Course Name:	**Advance Python Programming and its application**	**Semester:**	**V**
Date of Performance:	**6 / 08 / 2024**	**Batch No:**	**APPA 2**
Faculty Name:	**Prof. Deepa Jain**	**Roll No:**	**16014022096**
Faculty Sign & Date:		**Grade/Marks:**	

Experiment No: 3

Title: Write a Python Program for Geometric Transformation of Images and Image Thresholding

Aim and Objective of the Experiment:

1) To learn following Geometric Transformation in OpenCV
   - Affine transformation
   - Perspective Transformation
   - Scaling
   - Translation
   - Rotation
2) To learn following Thresholding techniques in OpenCV
   - Simple thresholding
   - Adaptive thresholding
   - Otsu's thresholding

COs to be achieved:

CO2: Illustrate python libraries for image processing and its applications

Tools required:

Any python editor tool

Theory:

Transformations:

OpenCV provides two transformation functions, cv2.warpAffine and cv2.warpPerspective, with which you can have all kinds of transformations. cv2.warpAffine takes a 2x3 transformation matrix while cv2.warpPerspective takes a 3x3 transformation matrix as input.

Affine Transformation:

In affine transformation, all parallel lines in the original image will still be parallel in the output image. To find the transformation matrix, we need three points from input image and their corresponding locations in output image. Then cv2.getAffineTransform will create a 2x3 matrix which is to be passed to cv2.warpAffine.

Perspective Transformation:

For perspective transformation, you need a 3x3 transformation matrix. Straight lines will remain straight even after the transformation. To find this transformation matrix, you need 4 points on the input image and corresponding points on the output image. Among these 4 points, 3 of them should not be collinear. Then transformation matrix can be found by the function cv2.getPerspectiveTransform. Then apply cv2.warpPerspective with this 3x3 transformation matrix.

Scaling:

Scaling is just resizing of the image. OpenCV comes with a function cv2.resize() for this purpose. The size of the image can be specified manually, or you can specify the scaling factor. Different interpolation methods are used. Preferable interpolation methods are cv2.INTER_AREA for shrinking and cv2.INTER_CUBIC (slow) & cv2.INTER_LINEAR for zooming. By default, interpolation method used is cv2.INTER_LINEAR for all resizing purposes. You can resize an input image either of following methods:

Translation:

Translation is the shifting of object's location. If you know the shift in (x,y) direction, let it be (tx, ty), you can create the transformation matrix M as follows:

$$M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$$

You can take make it into a Numpy array of type np.float32 and pass it into cv2.warpAffine() function.

Rotation:

Rotation of an image for an angle θ is achieved by the transformation matrix of the form

$$M = \begin{bmatrix} cos\,\theta & -\sin\theta \\ sin\,\theta & cos\,\theta \end{bmatrix}$$

OpenCV also provides scaled rotation with adjustable center of rotation so you can rotate at any location you prefer.

Simple Thresholding:

Here, the matter is straight forward. If pixel value is greater than a threshold value, it is assigned one value (may be white), else it is assigned another value (may be black). The function used is cv2.threshold. First argument is the source image, which should be a grayscale image. Second argument is the threshold value which is used to classify the pixel values. Third argument is the maxVal which represents the value to be given if pixel value is more than (sometimes less than) the

threshold value. OpenCV provides different styles of thresholding and it is decided by the fourth parameter of the function. Different types are:

- cv2.THRESH_BINARY
- cv2.THRESH_BINARY_INV
- cv2.THRESH_TRUNC
- cv2.THRESH_TOZERO
- cv2.THRESH_TOZERO_INV

Adaptive Thresholding:

In the previous section, we used a global value as threshold value. But it may not be good in all the conditions where image has different lighting conditions in different areas. In that case, we go for adaptive thresholding. In this, the algorithm calculate the threshold for a small regions of the image. So we get different thresholds for different regions of the same image and it gives us better results for images with varying illumination.

It has three 'special' input params and only one output argument.

Adaptive Method - It decides how thresholding value is calculated.

•cv2.ADAPTIVE_THRESH_MEAN_C: threshold value is the mean of neighborhood area.

•cv2.ADAPTIVE_THRESH_GAUSSIAN_C: threshold value is the weighted sum of neighborhood values where weights are a gaussian window.

Block Size - It decides the size of neighborhood area.

C - It is just a constant which is subtracted from the mean or weighted mean calculated.

Otsu's Binarization:

In global thresholding, we used an arbitrary value for threshold value, right? So, how can we know a value we selected is good or not? Answer is, trial and error method. But consider a bimodal image (In simple words, bimodal image is an image whose histogram has two peaks). For that image, we can approximately take a value in the middle of those peaks as threshold value, right ? That is what Otsu binarization does. So in simple words, it automatically calculates a threshold value from image histogram for a bimodal image. (For images which are not bimodal, binarization won't be accurate.) For this, our cv2.threshold() function is used, but pass an extra flag, cv2.THRESH_OTSU. For threshold value, simply pass zero. Then the algorithm finds the optimal threshold value and returns you as the second output, retVal. If Otsu thresholding is not used, retVal is same as the threshold value you used.

Code:

1) Write a python code for following Geometric Transformation in OpenCV
   - Affine transformation
   - Perspective Transformation
   - Scaling

- Translation
- Rotation

2) To learn following Thresholding techniques in OpenCV
   - Simple thresholding
   - Adaptive thresholding
   - Otsu's thresholding

Output:

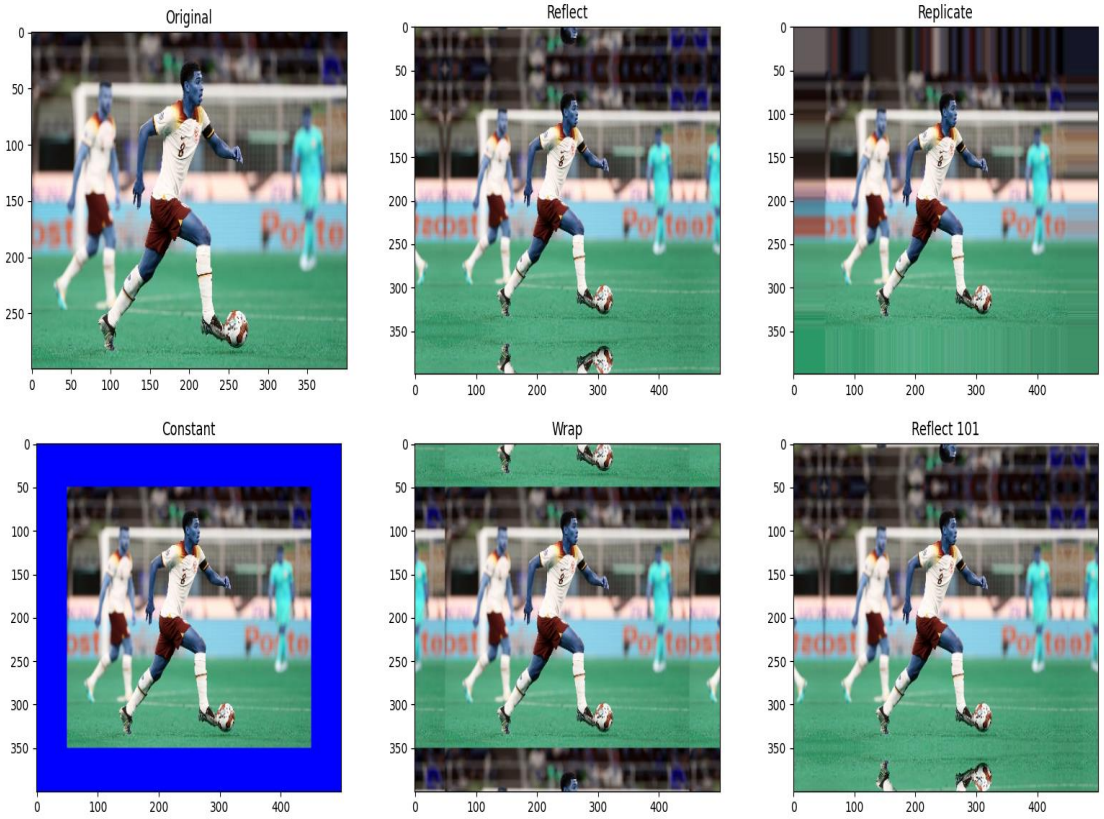
BODER:

CODE:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread('EXPERIMENT 02/bellingam.jpg', 1)
img = cv2.resize(img, (400, 300))

reflect = cv2.copyMakeBorder(img, 50, 50, 50, 50, cv2.BORDER_REFLECT)
replicate = cv2.copyMakeBorder(img, 50, 50, 50, 50, cv2.BORDER_REPLICATE)
constant = cv2.copyMakeBorder(img, 50, 50, 50, 50, cv2.BORDER_CONSTANT, value=(0,
0, 255))
wrap = cv2.copyMakeBorder(img, 50, 50, 50, 50, cv2.BORDER_WRAP)
reflect101 = cv2.copyMakeBorder(img, 50, 50, 50, 50, cv2.BORDER_REFLECT_101)


plt.subplot(2, 3, 1)
plt.imshow(img, cmap='gray')
plt.title('Original')

plt.subplot(2, 3, 2)
plt.imshow(reflect, cmap='gray')
plt.title('Reflect')

plt.subplot(2, 3, 3)
plt.imshow(replicate, cmap='gray')
plt.title('Replicate')

plt.subplot(2, 3, 4)
plt.imshow(constant, cmap='gray')
plt.title('Constant')

plt.subplot(2, 3, 5)
plt.imshow(wrap, cmap='gray')
```

```
plt.title('Wrap')

plt.subplot(2, 3, 6)
plt.imshow(reflect101, cmap='gray')
plt.title('Reflect 101')

plt.show()
```

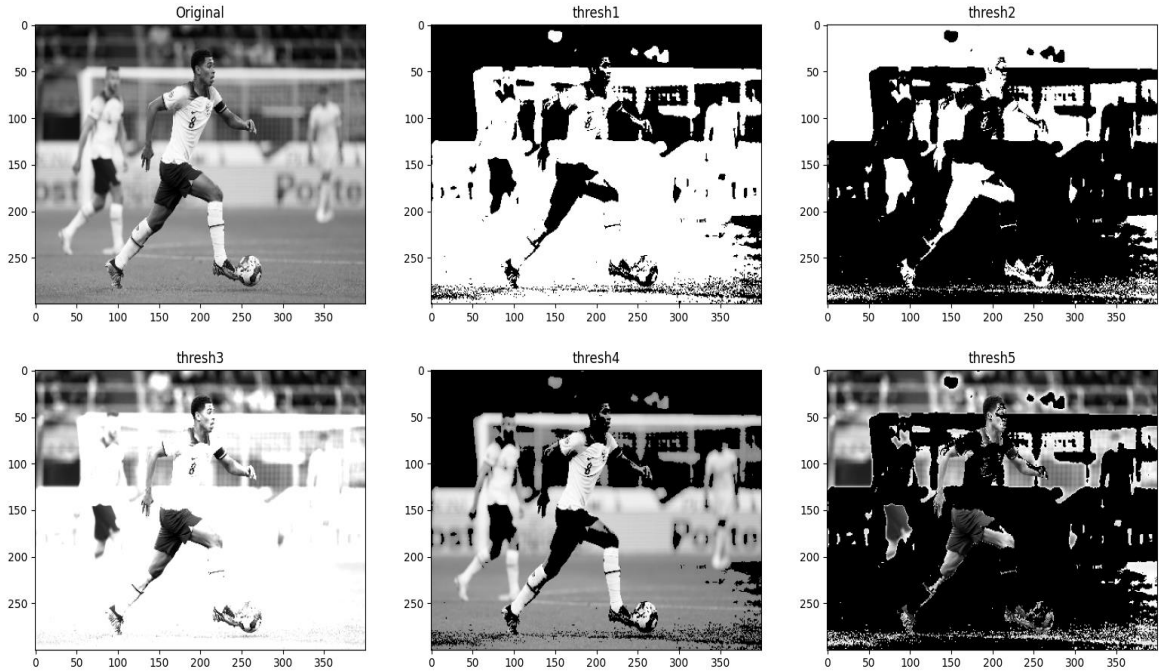
OUTPUT:



THRESHOLD:
CODE:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread('EXPERIMENT 02/bellingam.jpg', 0)
img = cv2.resize(img, (400, 300))

ret, thresh1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
ret, thresh2 = cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)
ret, thresh3 = cv2.threshold(img,127,255,cv2.THRESH_TRUNC)
ret, thresh4 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO)
ret, thresh5 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO_INV)


plt.subplot(2, 3, 1)
plt.imshow(img, cmap='gray')
plt.title('Original')

plt.subplot(2, 3, 2)
plt.imshow(thresh1, cmap='gray')
plt.title('thresh1')

plt.subplot(2, 3, 3)
plt.imshow(thresh2, cmap='gray')
plt.title('thresh2')

plt.subplot(2, 3, 4)
plt.imshow(thresh3, cmap='gray')
plt.title('thresh3')

plt.subplot(2, 3, 5)
plt.imshow(thresh4, cmap='gray')
plt.title('thresh4')

plt.subplot(2, 3, 6)
plt.imshow(thresh5, cmap='gray')
plt.title('thresh5')

plt.show()
```

OUTPUT:

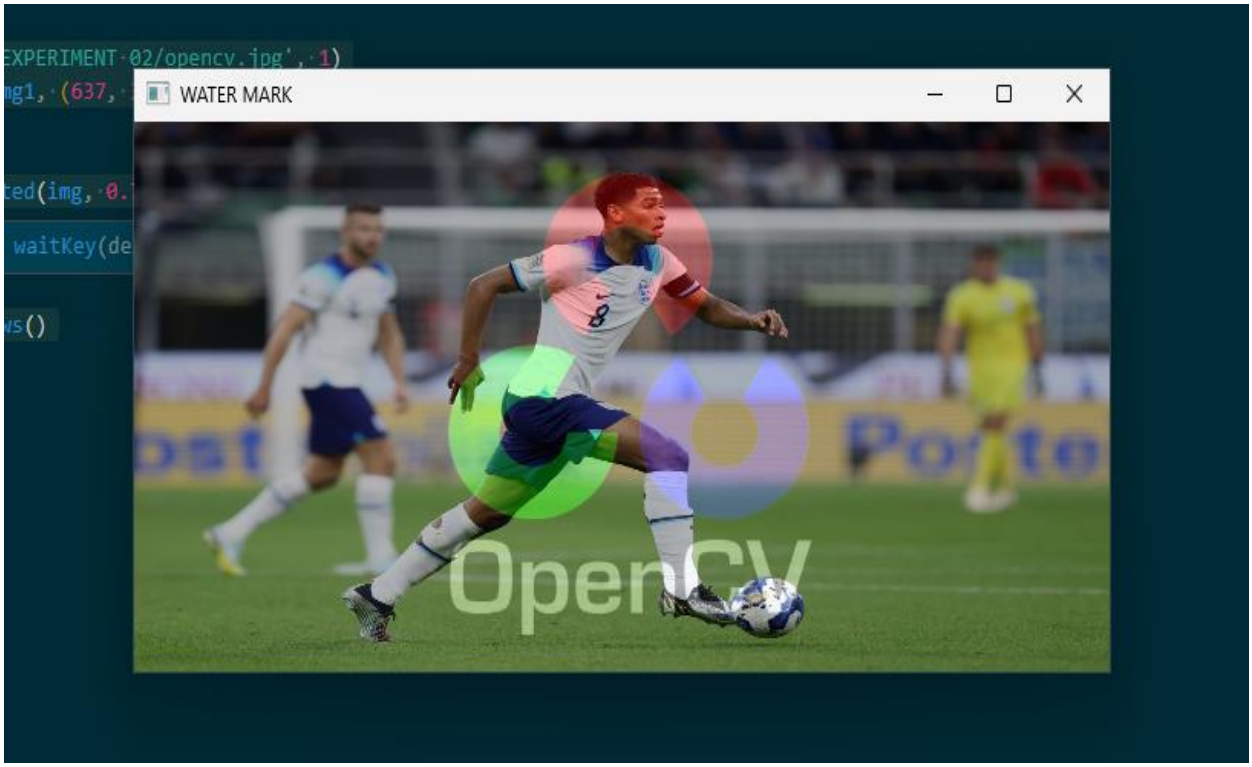


WATERMARK:
ADD BY WEIGHT:

CODE:

```
import cv2
import numpy as np


img = cv2.imread('EXPERIMENT 02/bellingam.jpg', 1)
img = cv2.resize(img, (637, 313))

img1 = cv2.imread('EXPERIMENT 02/opencv.jpg', 1)
img1 = cv2.resize(img1, (637, 313))


mark = cv2.addWeighted(img, 0.7, img1, 0.3, 0)

cv2.imshow('WATER MARK', mark)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



BITWISE AND OPERATOR:

CODE:

```
import cv2
import numpy as np


img = cv2.imread('EXPERIMENT 02/bellingam.jpg', 1)
img = cv2.resize(img, (637, 313))

img1 = cv2.imread('EXPERIMENT 02/opencv.jpg', 1)
img1 = cv2.resize(img1, (637, 313))


mark = cv2.bitwise_and(img,img1)

cv2.imshow('BITWISE AND', mark)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

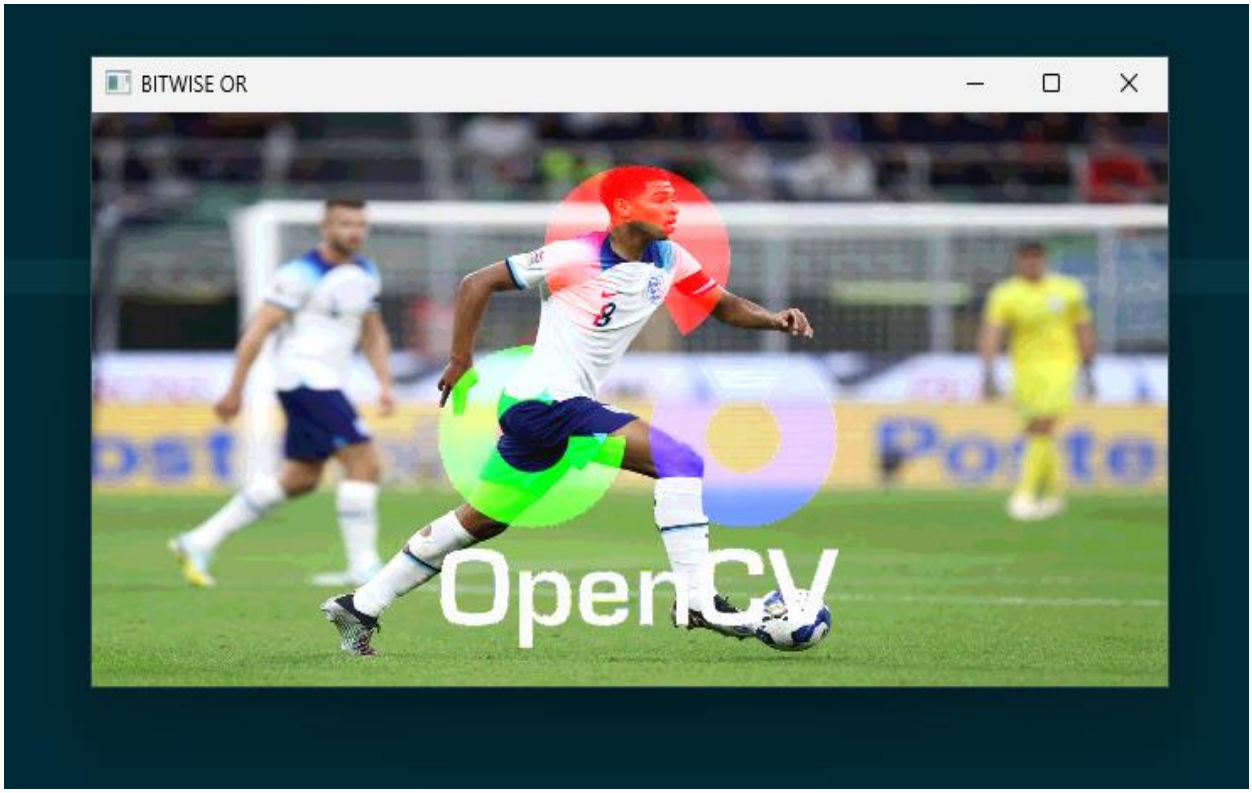
OUTPUT:

BITWISE OR OPERATOR:

CODE:

```
import cv2
import numpy as np


img = cv2.imread('EXPERIMENT 02/bellingam.jpg', 1)
img = cv2.resize(img, (637, 313))

img1 = cv2.imread('EXPERIMENT 02/opencv.jpg', 1)
img1 = cv2.resize(img1, (637, 313))


mark = cv2.bitwise_or(img,img1)

cv2.imshow('BITWISE OR', mark)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

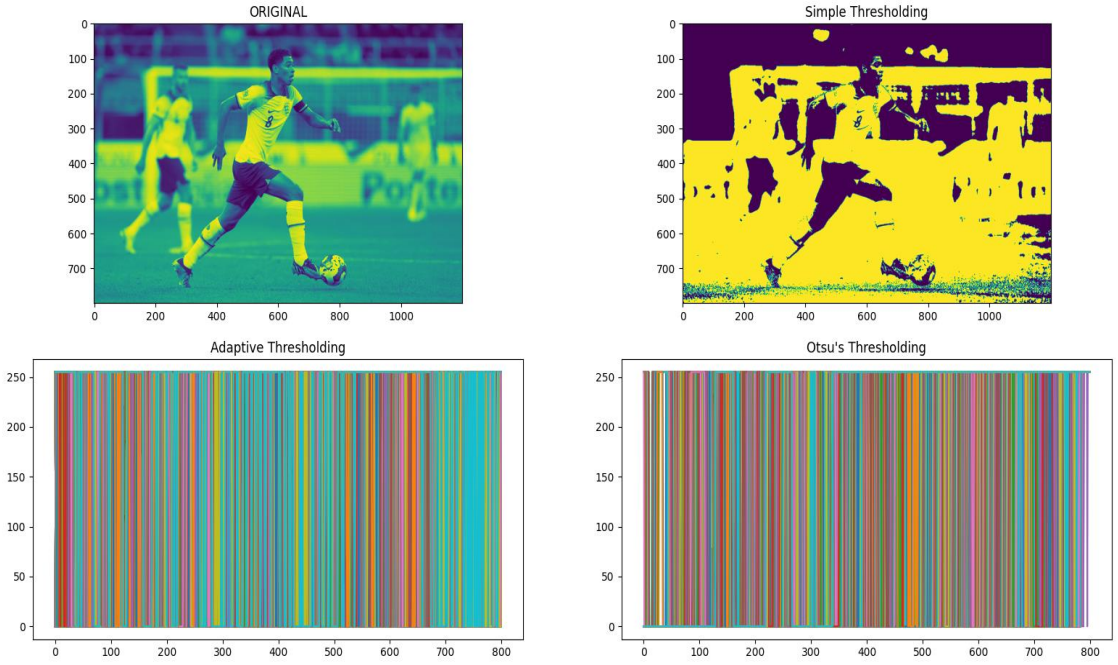
OUTPUT:



THRESHOLDING:
CODE:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

image = cv2.imread('EXPERIMENT 02/bellingam.jpg')
_, thresh1 = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)
adaptive_thresh = cv2.adaptiveThreshold(image, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY, 11, 2)
_, otsu_thresh = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)


plt.subplot(221)
plt.imshow(image)
plt.title('ORIGINAL')

plt.subplot(222)
plt.imshow(thresh1)
plt.title('Simple Thresholding')

plt.subplot(223)
plt.plot(adaptive_thresh)
plt.title('Adaptive Thresholding')

plt.subplot(224)
plt.plot(otsu_thresh)
plt.title('Otsu\'s Thresholding')

plt.show()
```

OUTPUT :


ORIGINAL
Simple Thresholding
Adaptive Thresholding
Otsu's Thresholding

Scaling & Translation & Rotation

CODE:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

image = cv2.imread('EXPERIMENT 02/bellingam.jpg')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
height, width = gray_image.shape

rotation_matrix = cv2.getRotationMatrix2D((width/2, height/2), 45, 1)
rotated_image = cv2.warpAffine(image, rotation_matrix, (width, height))

scaling_matrix = np.array([[1.5, 0, 0], [0, 1.5, 0]], dtype=float)
scaled_image = cv2.warpAffine(image, scaling_matrix, (int(width*1.5),
int(height*1.5)))

translation_matrix = np.array([[1, 0, 100], [0, 1, 50]], dtype=float)
translated_image = cv2.warpAffine(image, translation_matrix, (width, height))

points1 = np.float32([[50, 50], [200, 50], [50, 200]])
points2 = np.float32([[10, 100], [200, 50], [100, 250]])
affine_matrix = cv2.getAffineTransform(points1, points2)
affine_transformed = cv2.warpAffine(image, affine_matrix, (width, height))

points1 = np.float32([[56, 65], [368, 52], [28, 387], [389, 390]])
points2 = np.float32([[0, 0], [300, 0], [0, 300], [300, 300]])
perspective_matrix = cv2.getPerspectiveTransform(points1, points2)
perspective_transformed = cv2.warpPerspective(image, perspective_matrix, (300,
300))

plt.subplot(2, 3, 1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title('Original')

plt.subplot(2, 3, 2)
plt.imshow(cv2.cvtColor(rotated_image, cv2.COLOR_BGR2RGB))
plt.title('Rotated')

plt.subplot(2, 3, 3)
plt.imshow(cv2.cvtColor(scaled_image, cv2.COLOR_BGR2RGB))
plt.title('Scaled')

plt.subplot(2, 3, 4)
```

```
plt.imshow(cv2.cvtColor(translated_image, cv2.COLOR_BGR2RGB))
plt.title('Translated')

plt.subplot(2, 3, 5)
plt.imshow(cv2.cvtColor(affine_transformed, cv2.COLOR_BGR2RGB))
plt.title('Affine')

plt.subplot(2, 3, 6)
plt.imshow(cv2.cvtColor(perspective_transformed, cv2.COLOR_BGR2RGB))
plt.title('Perspective')

plt.show()
```

OUTPUT:

Post Lab Subjective/Objective type Questions:

1. Discuss the difference between Affine Transformation and Perspective Transformation.

ANS:

Affine Transformation:

Affine transformations preserve points, straight lines, and planes. Parallel lines remain parallel after an affine transformation. Uses a 2x3 matrix. Includes translation, scaling, rotation, and shearing. Transforms a rectangle into a parallelogram. If you apply an affine transformation to a square, it can become a rectangle or a parallelogram, but the sides will remain parallel.

Perspective Transformation:

Perspective transformations (homographies) can map parallel lines to intersecting lines, giving a sense of depth. Uses a 3x3 matrix. Includes all affine transformations plus perspective distortion. Transforms a rectangle into any quadrilateral. If you apply a perspective transformation to a square, it can become a trapezoid or any other quadrilateral, with lines that may no longer be parallel.

2. List down optimizations available for Otsu's binarization. and implement any 1.

ANS:

1.Histogram Smoothing: Applying smoothing techniques to the histogram can help reduce noise and improve the stability of the threshold calculation.

2. Adaptive Otsu's Method: Instead of using a single global threshold, adaptive methods divide the image into smaller regions and apply Otsu's method locally to handle varying lighting conditions.

3. Multilevel Otsu's Method: Extending Otsu's method to find multiple thresholds instead of just one. This is useful for images with more than two significant intensity levels.

4. Weighted Otsu's Method: Incorporating weights based on some criteria (e.g., local contrast) to adjust the importance of different regions in the image.

5. Modified Otsu's Method: Adjusting the basic Otsu's algorithm to account for specific characteristics of the image or noise.

6. Hardware Acceleration: Implementing Otsu's method using GPU or other hardware accelerations to speed up the computation, especially for large images.

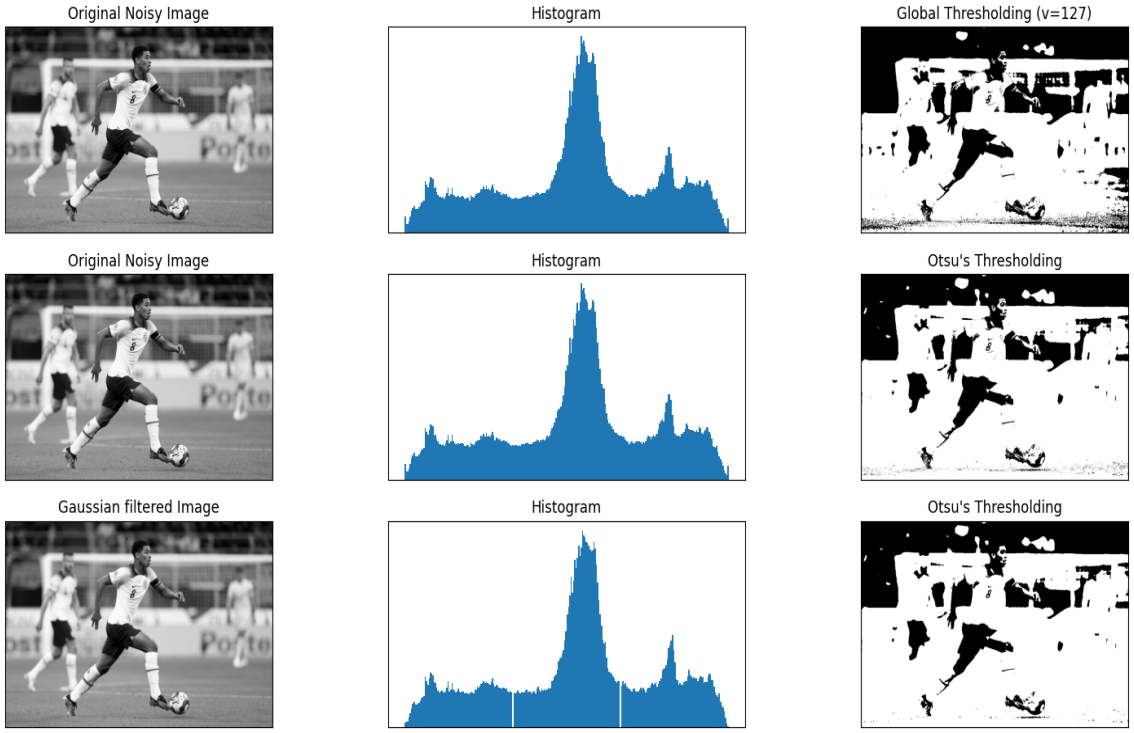
7. Parallel Processing: Leveraging multi-core processors to parallelize the computation of the threshold for faster performance.

8. Recursive Otsu's Method: Applying Otsu's method recursively to refine the threshold, especially useful in scenarios with complex image content.

CODE:

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

img = cv.imread('EXPERIMENT 02/bellingam.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:


Original Noisy Image
Histogram
Global Thresholding (v=127)
Original Noisy Image
Histogram
Otsu's Thresholding
Gaussian filtered Image
Histogram
Otsu's Thresholding

Conclusion:
We have successfully learned and implement Geometric Transformation like Affine transformation, Perspective Transformation, Scaling, Translation, Rotation and Thresholding techniques like Simple thresholding, Adaptive thresholding, Otsu's thresholding in Open cv using python library

Signature of faculty in-charge with Date: