7. Write a Program to read a digital image. Split and display image into 4 quadrants, up, down, right and left.

```
import cv2

# Load the image
image = cv2.imread('test.jpg')

# Get image height and width
height, width = image.shape[:2]

# Split the image into four quadrants
top_left = image[0:height//2, 0:width//2]
top_right = image[0:height//2, width//2:width]
bottom_left = image[height//2:height, 0:width//2]
bottom_right = image[height//2:height, width//2:width]

# Display the original image and the quadrants
cv2.imshow('Original Image', image)
cv2.imshow('Top Left Quadrant', top_left)
cv2.imshow('Top Right Quadrant', top_right)
cv2.imshow('Bottom Left Quadrant', bottom_left)
cv2.imshow('Bottom Right Quadrant', bottom_right)

# Wait for a key press and close all windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

8. Write a program to show rotation, scaling, and translation on an image.

```
import cv2
import numpy as np

# Load the image
image = cv2.imread('test.jpg')

# Define rotation angle (in degrees)
angle = 45

# Define scaling factors
scale_x = 1.5
scale_y = 1.5

# Define translation offsets
tx = 50
ty = 50

# Get image dimensions
height, width = image.shape[:2]

# Define rotation matrix
rotation_matrix = cv2.getRotationMatrix2D((width/2, height/2), angle,
1)

# Apply rotation
rotated_image = cv2.warpAffine(image, rotation_matrix, (width,
height))

# Apply scaling
```

```
scaled_image = cv2.resize(image, None, fx=scale_x, fy=scale_y)


# Apply translation
translation_matrix = np.float32([[1, 0, tx], [0, 1, ty]])
translated_image = cv2.warpAffine(image, translation_matrix, (width,
height))


# Display images
cv2.imshow('Original Image', image)
cv2.imshow('Rotated Image', rotated_image)
cv2.imshow('Scaled Image', scaled_image)
cv2.imshow('Translated Image', translated_image)


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

9. Read an image and extract and display low-level features such as edges, textures using filtering techniques.

```
import cv2
import numpy as np


# Load the image
image = cv2.imread('test.jpg', cv2.IMREAD_GRAYSCALE)


# Apply edge detection using Canny
edges = cv2.Canny(image, 100, 200)


# Apply texture analysis using Laplacian of Gaussian (LoG)
image_blur = cv2.GaussianBlur(image, (3, 3), 0)
```

```
image_log = cv2.Laplacian(image_blur, cv2.CV_64F)
image_log = np.uint8(np.absolute(image_log))

# Display images
cv2.imshow('Original Image', image)
cv2.imshow('Edges', edges)
cv2.imshow('Texture (LoG)', image_log)

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

0. Write a program to blur and smoothing an image.

```
import cv2

# Load the image
image = cv2.imread('test.jpg')

# Apply Gaussian blur
gaussian_blur = cv2.GaussianBlur(image, (15, 15), 0)

# Apply median blur
median_blur = cv2.medianBlur(image, 15)

# Apply bilateral filter
bilateral_blur = cv2.bilateralFilter(image, 15, 75, 75)

# Display images
cv2.imshow('Original Image', image)
cv2.imshow('Gaussian Blur', gaussian_blur)
cv2.imshow('Median Blur', median_blur)
cv2.imshow('Bilateral Blur', bilateral_blur)

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

11. Write a program to contour an image.

```
import cv2

# Read the image
```

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

# Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

_, thresh = cv2.threshold(gray_image, 0, 255,
cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

# Find contours in the threshold image
contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

# Draw contours on the original image
contour_image = image.copy()

# Draw all contours with green color and thickness 2
cv2.drawContours(contour_image, contours, -1, (0, 255, 0), 2)

# Display the original image with contours
cv2.imshow("Image with Contours", contour_image)

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

12. Write a program to detect a face/s in an image.

```
import cv2

# Load the pre-trained Haar Cascade classifier for face detection
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

# Load the image
image = cv2.imread('sample.jpg')

# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Detect faces in the image
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1,
minNeighbors=6, minSize=(30, 30))

# Draw rectangles around the detected faces
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)
```

```
# Display the image with detected faces
cv2.imshow('Faces Detected', image)

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