Experiment 4

Geometric Transformation and Filtering Using OpenCV

# Aim:

To implement the following Geometric Transformations a Filtering functions on an image in Open CV:

1. Translation, Rotation, Affine Transformation and Perspective Transformation
2. 2D convolution, Averaging and Blurring
3. Thresholding

# Software/ Packages Used:

1. Pycharm IDE
2. Libraries used:
   * NumPy
   * opencv-python
   * matplotlib
   * scipy

# Programs:

**1)Geometric Transformations**

#Translation #Rotation

# Affine Transformation #Perspective Transformation **2)Filtering**

#Image Blurring #Gausian Blurring #Median Blurring

#Bilateral Filtering #Average Blurring **3)Thresholding** #Simple #Adaptive

#Ostu’s

## 1)Geometric Transformation (Output):

******

(Input Image- Colour)





(Output Images- Colour)

## 1)Geometric Transformation (Code):

import cv2  
import numpy as np  
  
def geometric\_transformations(image\_path):  
 img = cv2.imread(image\_path)  
 rows, cols = img.shape[:2]  
  
 # Translation  
 translation\_matrix = np.float32([[1, 0, 50], [0, 1, 30]]) # Shift by (50, 30)  
 translated\_img = cv2.warpAffine(img, translation\_matrix, (cols, rows))  
  
 # Rotation  
 rotation\_matrix = cv2.getRotationMatrix2D((cols / 2, rows / 2), 45, 1) # Rotateby 45 degrees  
 rotated\_img = cv2.warpAffine(img, rotation\_matrix, (cols, rows))  
  
 # Affine Transformation (combination of translation, rotation, scaling, and shearing)  
 affine\_matrix = np.float32([[0.5, 0.5, 50], [-0.5, 0.5, 30]]) # Custom affine matrix  
 affine\_transformed\_img = cv2.warpAffine(img, affine\_matrix, (cols, rows))

# Perspective Transformation  
 perspective\_matrix = np.float32([[0.5, 0.5, 50], [-0.5, 0.5, 30], [0, 0, 1]])  
 perspective\_transformed\_img = cv2.warpPerspective(img, perspective\_matrix,(cols, rows))

cv2.imshow('Original Image', img)  
 cv2.imshow('Translated Image', translated\_img)  
 cv2.imshow('Rotated Image', rotated\_img)  
 cv2.imshow('Affine Transformed Image', affine\_transformed\_img)  
 cv2.imshow('Perspective Transformed Image', perspective\_transformed\_img)  
  
 cv2.waitKey(0)  
 cv2.destroyAllWindows()

#Usage  
geometric\_transformations(r"C:\Users\RAGHUL\Downloads\krishna.jpg")

## Image Blurring (Output):

******

(Input Image-Colour) (Output image-Colour)

## Gausian Blurring (Output):

******

(Input Image-Colour) (Output image-Colour)

## Image Blurring (Code):

import cv2  
image = cv2.imread(r"C:\Users\RAGHUL\Downloads\krishna.jpg")  
average\_blurred\_image = cv2.blur(image, (5, 5))  
cv2.imshow("Original image",image )  
cv2.imshow("Average Blurred Image", average\_blurred\_image)  
cv2.waitKey(0)  
cv2.destroyAllWindows()

## Gausian Blurring (Code):

import cv2  
image = cv2.imread(r"C:\Users\RAGHUL\Downloads\krishna.jpg")  
blurred\_image = cv2.GaussianBlur(image, (5,5), 0)  
cv2.imshow("Original image",image )  
cv2.imshow("Gaussian Blurred Image", blurred\_image)  
cv2.waitKey(0)  
cv2.destroyAllWindows()

## Bilateral Blurring (Output):

******

******

(Input Image-Colour) (Output image-Colour)

## Median Blurring (Output):

******

(Input Image-Colour) (Output image-Colour)

## Bilateral Blurring (Code):

import cv2  
image = cv2.imread(r"C:\Users\RAGHUL\Downloads\krishna.jpg")  
bilateral\_filtered\_image = cv2.bilateralFilter(image, 9, 175, 715)  
cv2.imshow("Original image",image )  
cv2.imshow("Bilateral Filtered Image", bilateral\_filtered\_image)  
cv2.waitKey(0)  
cv2.destroyAllWindows()

## Median Blurring (Code):

import cv2

image = cv2.imread (r"C:\Users\RAGHUL\Downloads\krishna.jpg")

median\_blurred\_image = cv2.medianBlur(image, 25)

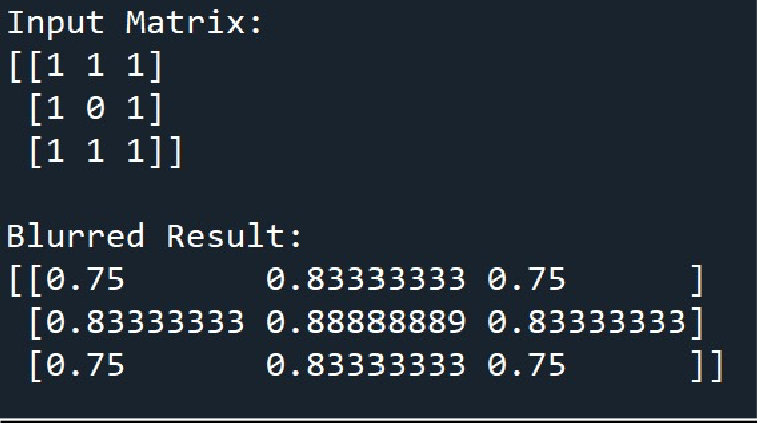
cv2.imshow("Original image",image )

cv2.imshow("Median Blurred Image", median\_blurred\_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

## User Defined Average Blurring (Output):



(Output in Terminal )

## User Defined Average Blurring (Code):

import numpy as np

def average\_blurring(input\_matrix, kernel\_size): rows, cols = input\_matrix.shape

blurred\_matrix = np.zeros\_like(input\_matrix, dtype=float) kernel\_radius = kernel\_size // 2

for i in range(rows): for j in range(cols):

pixel\_sum = 0.0

count = 0

for m in range(-kernel\_radius, kernel\_radius + 1): for n in range(-kernel\_radius, kernel\_radius + 1):

if 0 <= i + m < rows and 0 <= j + n < cols: pixel\_sum += input\_matrix[i + m, j + n] count += 1

blurred\_matrix[i, j] = pixel\_sum / count return blurred\_matrix

input\_matrix = np.array([[1, 1, 1],

[1, 0, 1],

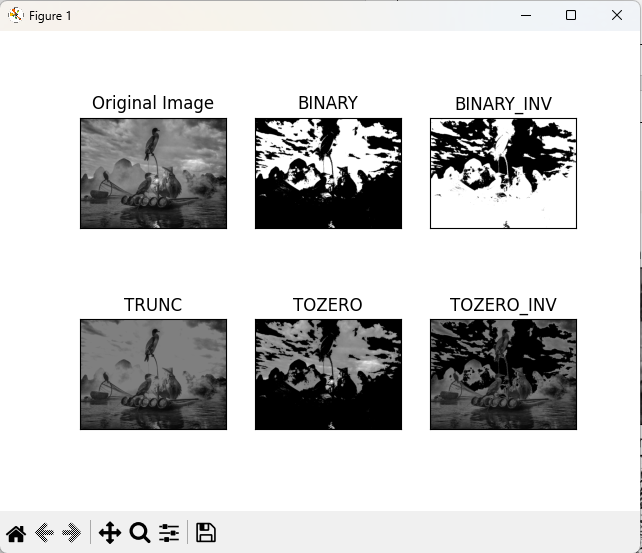
[1, 1, 1]])

kernel\_size = 3

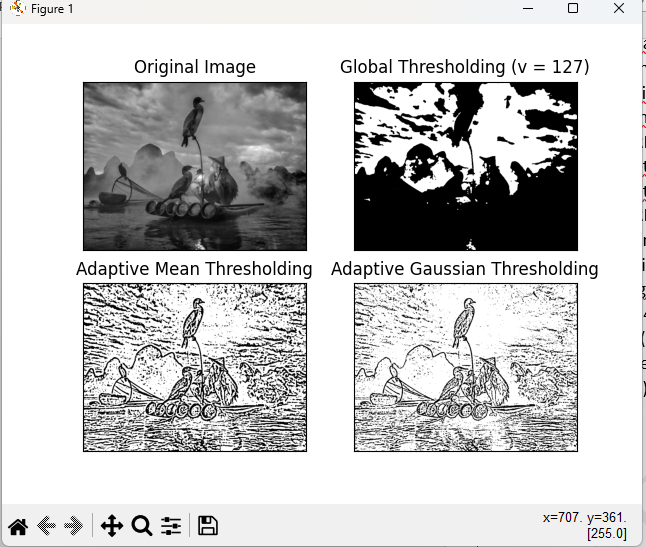
blurred\_result = average\_blurring(input\_matrix, kernel\_size)

print("Input Matrix:") print(input\_matrix) print("\nBlurred Result:") print(blurred\_result)

## 3)Thresholding (Output):

(SIMPLE THRESHOLDING)

(ADAPTIVE THRESHOLDING)



## 3)Thresholding (Code):

**SIMPLE THRESHOLDING**

import cv2 as cv  
import numpy as np  
from matplotlib import pyplot as plt  
img = cv.imread(r"X:\BIRD.jpg", cv.IMREAD\_GRAYSCALE)  
assert img is not None, "file could not be read, check with os.path.exists()"  
ret,thresh1 = cv.threshold(img,127,255,cv.THRESH\_BINARY)  
ret,thresh2 = cv.threshold(img,127,255,cv.THRESH\_BINARY\_INV)  
ret,thresh3 = cv.threshold(img,127,255,cv.THRESH\_TRUNC)  
ret,thresh4 = cv.threshold(img,127,255,cv.THRESH\_TOZERO)  
ret,thresh5 = cv.threshold(img,127,255,cv.THRESH\_TOZERO\_INV)  
titles = ['Original Image','BINARY','BINARY\_INV','TRUNC','TOZERO','TOZERO\_INV']  
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]  
for i in range(6):  
 plt.subplot(2,3,i+1),plt.imshow(images[i],'gray',vmin=0,vmax=255)  
 plt.title(titles[i])  
 plt.xticks([]),plt.yticks([])

plt.show()

**ADAPTIVE THRESHOLDING**

import cv2 as cv  
import numpy as np  
from matplotlib import pyplot as plt  
img = cv.imread(r"X:\BIRD.jpg", cv.IMREAD\_GRAYSCALE)  
assert img is not None, "file could not be read, check with os.path.exists()"  
img = cv.medianBlur(img,5)  
ret,th1 = cv.threshold(img,127,255,cv.THRESH\_BINARY),\  
 cv.THRESH\_BINARY,11,2)  
th2 = cv.adaptiveThreshold(img,255,cv.ADAPTIVE\_THRESH\_MEAN\_C  
th3 = cv.adaptiveThreshold(img,255,cv.ADAPTIVE\_THRESH\_GAUSSIAN\_C,\  
 cv.THRESH\_BINARY,11,2)  
titles = ['Original Image', 'Global Thresholding (v = 127)',  
 'Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding']  
images = [img, th1, th2, th3]  
for i in range(4):  
 plt.subplot(2,2,i+1),plt.imshow(images[i],'gray')  
 plt.title(titles[i])  
 plt.xticks([]),plt.yticks([])  
plt.show()

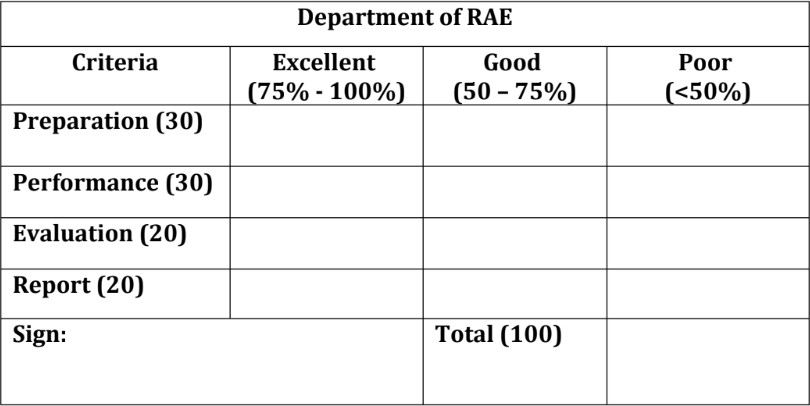
**(Otsu’s Thresholding )**

******

**Input Image** **Output Image**

**Otsu’s Thresholding:**

import cv2  
import numpy as np  
  
image1 = cv2.imread(r"C:\Users\RAGHUL\Downloads\krishna.jpg")  
  
img = cv2.cvtColor(image1, cv2.COLOR\_BGR2GRAY)  
  
ret, thresh1 = cv2.threshold(img, 120, 255, cv2.THRESH\_BINARY +  
 cv2.THRESH\_OTSU)  
  
  
cv2.imshow('Otsu Threshold', thresh1)  
  
if cv2.waitKey(0) & 0xff == 27:  
 cv2.destroyAllWindows()



# Result:

Thus, the Geometrical Transformations and Filtering Techniques were learnt using OpenCV.

# Post Lab Questions:

1. What is the difference between affine transformation and perspective transformation?
2. What do you mean by cartooning?
3. What does the filter2D function do? Explain with the arguments
4. Write a program for Ostu’s thresholding without using inbuilt function

# Post Lab Answers:

1. Difference between affine transformation and perspective transformation:
   * **Affine Transformation:** Preserves parallel lines in an image. It includes operations like translation, rotation, scaling, and shearing, where lines remain parallel before and after transformation.
   * **Perspective Transformation:** Involves transformations that allow perspective distortion, where parallel lines might converge or diverge. This transformation is more general and includes changes that aren't solely based on affine operations.
2. **– Cartooning** refers to the process of rendering images in a simplified or exaggerated manner, often resembling cartoons or illustrations. It involves reducing detail, enhancing outlines, and simplifying colors or shading to create a stylized or artistic effect.

# Explanation of the `filter2D` function with its arguments:

* + `filter2D` is an OpenCV function used for applying a custom convolution kernel to an image.

# Arguments:

* + - `src`: The input image.
    - `ddepth`: Depth of the output image; set to -1 to match `src` depth.
    - `kernel`: The convolution kernel, usually a numpy array.
    - `anchor`: The anchor point of the kernel (default is (-1, -1), which means the center of the kernel).
    - `delta`: Optional value added to the filtered pixels.
    - `borderType`: Specifies the pixel extrapolation method.

# Program for Otsu’s thresholding without using inbuilt function:

* + Here's an implementation of Otsu's thresholding without using the inbuilt

`cv2.threshold` function in Python:

import numpy as np import cv2

def otsu\_threshold(image):

pixel\_counts = [np.sum(image == i) for i in range(256)] total\_pixels = sum(pixel\_counts)

sum\_b = 0

weight\_b = 0

maximum = 0

threshold = 0

for i in range(256):

weight\_b += pixel\_counts[i] if weight\_b == 0:

continue

weight\_f = total\_pixels - weight\_b if weight\_f == 0:

break

sum\_b += i \* pixel\_counts[i] mean\_b = sum\_b / weight\_b mean\_f = (sum - sum\_b) / weight\_f

between = weight\_b \* weight\_f \* (mean\_b - mean\_f) \*\* 2 if between > maximum:

maximum = between threshold = i

return threshold # Example usage

image = cv2.imread('path/to/your/image.jpg', cv2.IMREAD\_GRAYSCALE)

threshold\_value = otsu\_threshold(image)

ret, thresholded\_image = cv2.threshold(image, threshold\_value, 255, cv2.THRESH\_BINARY)

cv2.imshow('Thresholded Image', thresholded\_image) cv2.waitKey(0)

cv2.destroyAllWindows()