Program 1:

Using OpenCV in python read and display an image

**import** cv2

img **=** cv2.imread("geeksforgeeks.png", cv2.IMREAD\_COLOR)

cv2.imshow("image", img)

cv2.waitKey(0)

cv2.destroyAllWindows()

img.shape

**Matplotlib library uses RGB color format to read a colored image. Here we are demonstrating an example of reading an image using this library.**

**import** numpy as np

**import** matplotlib.pyplot as plt

img**=**cv2.imread("geeks.png")

#Displaying image using plt.imshow() method

plt.imshow(img)

**See the difference in colors of images read by cv2 and matplotlib library. Because cv2 uses BGR color format and matplotlib uses RGB color format. To convert BGR to RGB, we us a function**

**import** numpy as np

**import** matplotlib.pyplot as plt

img**=**cv2.imread("geeks.png")

# Converting BGR color to RGB color format

RGB\_img **=** cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

#Displaying image using plt.imshow() method

plt.imshow(img)

program 2

Opening in grayscale mode

**import** cv2

# path

path **=** r'geeksforgeeks.png'

# Using cv2.imread() method

# Using 0 to read image in grayscale mode

img **=** cv2.imread(path, 0)

# Displaying the image

cv2.imshow('image', img)

cv2.waitKey(0)

cv2.destroyAllWindows()

|  |
| --- |
|  |

Program 3

method is used to display an image in a window.

# importing cv2

**import** cv2

  # path

path **=** r'C:\Users\Rajnish\Desktop\geeksforgeeks.png'

  # Reading an image in default mode

image **=** cv2.imread(path)

  # Window name in which image is displayed

window\_name **=** 'image'

  # Using cv2.imshow() method

# Displaying the image

cv2.imshow(window\_name, image)

  #waits for user to press any key

#(this is necessary to avoid Python kernel form crashing)

cv2.waitKey(0)

  #closing all open windows

cv2.destroyAllWindows()

example 2

# importing cv2

**import** cv2

 # path

path **=** r'C:\Users\Rajnish\Desktop\geeksforgeeks.png'

  # Reading an image in grayscale mode

image **=** cv2.imread(path, 0)

  # Window name in which image is displayed

window\_name **=** 'image'

  # Using cv2.imshow() method

# Displaying the image

cv2.imshow(window\_name, image)

  # waits for user to press any key

# (this is necessary to avoid Python kernel form crashing)

cv2.waitKey(0)

# closing all open windows

cv2.destroyAllWindows()

program 4

**Python OpenCV | cv2.copyMakeBorder() method**

**OpenCV-Python** is a library of Python bindings designed to solve computer vision problems. cv2.copyMakeBorder() method is used to create a border around the image like a photo frame. 

# importing cv2

**import** cv2

  # path

path **=** r'C:\Users\Rajnish\Desktop\geeksforgeeks\geeks.png'

  # Reading an image in default mode

image **=** cv2.imread(path)

  # Window name in which image is displayed

window\_name **=** 'Image'

 # Using cv2.copyMakeBorder() method

image **=** cv2.copyMakeBorder(image, 10, 10, 10, 10, cv2.BORDER\_CONSTANT, None, value **=** 0)

 # Displaying the image

cv2.imshow(window\_name, image)

# program 5

# Python | Image blurring using OpenCV

**import** cv2

**import** numpy as np

image **=** cv2.imread('[C://Geeksforgeeks//image\_processing//fruits.jpg](c://Geeksforgeeks/image_processing/fruits.jpg)')

cv2.imshow('Original Image', image)

cv2.waitKey(0)

# Gaussian Blur

Gaussian **=** cv2.GaussianBlur(image, (7, 7), 0)

cv2.imshow('Gaussian Blurring', Gaussian)

cv2.waitKey(0)

# Median Blur

median **=** cv2.medianBlur(image, 5)

cv2.imshow('Median Blurring', median)

cv2.waitKey(0)

# Bilateral Blur

bilateral **=** cv2.bilateralFilter(image, 9, 75, 75)

cv2.imshow('Bilateral Blurring', bilateral)

cv2.waitKey(0)

cv2.destroyAllWindows()

# program 6

# Image Processing in Python Scaling, Rotating

**import** cv2

**import** numpy as np

FILE\_NAME **=** 'volleyball.jpg'

**try**:

    # Read image from disk.

    img **=** cv2.imread(FILE\_NAME)

    # Get number of pixel horizontally and vertically.

    (height, width) **=** img.shape[:2]

    # Specify the size of image along with interploation methods.

    # cv2.INTER\_AREA is used for shrinking, whereas cv2.INTER\_CUBIC

    # is used for zooming.

    res **=** cv2.resize(img, (int(width **/** 2), int(height **/** 2)), interpolation **=** cv2.INTER\_CUBIC)

    # Write image back to disk.

    cv2.imwrite('result.jpg', res)

**except** IOError:

    print ('Error while reading files !!!')

**Rotating an image :-**  
Images can be rotated to any degree clockwise or otherwise. We just need to define rotation matrix listing rotation point, degree of rotation and the scaling factor.

|  |
| --- |
| **import** cv2  **import** numpy as np    FILE\_NAME **=** 'volleyball.jpg'  **try**:      # Read image from the disk.      img **=** cv2.imread(FILE\_NAME)        # Shape of image in terms of pixels.      (rows, cols) **=** img.shape[:2]        # getRotationMatrix2D creates a matrix needed for transformation.      # We want matrix for rotation w.r.t center to 45 degree without scaling.      M **=** cv2.getRotationMatrix2D((cols **/** 2, rows **/** 2), 45, 1)      res **=** cv2.warpAffine(img, M, (cols, rows))        # Write image back to disk.      cv2.imwrite('result.jpg', res)  **except** IOError:      print ('Error while reading files !!!') |

# program 7

Image Processing in Python Shifting and Edge Detection

**Translating an Image :-**  
Translating an image means shifting it within a given frame of reference.

|  |
| --- |
| **import** cv2  **import** numpy as np    FILE\_NAME **=** 'volleyball.jpg'  # Create translation matrix.  # If the shift is (x, y) then matrix would be  # M = [1 0 x]  #     [0 1 y]  # Let's shift by (100, 50).  M **=** np.float32([[1, 0, 100], [0, 1, 50]])    **try**:        # Read image from disk.      img **=** cv2.imread(FILE\_NAME)      (rows, cols) **=** img.shape[:2]        # warpAffine does appropriate shifting given the      # translation matrix.      res **=** cv2.warpAffine(img, M, (cols, rows))        # Write image back to disk.      cv2.imwrite('result.jpg', res)    **except** IOError:      print ('Error while reading files !!!') |

**Output:**  


**Edge detection in an Image :-**  
The process of image detection involves detecting sharp edges in the image. This edge detection is essential in context of image recognition or [object localization/detection](https://en.wikipedia.org/wiki/Object_detection). There are several algorithms for detecting edges due to it’s wide applicability. We’ll be using one such algorithm known as [Canny Edge Detection](https://en.wikipedia.org/wiki/Canny_edge_detector).

|  |
| --- |
| **import** cv2  **import** numpy as np    FILE\_NAME **=** 'volleyball.jpg'  **try**:      # Read image from disk.      img **=** cv2.imread(FILE\_NAME)        # Canny edge detection.      edges **=** cv2.Canny(img, 100, 200)        # Write image back to disk.      cv2.imwrite('result.jpg', edges)  **except** IOError:      print ('Error while reading files !!!') |

# Program 8

# Erosion and Dilation of images using OpenCV in python

# Python program to demonstrate erosion and

# dilation of images.

**import** cv2

**import** numpy as np

# Reading the input image

img **=** cv2.imread('input.png', 0)

# Taking a matrix of size 5 as the kernel

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

# The first parameter is the original image,

# kernel is the matrix with which image is

# convolved and third parameter is the number

# of iterations, which will determine how much

# you want to erode/dilate a given image.

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

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

cv2.imshow('Input', img)

cv2.imshow('Erosion', img\_erosion)

cv2.imshow('Dilation', img\_dilation)

cv2.waitKey(0)

program 9

Below is the Python code explaining Opening Morphological Operation –   
 Python program to illustrate

# Opening morphological operation

# on an image

# organizing imports

**import** cv2

**import** numpy as np

# return video from the first webcam on your computer.

screenRead **=** cv2.VideoCapture(0)

# loop runs if capturing has been initialized.

**while**(1):

    # reads frames from a camera

    \_, image **=** screenRead.read()

    # Converts to HSV color space, OCV reads colors as BGR

    # frame is converted to hsv

    hsv **=** cv2.cvtColor(image, cv2.COLOR\_BGR2HSV)

    # defining the range of masking

    blue1 **=** np.array([110, 50, 50])

    blue2 **=** np.array([130, 255, 255])

    # initializing the mask to be

    # convoluted over input image

    mask **=** cv2.inRange(hsv, blue1, blue2)

 # passing the bitwise\_and over

    # each pixel convoluted

    res **=** cv2.bitwise\_and(image, image, mask **=** mask)

    # defining the kernel i.e. Structuring element

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

    # defining the opening function

    # over the image and structuring element

    opening **=** cv2.morphologyEx(mask, cv2.MORPH\_OPEN, kernel)

    # The mask and opening operation

    # is shown in the window

    cv2.imshow('Mask', mask)

    cv2.imshow('Opening', opening)

    # Wait for 'a' key to stop the program

**if** cv2.waitKey(1) & 0xFF **==** ord('a'):

**break**

# De-allocate any associated memory usage

cv2.destroyAllWindows()

# Close the window / Release webcam

screenRead.release()

program 9

Below is the code for finding circles using OpenCV on the above input image.

**import** cv2

**import** numpy as np

# Read image.

img **=** cv2.imread('eyes.jpg', cv2.IMREAD\_COLOR)

# Convert to grayscale.

gray **=** cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

# Blur using 3 \* 3 kernel.

gray\_blurred **=** cv2.blur(gray, (3, 3))

# Apply Hough transform on the blurred image.

detected\_circles **=** cv2.HoughCircles(gray\_blurred,

                   cv2.HOUGH\_GRADIENT, 1, 20, param1 **=** 50,

               param2 **=** 30, minRadius **=** 1, maxRadius **=** 40)

# Draw circles that are detected.

**if** detected\_circles **is** **not** None:

    # Convert the circle parameters a, b and r to integers.

    detected\_circles **=** np.uint16(np.around(detected\_circles))

**for** pt **in** detected\_circles[0, :]:

        a, b, r **=** pt[0], pt[1], pt[2]

        # Draw the circumference of the circle.

        cv2.circle(img, (a, b), r, (0, 255, 0), 2)

        # Draw a small circle (of radius 1) to show the center.

        cv2.circle(img, (a, b), 1, (0, 0, 255), 3)

        cv2.imshow("Detected Circle", img)

        cv2.waitKey(0)

|  |
| --- |
|  |