**Experiment 4**

**Geometric Transformation and Filtering Using OpenCV**

**Aim:**

1. To implement the following Geometric Transformations a Filtering functions on an image in Open CV:
2. Translation, Rotation, Affine Transformation and Perspective Transformation
3. 2D convolution, Averaging and Blurring
4. Thresholding

# **Software/ Packages Used:**

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

**SMOOTHENING WITHOUT INBUILT FUNCTION:**

**PROGRAM:**

import cv2  
import numpy as np  
  
# Define your matrix data (replace this with your own array)  
matrix\_data = np.array([[10, 20, 30, 40, 50],  
 [60, 70, 80, 90, 100],  
 [110, 120, 130, 140, 150],  
 [160, 170, 180, 190, 200],  
 [210, 220, 230, 240, 250]], dtype=np.uint8)  
  
# Display the original matrix  
print("Original Matrix:")  
print(matrix\_data)  
  
# Apply different smoothing filters  
gaussian\_filtered = cv2.GaussianBlur(matrix\_data, (5, 5), 0)  
median\_filtered = cv2.medianBlur(matrix\_data, 5)  
average\_filtered = cv2.blur(matrix\_data, (5, 5))  
bilateral\_filtered = cv2.bilateralFilter(matrix\_data, 9, 75, 75)  
box\_filtered = cv2.boxFilter(matrix\_data, -1, (5, 5))  
  
# Display the resultant matrices after applying different filters  
print("\nResultant Matrices:")  
print("Gaussian Filtered:")  
print(gaussian\_filtered)  
print("\nMedian Filtered:")  
print(median\_filtered)  
print("\nAverage Filtered:")  
print(average\_filtered)  
print("\nBilateral Filtered:")  
print(bilateral\_filtered)  
print("\nBox Filtered:")  
print(box\_filtered)

**OUTPUT**

**Original Matrix:**

**[[ 10 20 30 40 50]**

**[ 60 70 80 90 100]**

**[110 120 130 140 150]**

**[160 170 180 190 200]**

**[210 220 230 240 250]]**

**Resultant Matrices:**

**Gaussian Filtered:**

**[[ 55 59 68 76 80]**

**[ 74 78 86 95 99]**

**[118 121 130 139 143]**

**[161 165 174 183 186]**

**[180 184 193 201 205]]**

**Median Filtered:**

**[[ 30 40 50 50 50]**

**[ 60 70 80 90 100]**

**[110 120 130 140 150]**

**[160 170 180 190 200]**

**[210 210 210 220 230]]**

**Average Filtered:**

**[[ 82 84 90 96 98]**

**[ 92 94 100 106 108]**

**[122 124 130 136 138]**

**[152 154 160 166 168]**

**[162 164 170 176 178]]**

**Bilateral Filtered:**

**[[ 73 76 78 82 86]**

**[ 91 95 99 103 108]**

**[118 124 130 136 142]**

**[152 157 161 165 169]**

**[174 178 182 184 187]]**

**Box Filtered:**

**[[ 82 84 90 96 98]**

**[ 92 94 100 106 108]**

**[122 124 130 136 138]**

**[152 154 160 166 168]**

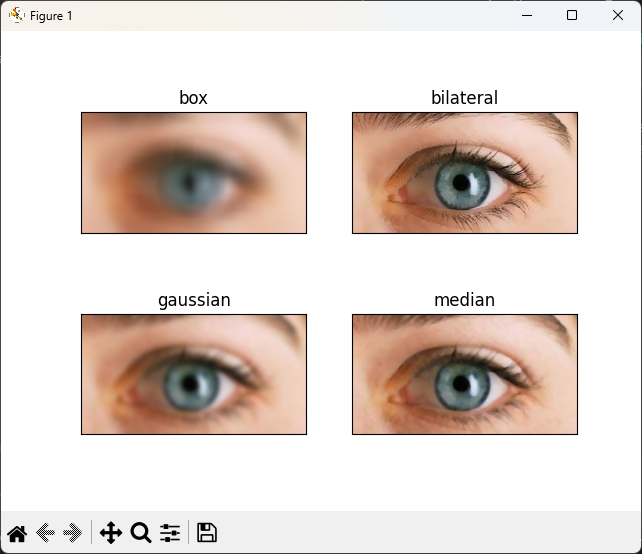
**[162 164 170 176 178]]**

**SMOOTHENING WITH INBUILT FUNCTION**

**PROGRAMS**

import cv2  
import matplotlib.pyplot as plt  
img = cv2.imread('images.jpg')  
cv2.imshow('i', img)  
cv2.waitKey(0)  
img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)  
# blurrings in cv2  
box = cv2.boxFilter(img, -1, (25, 25))  
bilateral = cv2.bilateralFilter(img, 10, 35, 25)  
gaussian = cv2.GaussianBlur(img, (17, 17), 0)  
median = cv2.medianBlur(img, ksize=5)  
images = [box, bilateral, gaussian, median]  
title = ['box', 'bilateral', 'gaussian', 'median']  
# blurrings in cv2  
for i in range(4):  
 plt.subplot(2, 2, i+1)  
 plt.title(title[i])  
 plt.imshow(images[i])  
 plt.xticks([])  
 plt.yticks([])  
plt.show()  
cv2.waitKey(0)  
cv2.destroyAllWindows()

**OUTPUT**

****

**AVERAGE BLUR WITH INBUILT FUNCTION:**

**PROGRAM:**

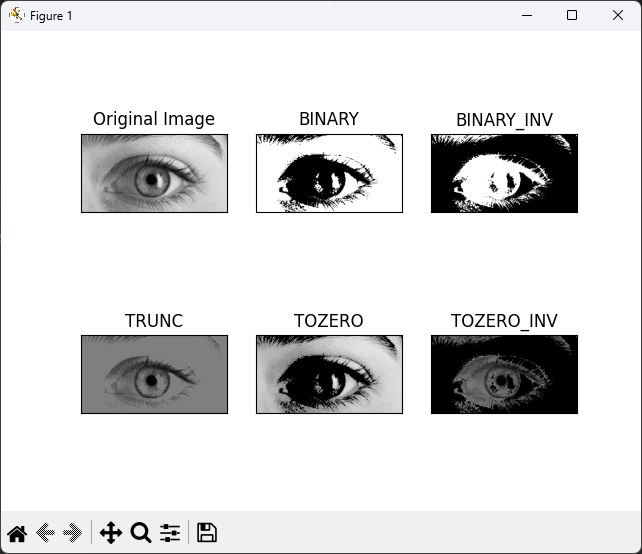
import cv2 as cv  
img = cv.imread('images.jpg')  
cv.imshow('eye', img)  
# Averaging  
average = cv.blur(img, (3,3))  
cv.imshow('Average Blur', average)  
cv.waitKey(0)  
cv.destroyAllWindows()

# **SIMPLE THRESHOLDING WITH INBUILT FUNCTION:**

**PROGRAMS:**

import cv2 as cv  
import numpy as np  
from matplotlib import pyplot as plt  
img = cv.imread('images.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()

**OUTPUT**:

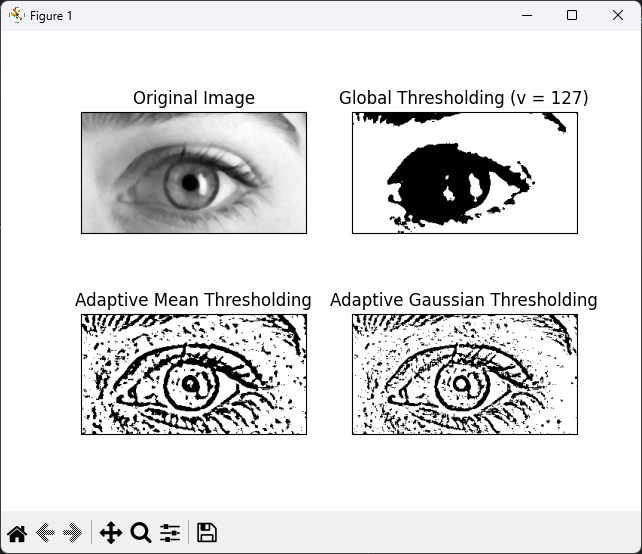


**ADAPTIVE THRESHOLDING WITH IINBUILT FUNCTION**

**PROGRAMS:**

import cv2 as cv  
import numpy as np  
from matplotlib import pyplot as plt  
img = cv.imread('images.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)  
th2 = cv.adaptiveThreshold(img,255,cv.ADAPTIVE\_THRESH\_MEAN\_C,\  
 cv.THRESH\_BINARY,11,2)  
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()

**OUTPUT:**

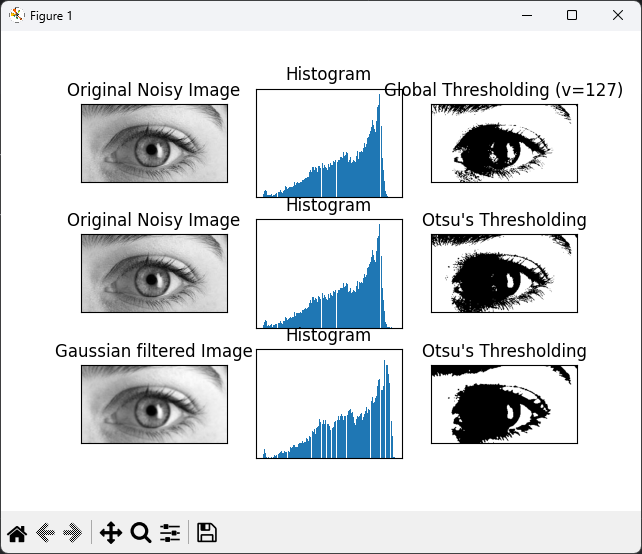
****

**OPTIMAL THRESHOLDING WITH INBUILT FUNCTION:**

**PROGRAM:**

import cv2 as cv  
import numpy as np  
from matplotlib import pyplot as plt  
img = cv.imread('images.jpg', cv.IMREAD\_GRAYSCALE)  
assert img is not None, "file could not be read, check with os.path.exists()"  
# global thresholding  
ret1,th1 = cv.threshold(img,127,255,cv.THRESH\_BINARY)  
# Otsu's thresholding  
ret2,th2 = cv.threshold(img,0,255,cv.THRESH\_BINARY+cv.THRESH\_OTSU)  
# Otsu's thresholding after Gaussian filtering  
blur = cv.GaussianBlur(img,(5,5),0)  
ret3,th3 = cv.threshold(blur,0,255,cv.THRESH\_BINARY+cv.THRESH\_OTSU)  
# plot all the images and their histograms  
images = [img, 0, th1,  
 img, 0, th2,  
 blur, 0, th3]  
titles = ['Original Noisy Image','Histogram','Global Thresholding (v=127)',  
 'Original Noisy Image','Histogram',"Otsu's Thresholding",  
 'Gaussian filtered Image','Histogram',"Otsu's Thresholding"]  
for i in range(3):  
 plt.subplot(3,3,i\*3+1),plt.imshow(images[i\*3],'gray')  
 plt.title(titles[i\*3]), plt.xticks([]), plt.yticks([])  
 plt.subplot(3,3,i\*3+2),plt.hist(images[i\*3].ravel(),256)  
 plt.title(titles[i\*3+1]), plt.xticks([]), plt.yticks([])  
 plt.subplot(3,3,i\*3+3),plt.imshow(images[i\*3+2],'gray')  
 plt.title(titles[i\*3+2]), plt.xticks([]), plt.yticks([])  
plt.show()

**OUTPUT:**

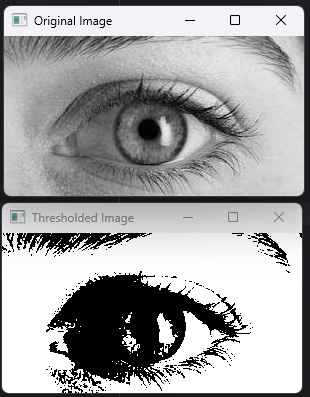


**SIMPLE THRESHOLDING WITHOUT INBUILT FUNCTION:**

**PROGRAM:**

import cv2  
import numpy as np  
# Read an image with error handling  
image\_path = 'images.jpg'  
# Replace 'image.jpg' with the correct file path  
image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)  
# Check if the image was loaded successfully  
if image is None:  
 print(f"Error: Unable to load the image at '{image\_path}'. Please check the file path.")  
else:  
 # Define threshold value (replace this value as needed)  
 threshold\_value = 127  
 # Create an empty output image of the same size as the input image  
 thresholded\_image = np.zeros\_like(image)  
 # Apply thresholding manually  
 rows, cols = image.shape  
 for i in range(rows):  
 for j in range(cols):  
 if image[i, j] > threshold\_value:  
 thresholded\_image[i, j] = 255 # Set pixel to white (255)  
 else:  
 thresholded\_image[i, j] = 0 # Set pixel to black (0)  
 # Display the original and thresholded images  
 cv2.imshow('Original Image', image)  
 cv2.imshow('Thresholded Image', thresholded\_image)  
 cv2.waitKey(0)  
 cv2.destroyAllWindows()

**OUTPUT**:

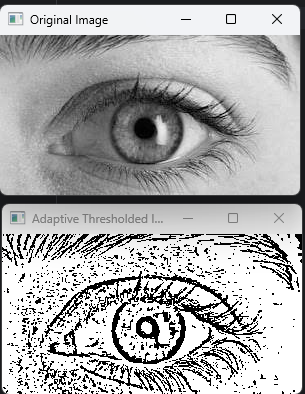


**ADAPTIVE THRESHOLDING WITHOUT INBUILT FUNCTION:**

**PROGRAM:**

import cv2  
import numpy as np  
# Read an image with error handling  
image\_path = 'images.jpg' # Replace 'image.jpg' with the correct file path  
image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)  
# Check if the image was loaded successfully  
if image is None:  
 print(f"Error: Unable to load the image at '{image\_path}'. Please check the file path.")  
else:  
 # Define the window size and constant C for adaptive thresholding  
 window\_size = 11 # Size of the window for local analysis  
 C = 5 # Constant subtracted from the mean  
 # Create an empty output image of the same size as the input image  
 output\_image = np.zeros\_like(image)  
  
 # Iterate through the image to perform adaptive thresholding manually  
 rows, cols = image.shape  
 for i in range(rows):  
 for j in range(cols):  
 # Define the region of interest (ROI) within the window size  
 roi = image[max(0, i - window\_size // 2): min(rows, i + window\_size // 2 + 1),  
 max(0, j - window\_size // 2): min(cols, j + window\_size // 2 + 1)]  
 # Check if the ROI is not empty  
 if roi.size != 0:  
 # Calculate the local mean within the ROI  
 local\_mean = np.mean(roi)  
 # Apply adaptive thresholding logic  
 if image[i, j] > local\_mean - C:  
 output\_image[i, j] = 255 # Set pixel to white (255)  
 else:  
 output\_image[i, j] = 0 # Set pixel to black (0)  
  
 # Display the original and adaptively thresholded images  
 cv2.imshow('Original Image', image)  
 cv2.imshow('Adaptive Thresholded Image', output\_image)  
 cv2.waitKey(0)  
 cv2.destroyAllWindows()

**OUTPUT**:



**OPTIMAL THRESHOLDING WITHOUT INBUILT FUNCTION:**

**PROGRAM:**

import cv2  
import numpy as np  
  
# Read an image (replace 'image.jpg' with your image file)  
image = cv2.imread('images.jpg', cv2.IMREAD\_GRAYSCALE)  
  
# Check if the image was loaded successfully  
if image is None:  
 print("Error: Unable to load the image. Please check the file path.")  
else:  
 # Normalize pixel values to the range [0, 1]  
 normalized\_image = image.astype(float) / 255.0  
  
 # Calculate histogram and probabilities  
 histogram, bins = np.histogram(normalized\_image.flatten(), bins=256, range=[0, 1], density=True)  
 probabilities = histogram / np.sum(histogram)  
  
 # Calculate cumulative distribution function (CDF)  
 cdf = np.cumsum(probabilities)  
  
 # Calculate between-class variance for all possible thresholds  
 thresholds = np.arange(0, 1, 0.01)  
 variances = []  
 for threshold in thresholds:  
 w0 = cdf[int(threshold \* 255)]  
 w1 = 1 - w0  
  
 if w0 == 0 or w1 == 0:  
 variances.append(0)  
 continue  
  
 hist\_segment0 = histogram[:int(threshold \* 255) + 1]  
 hist\_segment1 = histogram[int(threshold \* 255) + 1:]  
  
 mu0 = np.sum(np.arange(0, len(hist\_segment0)) \* hist\_segment0) / w0  
 mu1 = np.sum(np.arange(len(hist\_segment0), len(hist\_segment0) + len(hist\_segment1)) \* hist\_segment1) / w1  
  
 variances.append(w0 \* w1 \* (mu0 - mu1) \*\* 2)  
  
 # Find the optimal threshold that maximizes between-class variance  
 optimal\_threshold = thresholds[np.argmax(variances)]  
  
 # Apply the optimal thresholding manually  
 rows, cols = image.shape  
 optimal\_thresholded\_image = np.zeros\_like(image)  
 for i in range(rows):  
 for j in range(cols):  
 if normalized\_image[i, j] > optimal\_threshold:  
 optimal\_thresholded\_image[i, j] = 255 # Set pixel to white (255)  
 else:  
 optimal\_thresholded\_image[i, j] = 0 # Set pixel to black (0)  
  
 # Display the original and optimal thresholded images  
 cv2.imshow('Original Image', image)  
 cv2.imshow('Optimal Thresholded Image', optimal\_thresholded\_image)  
 cv2.waitKey(0)  
 cv2.destroyAllWindows()

**OUTPUT**:

# 

**SIMPLE THRESHOLDING WITH MATRIX:**

**PROGRAM:**

import numpy as np  
  
# Function for simple thresholding without using inbuilt functions  
def simple\_threshold(matrix, threshold\_value):  
 rows, cols = matrix.shape  
 thresholded\_matrix = np.zeros\_like(matrix, dtype=np.uint8)  
  
 for i in range(rows):  
 for j in range(cols):  
 if matrix[i, j] > threshold\_value:  
 thresholded\_matrix[i, j] = 255 # Set pixel to white (255)  
 else:  
 thresholded\_matrix[i, j] = 0 # Set pixel to black (0)  
  
 return thresholded\_matrix  
  
# Example matrix input (replace this with user input)  
matrix = np.array([[10, 20, 30, 40, 50],  
 [60, 70, 80, 90, 100],  
 [110, 120, 130, 140, 150],  
 [160, 170, 180, 190, 200],  
 [210, 220, 230, 240, 250]], dtype=np.uint8)  
  
# Define the threshold value  
threshold\_value = 120  
  
# Apply simple thresholding manually  
result\_thresholded = simple\_threshold(matrix, threshold\_value)  
  
# Display the original and thresholded images  
print("Original Matrix:")  
print(matrix)  
print("\nThresholded Matrix:")  
print(result\_thresholded)

**OUTPUT:**

**Original Matrix:**

**[[ 10 20 30 40 50]**

**[ 60 70 80 90 100]**

**[110 120 130 140 150]**

**[160 170 180 190 200]**

**[210 220 230 240 250]]**

**Thresholded Matrix:**

**[[ 0 0 0 0 0]**

**[ 0 0 0 0 0]**

**[ 0 0 255 255 255]**

**[255 255 255 255 255]**

**[255 255 255 255 255]]**

**ADAPTIVE THRESHOLDING WITH MATRIX:**

**PROGRAM:**

**import cv2  
import numpy as np  
image = cv2.imread('images.jpg', cv2.IMREAD\_GRAYSCALE)  
arr=np.array([[11,222,206],[45,55,26],[70,81,99]])  
print(arr)  
window\_size = 11 # Size of the window for local analysis  
C = 5 # Constant subtracted from the mean  
 # Create an empty output image of the same size as the input image  
output\_image = np.zeros\_like(image)**

**# Iterate through the image to perform adaptive thresholding manually  
rows, cols = image.shape  
for i in range(rows):  
 for j in range(cols):  
 # Define the region of interest (ROI) within the window size  
 roi = image[max(0, i - window\_size // 2): min(rows, i + window\_size // 2 + 1)  
 max(0, j - window\_size // 2): min(cols, j + window\_size // 2 + 1)]  
 # Check if the ROI is not empty  
 if roi.size != 0:  
 # Calculate the local mean within the ROI  
 local\_mean = np.mean(roi)  
 # Apply adaptive thresholding logic  
 if image[i, j] > local\_mean - C:  
 output\_image[i, j] = 255 # Set pixel to white (255)  
 else:  
 output\_image[i, j] = 0 # Set pixel to black (0)  
cv2.imshow('Original Image', image)  
cv2.imshow('Adaptive Thresholded Image', output\_image)  
print(output\_image)  
cv2.waitKey(0)  
cv2.destroyAllWindows()**

**OUTPUT:**

**[[ 11 222 206]**

**[ 45 55 26]**

**[ 70 81 99]]**

**[[255 0 255 ... 0 0 0]**

**[255 0 255 ... 0 0 0]**

**[255 255 255 ... 0 0 0]**

**...**

**[255 255 255 ... 255 255 255]**

**[255 255 255 ... 255 255 255]**

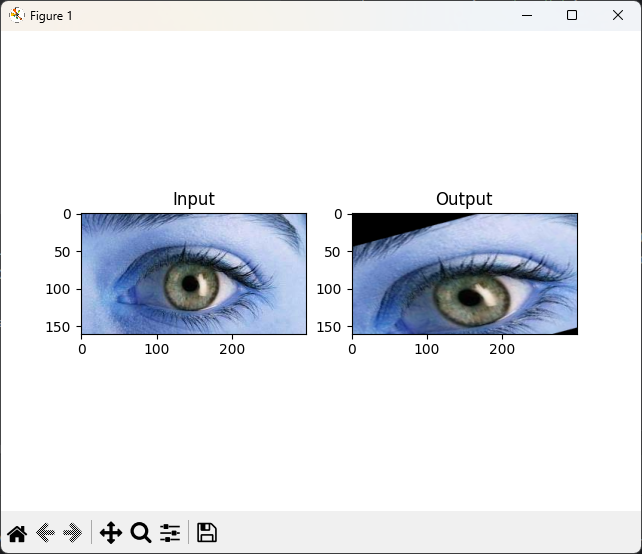
**[255 255 0 ... 255 255 255]]**

**AFFINE TRANSFORMATION:**

**PROGRAM:**

import cv2  
import numpy as np  
from matplotlib import pyplot as plt  
  
img = cv2.imread('images.jpg')  
rows, cols, ch = img.shape  
  
pts1 = np.float32([[50, 50],  
 [200, 50],  
 [50, 200]])  
  
pts2 = np.float32([[10, 100],  
 [200, 50],  
 [100, 250]])  
  
M = cv2.getAffineTransform(pts1, pts2)  
dst = cv2.warpAffine(img, M, (cols, rows))  
  
plt.subplot(121)  
plt.imshow(img)  
plt.title('Input')  
  
plt.subplot(122)  
plt.imshow(dst)  
plt.title('Output')  
  
plt.show()  
  
cv2.destroyAllWindows()

**OUTPUT:**

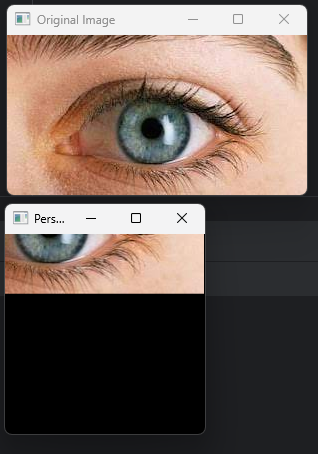
****

**PERSPECTIVE:**

**PROGRAM:**

import cv2  
import numpy as np  
image = cv2.imread('images.jpg')  
roi\_corners = np.array([[100, 100], [300, 100], [300, 300], [100, 300]], dtype=np.float32)  
output\_size = (200, 200)  
output\_corners = np.array([[0, 0], [output\_size[0] - 1, 0], [output\_size[0] - 1, output\_size[1] - 1], [0, output\_size[1] - 1]], dtype=np.float32)  
perspective\_matrix = cv2.getPerspectiveTransform(roi\_corners, output\_corners)  
result = cv2.warpPerspective(image, perspective\_matrix, output\_size)  
cv2.imshow('Original Image', image)  
cv2.imshow('Perspective Projection', result)  
cv2.waitKey(0)  
cv2.destroyAllWindows()

**OUTPUT:**

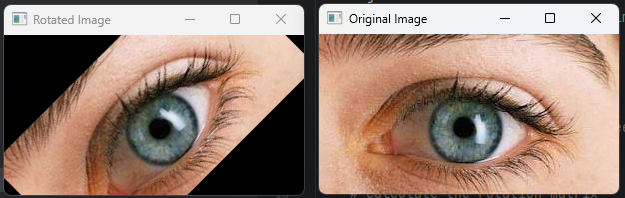
****

**ROTATED:**

**PROGRAM:**

import cv2  
# Read the image  
image = cv2.imread('images.jpg')  
# Check if the image was loaded successfully  
if image is None:  
 print("Error: Unable to load the image. Please check the file path.")  
else:  
 # Get image dimensions  
 height, width = image.shape[:2]  
 # Set the rotation angle (in degrees)  
 angle = 45  
 # Calculate the rotation matrix  
 rotation\_matrix = cv2.getRotationMatrix2D((width / 2, height / 2), angle, 1)  
 # Apply the rotation transformation  
 rotated\_image = cv2.warpAffine(image, rotation\_matrix, (width, height))  
 # Display the original and rotated images  
 cv2.imshow('Original Image', image)  
 cv2.imshow('Rotated Image', rotated\_image)  
 cv2.waitKey(0)  
 cv2.destroyAllWindows()

**OUTPUT:**

****

**FILTER 2D:**

**PROGRAM:**

import cv2 as cv

img = cv.imread('MyPic.jpg')

kernel = np.ones((5,5),np.float32)/25

dst = cv.filter2D(img,-1,kernel)

cv.imshow('averageimage', dst)

cv.waitKey(0)

plt.subplot(121),plt.imshow(img),plt.title('Original')

plt.xticks([]), plt.yticks([])

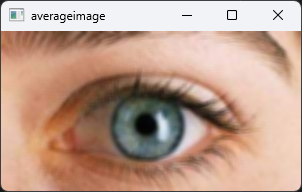
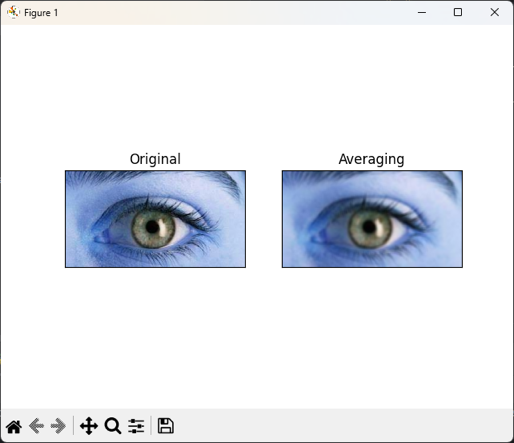
plt.subplot(122),plt.imshow(dst),plt.title('Averaging')

plt.xticks([]), plt.yticks([])

plt.show()

cv.destroyAllWindows()

**OUTPUT**:



# 

# **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?**

Affine transformation involves preserving parallel lines and ratios of distances between points, making it suitable for scaling, rotation, translation, and skewing. Perspective transformation, on the other hand, simulates the way objects appear smaller as they move away, incorporating vanishing points and depth to represent a three- dimensional view.

1. **What do you mean by cartooning?**

Cartooning refers to the process of simplifying and exaggerating visual elements to create a stylized or comical representation, often characterized by bold outlines, simplified shapes, and vibrant colors. It is commonly used in art, animation, and image processing to achieve a playful or artistic effect.

1. **What does the filter2D function do? Explain with the arguments**

The filter2D function is used in image processing to perform a 2D convolution operation. It takes two main arguments: the input image and the kernel matrix. The input image undergoes convolution with the kernel, resulting in a filtered output image. This operation is crucial for tasks like blurring, sharpening, and edge detection in image processing.

1. **Write a program for Ostu’s thresholding without using inbuilt function**

def otsu\_thresholding(image):

hist, bins = np.histogram(image.flatten(), bins=256, range=[0, 256]) total\_pixels = image.size

sum\_all = np.sum(np.arange(256) \* hist) sumB = 0

wB = 0

maximum\_variance = 0

threshold = 0

for i in range(256): wB += hist[i]

if wB == 0: continue

wF = total\_pixels - wB if wF == 0:

break

sumB += i \* hist[i] mB = sumB / wB

mF = (sum\_all - sumB) / wF variance = wB \* wF \* (mB - mF) \*\* 2 if variance > maximum\_variance:

maximum\_variance = variance threshold = i

result = np.zeros\_like(image) result[image > threshold] = 255

return result