|  |  |
| --- | --- |
| EXNO: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 execute the

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

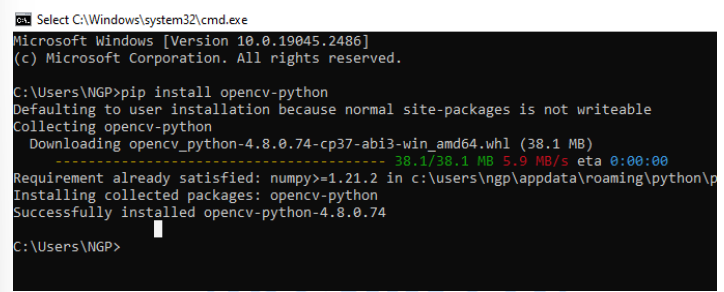
Step9: To check if OpenCV is correctly installed, just run the following commands to perform a version

check: Python

import cv2

print(cv2.\_\_version\_\_)

**OUTPUT:**



**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.

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.

**Program**:

import cv2

import matplotlib.pyplot as plt

import numpy as np

image = cv2.imread('image.jpg')

image2 = cv2.imread('image4.png')

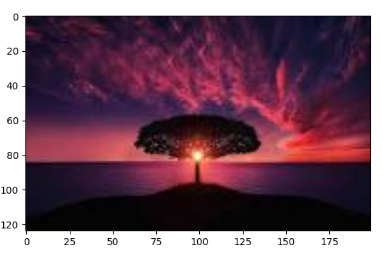
image3=cv2.imread('images2.jpg')

**Load The Image:-**

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

plt.axis('on')

plt.show()



**Cropped Image**:

x = 40

y = 75

width = 200

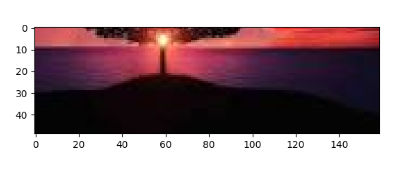
height = 200

cropped\_image = image[y:y+height, x:x+width]

plt.imshow(cv2.cvtColor(cropped\_image, cv2.COLOR\_BGR2RGB))

plt.axis('on')

plt.show()



**Resized Image:**

new\_width = 300

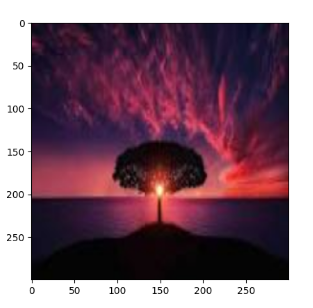
new\_height = 300

resized\_image = cv2.resize(image, (new\_width, new\_height))

plt.imshow(cv2.cvtColor(resized\_image, cv2.COLOR\_BGR2RGB))

plt.axis('on')

plt.show()



**Thresholding and contour analysis:**

gray\_image = cv2.cvtColor(image3, cv2.COLOR\_BGR2GRAY)

\_, thresholded\_image = cv2.threshold(gray\_image, 127, 255, cv2.THRESH\_BINARY)

plt.imshow(thresholded\_image, cmap='gray')

plt.axis('off')

plt.show()



**Contour analysis:-**

contours, \_ = cv2.findContours(thresholded\_image, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

cv2.drawContours(image3, contours, -1, (0, 255, 0), 2)

plt.imshow(cv2.cvtColor(image3, cv2.COLOR\_BGR2RGB))

plt.axis('on')

plt.show()



**Blob Detection:**

params = cv2.SimpleBlobDetector\_Params()

blob\_detector = cv2.SimpleBlobDetector\_create(params)

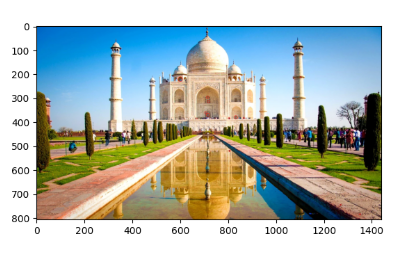
keypoints = blob\_detector.detect(image2)

blob\_image = cv2.drawKeypoints(image2, keypoints, np.array([]), (0, 0, 255), cv2.DRAW\_MATCHES\_FLAGS\_DRAW\_RICH\_KEYPOINTS)

plt.imshow(cv2.cvtColor(blob\_image, cv2.COLOR\_BGR2RGB))

plt.axis('on')

plt.show()



**Result:**

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

|  |  |
| --- | --- |
| EX NO: 3 | **Image Annotation – Drawing lines, text circles, rectangle, ellipse on image** |
| DATE : |

**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**:

import cv2

import matplotlib.pyplot as plt

# Read the original image

img = cv2.imread("images.jpeg")

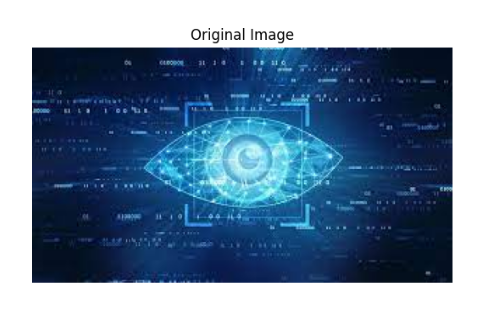
**# Display the original image**

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

plt.axis('off')

plt.title("Original Image")

plt.show()



**# Line Image:**

imageLine = img.copy()

pointA = (50, 50)

pointB = (100, 50)

cv2.line(imageLine, pointA, pointB, (255, 255, 0), thickness=3)

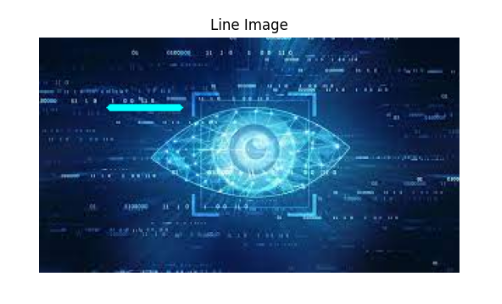
# Display the line image

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

plt.axis('off')

plt.title("Line Image")

plt.show()



**# Circle Image:**

imageCircle = img.copy()

circle\_center = (50, 50)

radius = 50

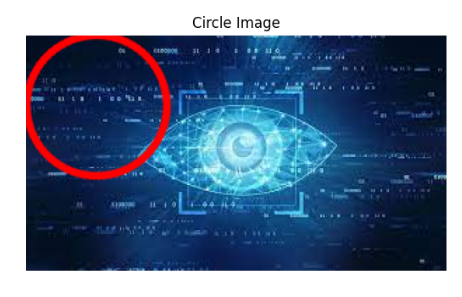
cv2.circle(imageCircle, circle\_center, radius, (0, 0, 255), thickness=3, lineType=cv2.LINE\_AA)

# Display the circle image

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

plt.axis('off')

plt.title("Circle Image")

plt.show()

**# Rectangle Image:**

height, width, \_ = img.shape

rect\_width = 100

rect\_height = 50

x = (width - rect\_width) // 2

y = (height - rect\_height) // 2

start\_point = (x, y)

end\_point = (x + rect\_width, y + rect\_height)

color = (0, 0, 255)

thickness = 2

annotated\_image = cv2.rectangle(img.copy(), start\_point, end\_point, color, thickness)

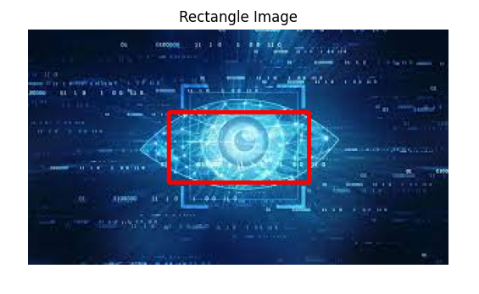
# Display the rectangle image

plt.imshow(cv2.cvtColor(annotated\_image, cv2.COLOR\_BGR2RGB))

plt.axis('off')

plt.title("Rectangle Image")

plt.show()



**# Ellipse Image:**

center\_x = width // 2

center\_y = height // 2

axis1\_length = 100

axis2\_length = 50

color = (0, 0, 255)

thickness = 2

annotated\_image = cv2.ellipse(img.copy(), (center\_x, center\_y), (axis1\_length, axis2\_length), 0, 0, 360, color, thickness)

# Display the ellipse image

plt.imshow(cv2.cvtColor(annotated\_image, cv2.COLOR\_BGR2RGB))

plt.axis('off')

plt.title("Ellipse Image")

plt.show()



**Result:**

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

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

**Aim:**

To write a python code to perform Image Enhancement - Understanding Color spaces, color space Conversion , Histogram equalization, Convolution, Image smoothing, Gradients, Edge Detection.

**Algorithm:**

Step1: Import Necessary packages.

Step2: Load the image using cv2.imread()

Step3: Convert image to grayscale.

Step4: Convert image to HSV.

Step5: Perform histogram equalization.

Step6: Perform Convolution (Edge Detection).

Step7: Perform Image Smoothing.

Step8: Perform Gradients and Edge Detection.

Step9: display all the images.

Step10: Stop the program.

**Program**:

import cv2

from google.colab.patches import cv2\_imshow

import numpy as np

# Load the image

image = cv2.imread('/content/drive/MyDrive/Images/ball.jpg')

# Convert the image to grayscale

gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Convert the image to HSV color space

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

# Apply histogram equalization to the grayscale image

equalized\_image = cv2.equalizeHist(gray\_image)

# Define a kernel for convolution

kernel = np.array([[0, -1, 0], [-1, 4, -1], [0, -1, 0]])

# Apply convolution using the defined kernel

convolved\_image = cv2.filter2D(image, -1, kernel)

# Apply Gaussian blur to the image

blurred\_image = cv2.GaussianBlur(image, (5, 5), 0)

# Apply Sobel edge detection in the x and y directions

sobel\_x = cv2.Sobel(gray\_image, cv2.CV\_64F, 1, 0, ksize=3)

sobel\_y = cv2.Sobel(gray\_image, cv2.CV\_64F, 0, 1, ksize=3)

# Calculate gradient magnitude using Sobel derivatives

gradient\_magnitude = cv2.magnitude(sobel\_x, sobel\_y)

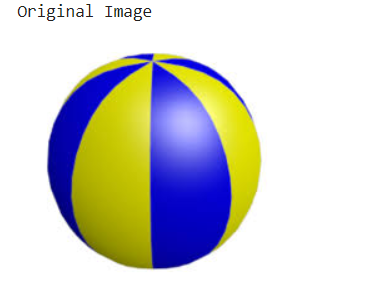
# Apply Canny edge detection

canny\_edges = cv2.Canny(gray\_image, threshold1=100, threshold2=200)

# Display the results

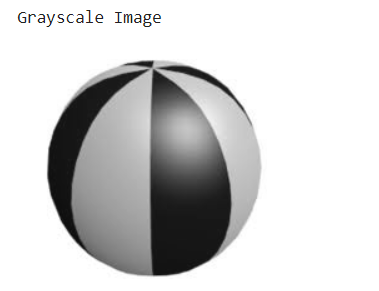
print("Original Image")

cv2\_imshow(image)



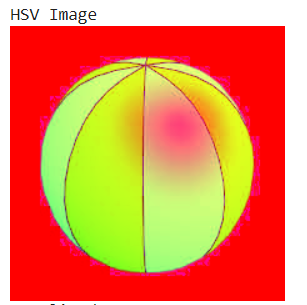
print("Grayscale Image")

cv2\_imshow(gray\_image)



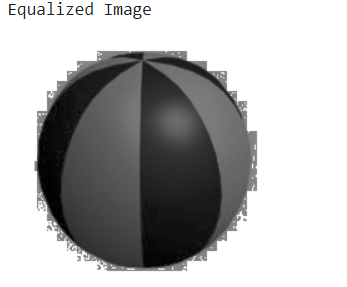
print("HSV Image")

cv2\_imshow(hsv\_image)



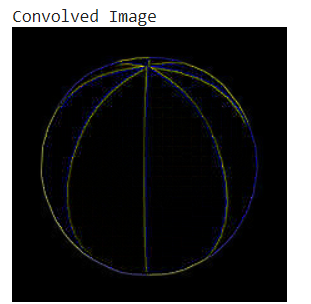
print("Equalized Image")

cv2\_imshow(equalized\_image)



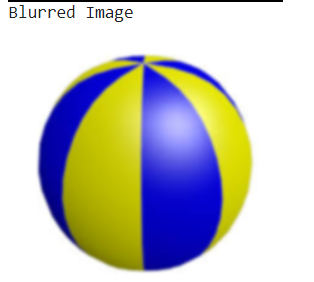
print("Convolved Image")

cv2\_imshow(convolved\_image)



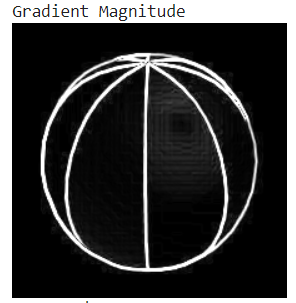
print("Blurred Image")

cv2\_imshow(blurred\_image)



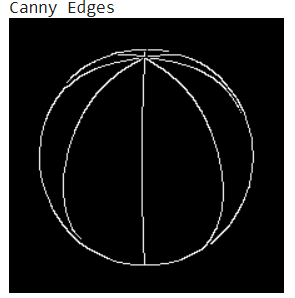
print("Gradient Magnitude")

cv2\_imshow(gradient\_magnitude)



print("Canny Edges")

cv2\_imshow(canny\_edges)



# Wait for a key event and close all windows

cv2.waitKey(0)

cv2.destroyAllWindows()

**Result:**

Thus, the python program to perform Image Enhancement - Understanding Color spaces, color space Conversion , Histogram equalization, Convolution, Image smoothing, Gradients, Edge Detection was verified and executed successfully.

|  |  |
| --- | --- |
| 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 ,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

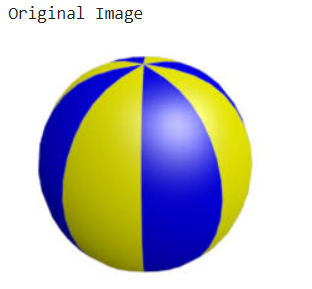
# Load the image

image = cv2.imread("/content/drive/MyDrive/Images/ball.jpg")

# Display the original image

print('\nOriginal Image')

cv2\_imshow(image)



# Convert the image to grayscale

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Compute the Fourier Transform

f\_transform = np.fft.fft2(gray)

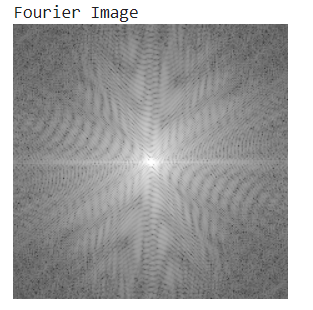
f\_shift = np.fft.fftshift(f\_transform)

magnitude\_spectrum = 20 \* np.log(np.abs(f\_shift))

# Display the Fourier Transform magnitude spectrum

print('\nFourier Image')

cv2\_imshow(magnitude\_spectrum)



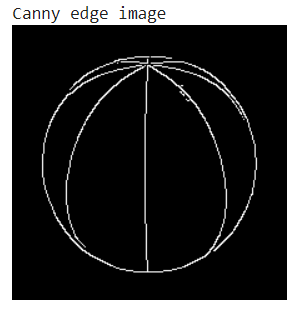
# Apply Canny edge detection

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

# Display the Canny edge image

print('\nCanny edge image')

cv2\_imshow(edges)



# Create an ORB detector

orb = cv2.ORB\_create()

# Detect keypoints and compute descriptors

keypoints, descriptors = orb.detectAndCompute(gray, None)

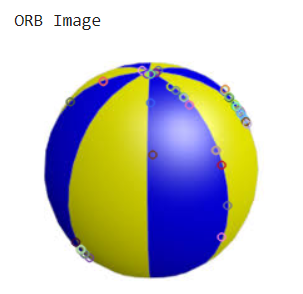
# Draw keypoints on the image

keypoint\_image = cv2.drawKeypoints(image, keypoints, None)

# Display the ORB keypoints image

print('\nORB Image')

cv2\_imshow(keypoint\_image)



# Create a Brute-Force Matcher

bf = cv2.BFMatcher(cv2.NORM\_HAMMING, crossCheck=True)

# Match descriptors

matches = bf.match(descriptors, descriptors)

matches = sorted(matches, key=lambda x: x.distance)

# Draw matched features

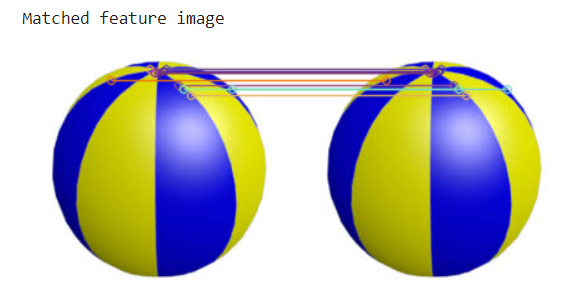
matched\_image = cv2.drawMatches(image, keypoints, image, keypoints, matches[:10], None,

flags=cv2.DrawMatchesFlags\_NOT\_DRAW\_SINGLE\_POINTS)

# Display the matched feature image

print('\nMatched feature image')

cv2\_imshow(matched\_image)



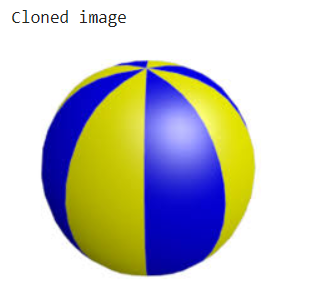
# Create a clone of the original image

cloned\_image = image.copy()

# Display the cloned image

print('\nCloned image')

cv2\_imshow(cloned\_image)



**Result:**

Thus the program for image features and image alignment was executed and the output

verified successfully