original", image)
cv2.waitKey(0)
cv2.imshow("blue", B)
cv2.waitKey(0)
cv2.imshow("Green", G)
cv2.waitKey(0)
cv2.imshow("red", R)
```

```
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:

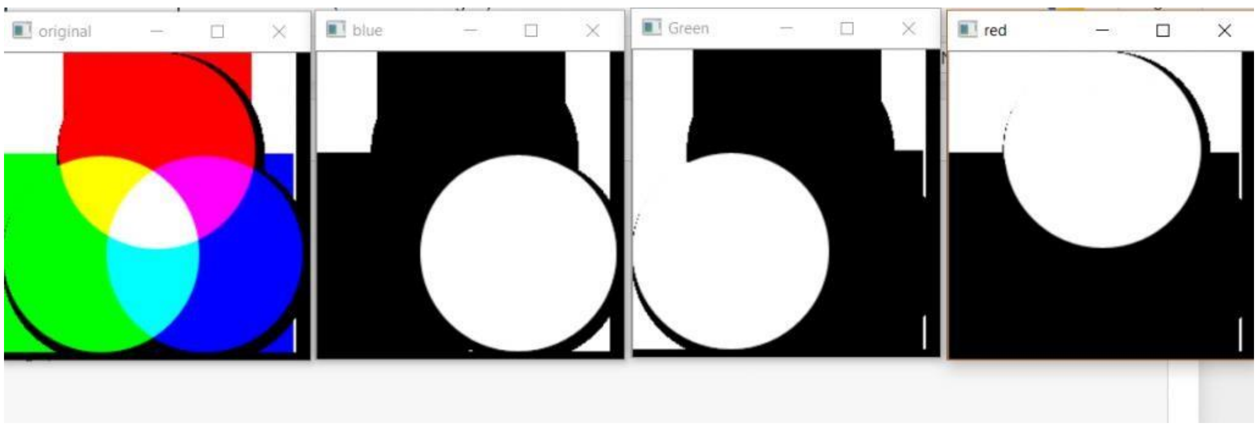


COLOR SPACE CONVERSION

PROGRAM:

```
import cv2
image = cv2.imread('C://Users//Gfg//rgb.png')
B, G, R = cv2.split(image)
# Corresponding channels are separated
cv2.imshow("original", image)
cv2.waitKey(0)
cv2.imshow("blue", B)
cv2.waitKey(0)
cv2.imshow("Green", G)
cv2.waitKey(0)
cv2.imshow("red", R)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

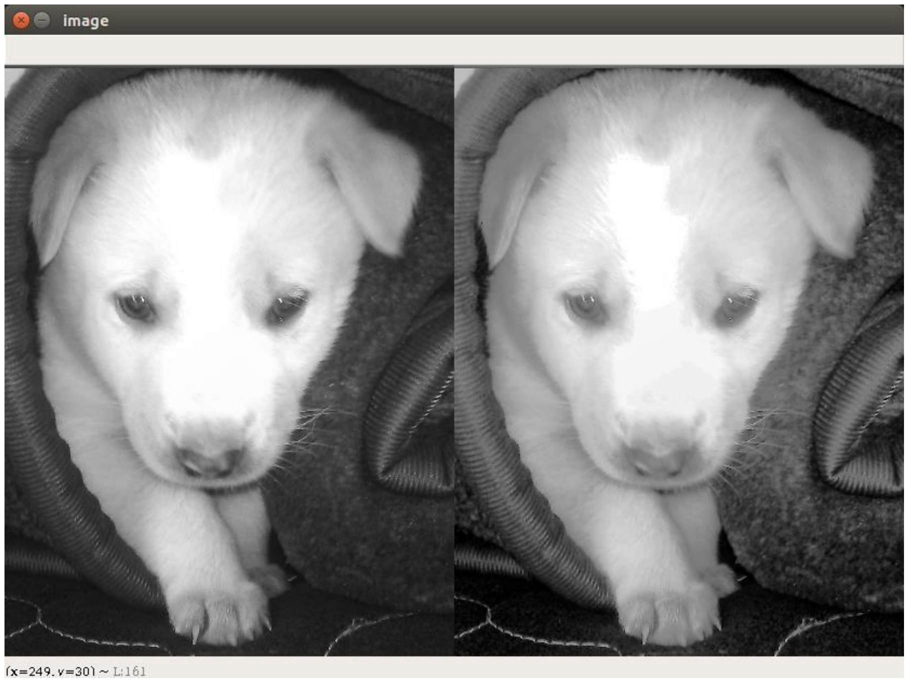
OUTPUT:



HISTOGRAM EQUIALIZATION
PROGRAM:

```
# import Opencv
import cv2
# import Numpy
import numpy as np
# read a image using imread
img = cv2.imread(\'F:\\do_nawab.png\', 0)
# creating a Histograms Equalization
# of a image using cv2.equalizeHist()
equ = cv2.equalizeHist(img)
# stacking images side-by-side
res = np.hstack((img, equ))
# show image input vs output
cv2.imshow(\'image\', res)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT :



CONVOLUTION

PROGRAM:

```
import matplotlib.pyplot as plt
import tensorflow as tf
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
from tensorflow import keras
from keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dropout, Flatten, Dense
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.utils import image_dataset_from_directory
from tensorflow.keras.preprocessing.image import ImageDataGenerator,
load_img
from tensorflow.keras.preprocessing import image_dataset_from_directory
import os
import matplotlib.image as mpimg
from zipfile import ZipFile
data_path = 'dog-vs-cat-classification.zip'
with ZipFile(data_path, 'r') as zip:
        zip.extractall()
        print('The data set has been extracted.')
path = 'dog-vs-cat-classification'
classes = os.listdir(path)
classes
fig = plt.gcf()
fig.set_size_inches(16, 16)
cat_dir = os.path.join('dog-vs-cat-classification/cats')
dog_dir = os.path.join('dog-vs-cat-classification/dogs')
cat_names = os.listdir(cat_dir)
dog_names = os.listdir(dog_dir)
pic_index = 210
cat_images = [os.path.join(cat_dir, fname)
                        for fname in cat_names[pic_index-8:pic_index]]
dog_images = [os.path.join(dog_dir, fname)
                        for fname in dog_names[pic_index-8:pic_index]]
for i, img_path in enumerate(cat_images + dog_images):
        sp = plt.subplot(4, 4, i+1)
        sp.axis('Off')
        img = mpimg.imread(img_path)
        plt.imshow(img)
plt.show()
```

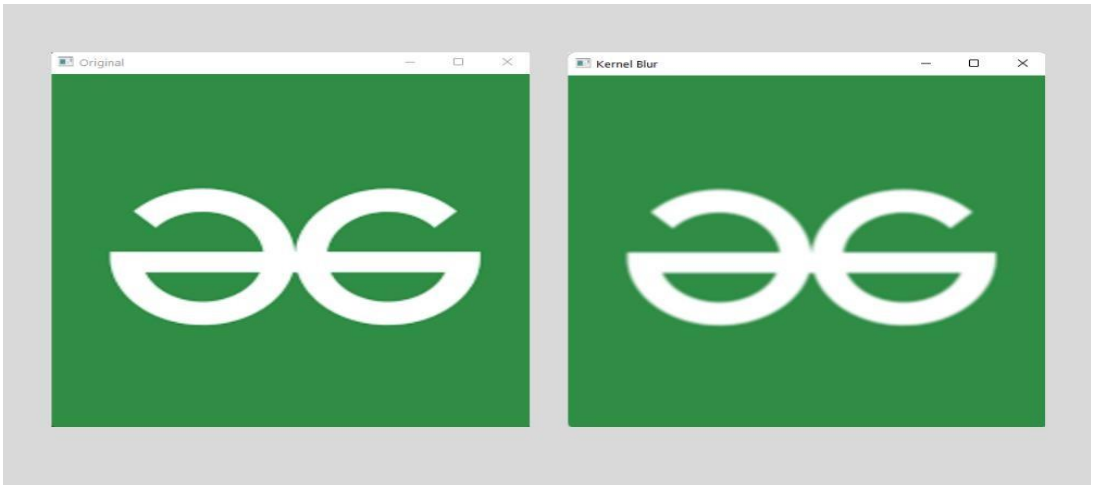
OUTPUT:

IMAGE SMOOTHING
PROGRAM:

```
# Importing the modules
import cv2
import numpy as np
# Reading the image
image = cv2.imread('image.png')
# Creating the kernel with numpy
kernel2 = np.ones((5, 5), np.float32)/25
# Applying the filter
img = cv2.filter2D(src=image, ddepth=-1, kernel=kernel2)
# showing the image
cv2.imshow('Original', image)
cv2.imshow('Kernel Blur', img)
```

cv2.waitKey()
cv2.destroyAllWindows()

OUTPUT:

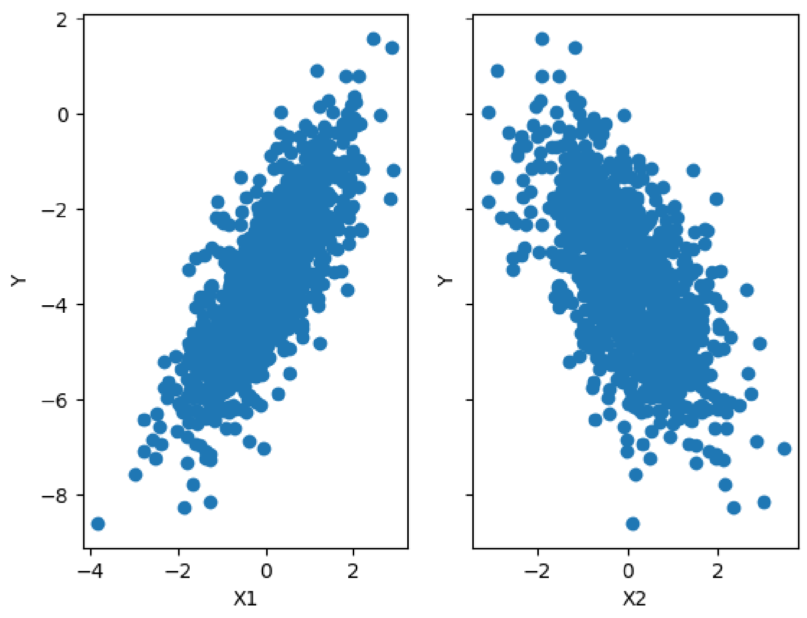


GRADIENTS
PROGRAM:

```
import torch
import torch.nn as nn
import matplotlib.pyplot as plt
# set random seed for reproducibility
torch.manual_seed(42)
# set number of samples
num_samples = 1000
# create random features with 2 dimensions
x = torch.randn(num_samples, 2
# create random weights and bias for the linear regression model
true_weights = torch.tensor([1.3, -1])
true_bias = torch.tensor([-3.5])
# Target variable
```

```
y = x @ true_weights.T + true_bias
# Plot the dataset
fig, ax = plt.subplots(1, 2, sharey=True)
ax[0].scatter(x[:,0],y)
ax[1].scatter(x[:,1],y)
ax[0].set_xlabel('X1')
ax[0].set_ylabel('Y')
ax[1].set_xlabel('X2')
ax[1].set_ylabel('Y')
plt.show()
```

OUTPUT:



RESULT:

Thus the Image Enhancement - Understanding Color spaces, color space conversion, Histogram equalization, Convolution, Image smoothing, Gradients, Edge Detection have been executed successfully and the output got verified

EX.NO: 5
DATE:

Image Features and Image Alignment – Image transforms – Fourier, Hough, Extract ORB Image features, Feature matching, cloning, Feature matching based image alignment

AIM:

To write a program about Image Features and Image Alignment – Image transforms – Fourier, Hough, Extract ORB Image features, Feature matching, cloning, Feature matching based image alignment

ALGORITHM:

- Convert both images to grayscale.
- Match features from the image to be aligned, to the reference image and store the coordinates of the corresponding key points. Keypoints are simply the selected few points that are used to compute the transform (generally points that stand out), and descriptors are histograms of the image gradients to characterize the appearance of a keypoint. In this post, we use ORB (Oriented FAST and Rotated BRIEF) implementation in the OpenCV library, which provides us with both key points as well as their associated descriptors.
- Match the key points between the two images. In this post, we use BFMatcher, which is a brute force matcher. BFMatcher.match() retrieves the best match, while BFMatcher.knnMatch() retrieves top K matches, where K is specified by the user.
- Pick the top matches, and remove the noisy matches.
- Find the homomorphy transform.
- Apply this transform to the original unaligned image to get the output image.

PROGRAM:

```
import cv2
import numpy as np
# Open the image files.
```

```
img1_color = cv2.imread("align.jpg") # Image to be aligned.
img2_color = cv2.imread("ref.jpg") # Reference image.
# Convert to grayscale.
img1 = cv2.cvtColor(img1_color, cv2.COLOR_BGR2GRAY)
img2 = cv2.cvtColor(img2_color, cv2.COLOR_BGR2GRAY)
height, width = img2.shape
# Create ORB detector with 5000 features.
orb_detector = cv2.ORB_create(5000)
# Find keypoints and descriptors.
# The first arg is the image, second arg is the mask
# (which is not required in this case).
kp1, d1 = orb_detector.detectAndCompute(img1, None)
kp2, d2 = orb_detector.detectAndCompute(img2, None)
# Match features between the two images.
# We create a Brute Force matcher with
# Hamming distance as measurement mode.
matcher = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck = True)
# Match the two sets of descriptors.
matches = matcher.match(d1, d2)
# Sort matches on the basis of their Hamming distance.
matches.sort(key = lambda x: x.distance)
# Take the top 90 % matches forward.
matches = matches[:int(len(matches)*0.9)]
no_of_matches = len(matches)
# Define empty matrices of shape no_of_matches * 2.
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
# Find the homography matrix.
homography, mask = cv2.findHomography(p1, p2, cv2.RANSAC)
# Use this matrix to transform the
# colored image wrt the reference image.
transformed_img = cv2.warpPerspective(img1_color,
                                      homography, (width, height))
# Save the output.
cv2.imwrite('output.jpg', transformed_img)
```

OUTPUT:

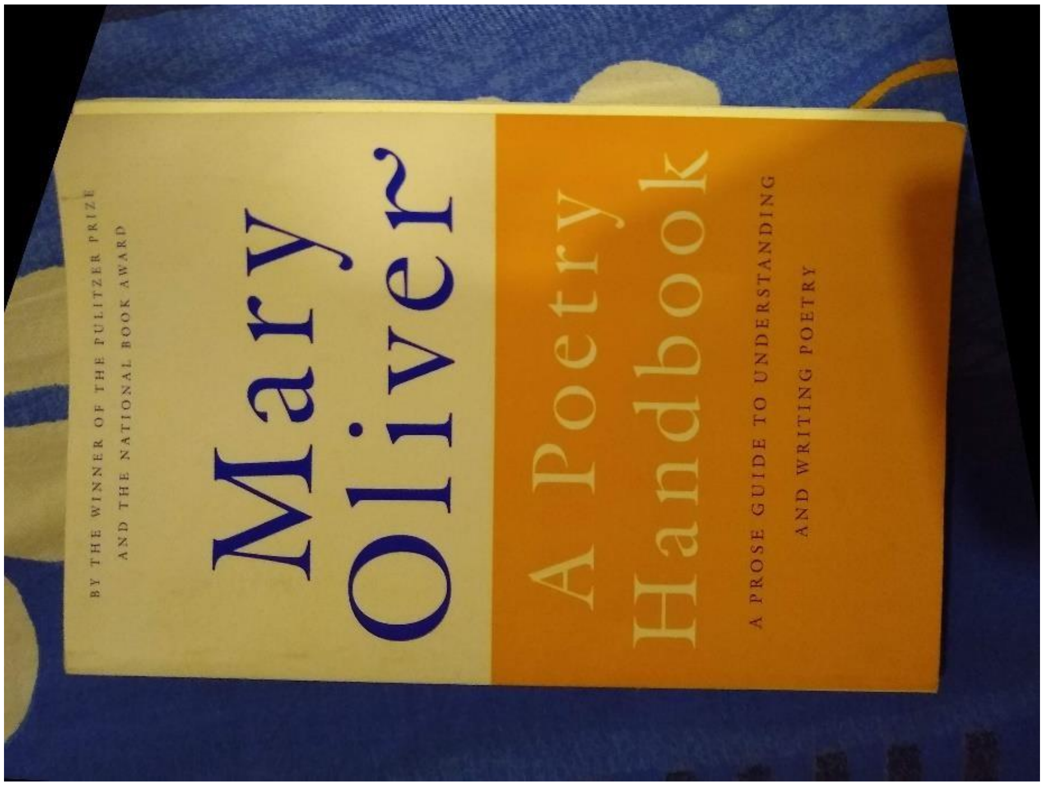
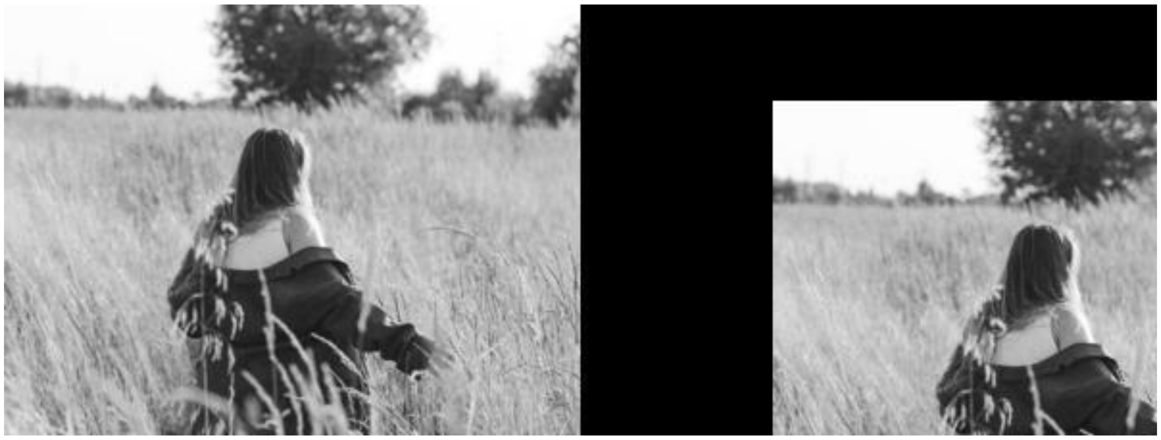

BY THE WINNER OF THE PULITZER PRIZE
AND THE NATIONAL BOOK AWARD
Mary Oliver
A Poetry Handbook
A PROSE GUIDE TO UNDERSTANDING
AND WRITING POETRY

IMAGE TRANSFORMS
PROGRAM:

```
import numpy as np
import cv2 as cv
img = cv.imread('girlImage.jpg', 0)
rows, cols = img.shape
M = np.float32([[1, 0, 100], [0, 1, 50]])
dst = cv.warpAffine(img, M, (cols, rows))
```

```
cv.imshow('img', dst)
cv.waitKey(0)
cv.destroyAllWindows()
```

OUTPUT:



RESULT:

Thus the Image Features and Image Alignment – Image transforms – Fourier, Hough, Extract ORB Image features, Feature matching, cloning, Feature matching based image alignment have been executed successfully and the output got verified

EX.NO: 6 **Image segmentation using Graphcut / Grabcut**
DATE:

AIM:

To write a program about Image segmentation using Graphcut / Grabcut

ALGORITHM:

Step 1: Estimating the color distribution of the foreground and background via a Gaussian Mixture Model (GMM)

Step 2: Constructing a Markov random field over the pixels labels (i.e., foreground vs. background)

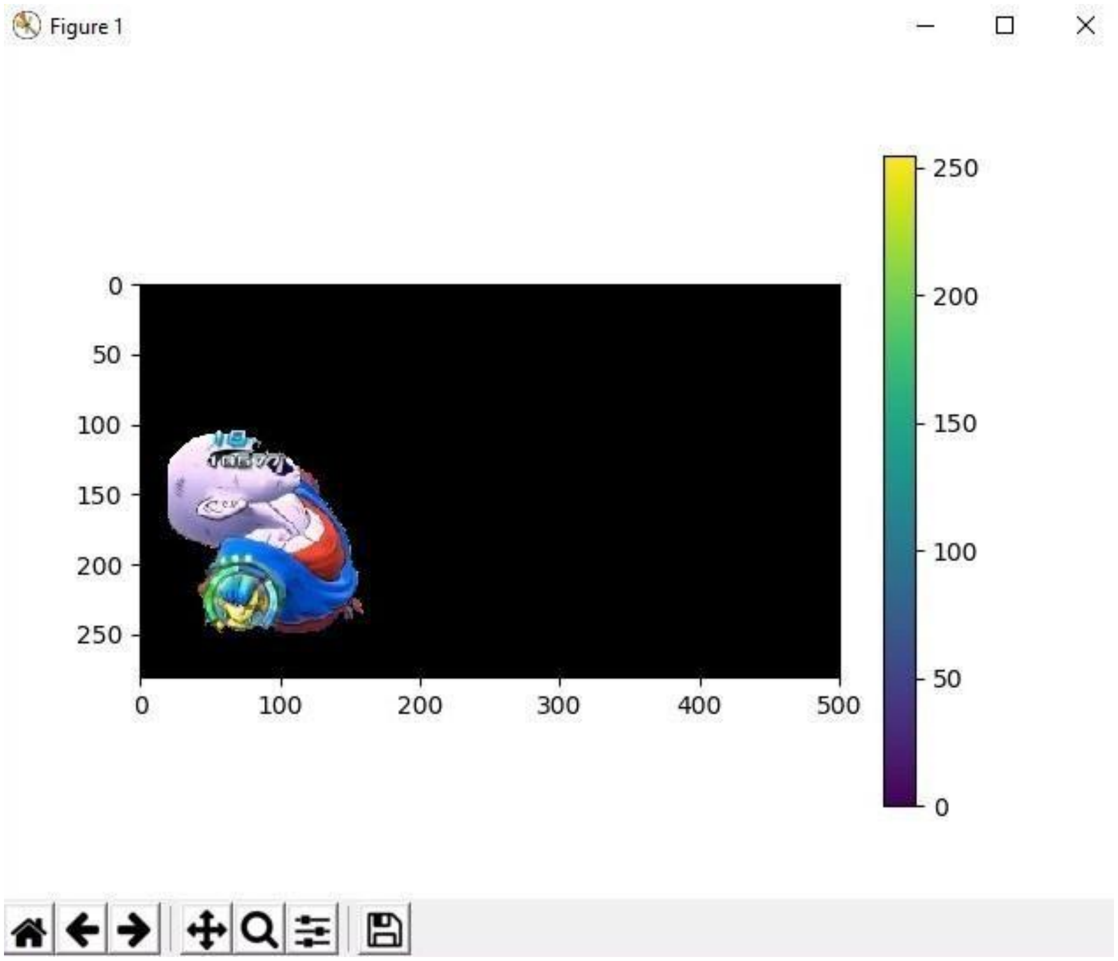
Step 3: Applying a graph cut optimization to arrive at the final segmentation.

PROGRAM:

```
# Python program to illustrate
# foreground extraction using
# GrabCut algorithm
# organize imports
import numpy as np
import cv2
from matplotlib import pyplot as plt
# path to input image specified and
# image is loaded with imread command
image = cv2.imread('image.jpg')
# create a simple mask image similar
# to the loaded image, with the
# shape and return type
mask = np.zeros(image.shape[:2], np.uint8)
# specify the background and foreground model
# using numpy the array is constructed of 1 row
# and 65 columns, and all array elements are 0
# Data type for the array is np.float64 (default)
backgroundModel = np.zeros((1, 65), np.float64)
foregroundModel = np.zeros((1, 65), np.float64)
# define the Region of Interest (ROI)
# as the coordinates of the rectangle
# where the values are entered as
# (startingPoint_x, startingPoint_y, width, height)
# these coordinates are according to the input image
# it may vary for different images
rectangle = (20, 100, 150, 150)
# apply the grabcut algorithm with appropriate
# values as parameters, number of iterations = 3
# cv2.GC_INIT_WITH_RECT is used because
# of the rectangle mode is used
cv2.grabCut(image, mask, rectangle,
                        backgroundModel, foregroundModel,
                        3, cv2.GC_INIT_WITH_RECT)
# In the new mask image, pixels will
# be marked with four flags
# four flags denote the background / foreground
# mask is changed, all the 0 and 2 pixels
# are converted to the background
# mask is changed, all the 1 and 3 pixels
# are now the part of the foreground
# the return type is also mentioned,
# this gives us the final mask
mask2 = np.where((mask == 2)|(mask == 0), 0, 1).astype('uint8')
# The final mask is multiplied with
# the input image to give the segmented image.
image = image * mask2[:, :, np.newaxis]
# output segmented image with colorbar
plt.imshow(image)
```

plt.colorbar()
plt.show()

OUTPUT:



RESULT:

Thus the Image segmentation using Graphcut / Grabcut have been executed successfully and the output got verified

EX.NO: 7 **Camera Calibration with circular grid**
DATE:

AIM:

To write a program about Camera Calibration with circular grid

ALGORITHM:

1. Select a pattern, download (or create your own), and print
2. Mount the pattern onto a rigid flat surface
3. Take many pictures of the target at different orientations and distances
4. Download pictures to compute and select ones that are in focus
5. Use provided examples to automatically detect calibration target and compute parameters
6. Move calibration file to a safe location

PROGRAM:

```
# Import required modules
import cv2
import numpy as np
import os
import glob
# Define the dimensions of checkerboard
CHECKERBOARD = (6, 9)
# stop the iteration when specified
# accuracy, epsilon, is reached or
# specified number of iterations are completed.
criteria = (cv2.TERM_CRITERIA_EPS +
            cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
# Vector for 3D points
threedpoints = []
# Vector for 2D points
twodpoints = []
```

```
# 3D points real world coordinates
objectp3d = np.zeros((1, CHECKERBOARD[0]
                                        * CHECKERBOARD[1],
                                        3), np.float32)
objectp3d[0, :, :2] = np.mgrid[0:CHECKERBOARD[0],

        0:CHECKERBOARD[1]].T.reshape(-1, 2)
prev_img_shape = None

# Extracting path of individual image stored
# in a given directory. Since no path is
# specified, it will take current directory
# jpg files alone
images = glob.glob('*.jpg')
for filename in images:
        image = cv2.imread(filename)
        grayColor = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        # Find the chess board corners
        # If desired number of corners are
        # found in the image then ret = true
        ret, corners = cv2.findChessboardCorners(
                                        grayColor, CHECKERBOARD,
                                        cv2.CALIB_CB_ADAPTIVE_THRESH
                                        + cv2.CALIB_CB_FAST_CHECK +
                                        cv2.CALIB_CB_NORMALIZE_IMAGE)

        # If desired number of corners can be detected then,
        # refine the pixel coordinates and display
        # them on the images of checker board
        if ret == True:
                threedpoints.append(objectp3d)
                # Refining pixel coordinates
                # for given 2d points.
                corners2 = cv2.cornerSubPix(
                        grayColor, corners, (11, 11), (-1, -1), criteria)
                twodpoints.append(corners2)
                # Draw and display the corners
                image = cv2.drawChessboardCorners(image,
                CHECKERBOARD, corners2, ret)
        cv2.imshow('img', image)
        cv2.waitKey(0)
cv2.destroyAllWindows()
h, w = image.shape[:2]
# Perform camera calibration by
# passing the value of above found out 3D points (threedpoints)
# and its corresponding pixel coordinates of the
# detected corners (twodpoints)
ret, matrix, distortion, r_vecs, t_vecs = cv2.calibrateCamera(
        threedpoints, twodpoints, grayColor.shape[::-1], None, None)

# Displaying required output
print(" Camera matrix:")
print(matrix)
print("\n Distortion coefficient:")
print(distortion)
print("\n Rotation Vectors:")
print(r_vecs)
print("\n Translation Vectors:")
print(t_vecs)
```

OUTPUT:

Camera matrix:

```
[[ 36.26378216    0.         125.68539168]
 [  0.          36.76607372 142.49821147]
 [  0.           0.           1.        ]]
```

Distortion coefficient:

```
[[-1.25491812e-03  9.89269357e-05 -2.89077718e-03  4.52760939e-04
  -3.29964245e-06]]
```

Rotation Vectors:

```
[array([[-0.05767492],
       [ 0.03549497],
       [ 1.50906953]]), array([[-0.09301982],
       [-0.01034321],
       [ 3.07733805]]), array([[-0.02175332],
       [ 0.05611105],
       [-0.07308161]])]
```

Translation Vectors:

```
[array([[ 4.63047351],
       [-3.74281386],
       [ 1.64238108]]), array([[2.31648737],
       [3.98801521],
       [1.64584622]]), array([[-3.17548808],
       [-3.46022466],
       [ 1.68200157]])]
```

RESULT:

Thus the Camera Calibration with circular grid have been executed successfully and the output got verified

EX.NO: 8 **Pose Estimation**
DATE:

AIM:

To write a program about Pose Estimation

ALGORITHM:

- keypoint detection and estimation.
- Detection identifies the presence of a human in an image.
- keypoint estimation involves determining the coordinates of specific body joints

PROGRAM:

```
# Necessary imports
%tensorflow_version 1.x
!pip3 install scipy pyyaml ipykernel opencv-python==3.4.5.20
# Clone some Code from GitHub
!git clone https://www.github.com/rwightman/posenet-python
import os
import cv2
import time
import argparse
import posenet
import tensorflow as tf
import matplotlib.pyplot as plt
print('Initializing')
input_file = '/content/posenet-python/video.avi'
output_file = '/content/posenet-python/output.mp4'
# Load input video files and
cap = cv2.VideoCapture(input_file)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = cap.get(cv2.CAP_PROP_FPS)
```

```
# create a video writer to write the output file
fourcc = cv2.VideoWriter_fourcc('M','J','P','G')
video = cv2.VideoWriter(output_file, fourcc, fps, (width, height))
model = 101
###scale_factor = 1.0
scale_factor = 0.4

with tf.Session() as sess:
        # Load PoseNet model
        model_cfg, model_outputs = posenet.load_model(model, sess)
        output_stride = model_cfg['output_stride']
        start = time.time()
        incnt = 0
        # Process the whole video frame by frame
        while True:
                # Increase frame count by one
                incnt = incnt + 1
                try:
                # read_cap is utility function to read and process from video
                input_image, draw_image, output_scale = posenet.read_cap(
                                cap, scale_factor=scale_factor,
output_stride=output_stride)
                except:
                break
                # run the model on the image and generate output results
                heatmaps_result, offsets_result, displacement_fwd_result,
displacement_bwd_result = sess.run(
                        model_outputs,
                        feed_dict={'image:0': input_image}
                )
                # here we filter poses generated by above model
                # and output pose score, keypoint scores and their keypoint
coordinates
```

```
            # this function will return maximum 10 pose, it can be changed
by maximum_pose
            # variable.
            pose_scores, keypoint_scores, keypoint_coords =
posenet.decode_multiple_poses(
                heatmaps_result.squeeze(axis=0),
                offsets_result.squeeze(axis=0),
                displacement_fwd_result.squeeze(axis=0),
                displacement_bwd_result.squeeze(axis=0),
                output_stride=output_stride,
                min_pose_score=0.25)
            # scale keypoint co-ordinate to output scale
            keypoint_coords *= output_scale
            # draw pose on input frame to obtain output frame
            draw_image = posenet.draw_skel_and_kp(
                    draw_image, pose_scores, keypoint_scores,
keypoint_coords,
                    min_pose_score=0.25, min_part_score=0.25)
            video.write(draw_image)
# release the videoreader and writer
video.release()
cap.release()
```

OUTPUT:

RESULT:

Thus the Pose Estimation have been executed successfully and the output got verified