Experiment 1: Point Processing Techniques

Aim:
To implement point processing techniques such as Digital Negative, Thresholding, Intensity Transformation, and Contrast Stretching on a grayscale image using OpenCV.

Software Requirements:
- Python
- OpenCV
- NumPy

Algorithm:
1. Start the program.
2. Import required libraries such as OpenCV and NumPy.
3. Read the input image in grayscale mode.
4. Apply Digital Negative transformation using s = 255 - r.
5. Apply Thresholding by converting pixel values based on a threshold.
6. Apply Intensity Transformation using linear scaling.
7. Apply Contrast Stretching to enhance image contrast.
8. Display original and processed images.
9. Stop the program.

Code:

```
import cv2
import numpy as np

img = cv2.imread('image.jpg', 0)

negative = 255 - img

_, threshold = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)

intensity = cv2.convertScaleAbs(img, alpha=1.2, beta=20)

r_min = np.min(img)
r_max = np.max(img)
```

```
contrast = ((img - r_min) / (r_max - r_min)) * 255
contrast = contrast.astype(np.uint8)

cv2.imshow('Original', img)
cv2.imshow('Negative', negative)
cv2.imshow('Threshold', threshold)
cv2.imshow('Intensity', intensity)

cv2.imshow('Contrast', contrast)
```

Result:
The point processing techniques were successfully implemented and visualized using OpenCV.

Experiment 2: Sharpening & Smoothing Filters

Aim:
To apply smoothing and sharpening filters on a digital image using OpenCV to reduce noise and enhance image details.

Software Requirements:
- Python
- OpenCV
- NumPy

Algorithm:
1. Start the program.
2. Import required libraries such as OpenCV and NumPy.
3. Read the input image in grayscale mode.
4. Apply smoothing filters such as averaging and Gaussian filter to reduce noise.
5. Apply sharpening filter using Laplacian operator to enhance edges.
6. Display original and filtered images.
7. Stop the program.

Code:

```
import cv2
import numpy as np

img = cv2.imread('image.jpg', 0)

smooth_avg = cv2.blur(img, (5, 5))

smooth_gaussian = cv2.GaussianBlur(img, (5, 5), 0)

laplacian = cv2.Laplacian(img, cv2.CV_64F)
sharpen = cv2.convertScaleAbs(laplacian)

cv2.imshow('Original Image', img)

cv2.imshow('Averaging Filter', smooth_avg)
cv2.imshow('Gaussian Filter', smooth_gaussian)
cv2.imshow('Sharpened Image', sharpen)
```

Result:
Smoothing and sharpening filters were successfully applied using OpenCV.

Experiment 3: Histogram Modelling

Aim:
To perform histogram modelling (histogram equalization) on a grayscale image using OpenCV to enhance image contrast.

Software Requirements:
- Python
- OpenCV
- NumPy

Algorithm:
1. Start the program.
2. Import required libraries such as OpenCV and NumPy.
3. Read the input image in grayscale mode.
4. Calculate the histogram of the image.
5. Apply histogram equalization to redistribute intensity values.
6. Display original and equalized images along with histograms.
7. Stop the program.

Code:

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

img = cv2.imread('image.jpg', 0)

equalized = cv2.equalizeHist(img)

cv2.imshow('Original Image', img)
cv2.imshow('Equalized Image', equalized)

plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.hist(img.ravel(), 256, [0,256])
plt.title('Original Histogram')

plt.subplot(1,2,2)
plt.hist(equalized.ravel(), 256, [0,256])
plt.title('Equalized Histogram')
plt.show()
```

Result:
Histogram modelling was successfully performed and image contrast was enhanced.

Experiment 4: Edge Detection

Aim:
To detect edges in a digital image using edge detection operators such as Sobel, Prewitt, and Canny using OpenCV.

Software Requirements:
- Python
- OpenCV
- NumPy

Algorithm:
1. Start the program.
2. Import required libraries such as OpenCV and NumPy.
3. Read the input image in grayscale mode.
4. Apply Sobel operator to detect horizontal and vertical edges.
5. Apply Prewitt operator using convolution masks.
6. Apply Canny edge detector for accurate edge detection.
7. Display original and edge-detected images.
8. Stop the program.

Code:

```
import cv2
import numpy as np

img = cv2.imread('image.jpg', 0)

sobel_x = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=3)
sobel_y = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=3)
sobel = cv2.convertScaleAbs(sobel_x + sobel_y)

kernel_x = np.array([[1, 0, -1], [1, 0, -1], [1, 0, -1]])
kernel_y = np.array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]])

prewitt_x = cv2.filter2D(img, -1, kernel_x)
prewitt_y = cv2.filter2D(img, -1, kernel_y)
prewitt = cv2.add(prewitt_x, prewitt_y)

canny = cv2.Canny(img, 100, 200)

cv2.imshow('Original Image', img)
cv2.imshow('Sobel Edge', sobel)
cv2.imshow('Prewitt Edge', prewitt)
cv2.imshow('Canny Edge', canny)
```

Result:
Edges were successfully detected using Sobel, Prewitt, and Canny edge detection methods.

Experiment 5: Morphological Operations

Aim:
To perform morphological operations such as erosion, dilation, opening, and closing on a binary image using OpenCV.

Software Requirements:
- Python
- OpenCV
- NumPy

Algorithm:
1. Start the program.
2. Import required libraries such as OpenCV and NumPy.
3. Read the input image in grayscale mode.
4. Convert the grayscale image into a binary image using thresholding.
5. Create a structuring element (kernel).
6. Apply erosion operation.
7. Apply dilation operation.
8. Apply opening operation.
9. Apply closing operation.
10. Display original and processed images.
11. Stop the program.

Code:

```
import cv2
import numpy as np

img = cv2.imread('image.jpg', 0)

_, binary = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)

kernel = np.ones((5, 5), np.uint8)

erosion = cv2.erode(binary, kernel, iterations=1)
dilation = cv2.dilate(binary, kernel, iterations=1)
opening = cv2.morphologyEx(binary, cv2.MORPH_OPEN, kernel)
closing = cv2.morphologyEx(binary, cv2.MORPH_CLOSE, kernel)
```

```
cv2.imshow('Original Image', img)
cv2.imshow('Binary Image', binary)
cv2.imshow('Erosion', erosion)
cv2.imshow('Dilation', dilation)
cv2.imshow('Opening', opening)
cv2.imshow('Closing', closing)
```

Result:
Morphological operations were successfully performed using OpenCV.

Experiment 6: Image Assessment using NumPy and OpenCV

Aim:
To assess the quality of a digital image by calculating image statistics and quality measures such as mean, standard deviation, MSE, and PSNR using NumPy and OpenCV.

Software Requirements:
- Python
- OpenCV
- NumPy

Algorithm:
1. Start the program.
2. Import required libraries such as OpenCV, NumPy, and math.
3. Read the original and processed images in grayscale mode.
4. Convert images into NumPy arrays.
5. Calculate mean and standard deviation of the image.
6. Compute Mean Squared Error (MSE) between images.
7. Calculate Peak Signal to Noise Ratio (PSNR).
8. Display calculated values.
9. Stop the program.

Code:

```
import cv2
import numpy as np
import math

img1 = cv2.imread('original.jpg', 0)
img2 = cv2.imread('processed.jpg', 0)

mean = np.mean(img1)
std_dev = np.std(img1)

mse = np.mean((img1 - img2) ** 2)

if mse == 0:
    psnr = 100
else:
    psnr = 20 * math.log10(255.0 / math.sqrt(mse))

print('Mean:', mean)
print('Standard Deviation:', std_dev)
print('MSE:', mse)
print('PSNR:', psnr)
```

Result:
Image assessment parameters were successfully calculated using NumPy and OpenCV.

Experiment 7: Feature Detection using OpenCV – Corner Detection

Aim:
To detect corner features in a digital image using Harris Corner Detection method in OpenCV.

Software Requirements:
- Python
- OpenCV
- NumPy

Algorithm:
1. Start the program.
2. Import required libraries such as OpenCV and NumPy.
3. Read the input image and convert it to grayscale.
4. Convert the grayscale image to float32 format.
5. Apply Harris Corner Detection algorithm.
6. Mark detected corners on the original image.
7. Display original and corner-detected images.
8. Stop the program.

Code:

```
import cv2
import numpy as np

img = cv2.imread('image.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

gray = np.float32(gray)

corners = cv2.cornerHarris(gray, 2, 3, 0.04)
corners = cv2.dilate(corners, None)

img[corners > 0.01 * corners.max()] = [0, 0, 255]

cv2.imshow('Corner Detection', img)
```

Result:
Corner features were successfully detected using Harris Corner Detection.

Experiment 8: Image Arithmetic Operations

Aim:
To perform image arithmetic operations such as addition, subtraction, multiplication, and bitwise operations on digital images using OpenCV.

Software Requirements:
- Python
- OpenCV
- NumPy

Algorithm:
1. Start the program.
2. Import required libraries such as OpenCV and NumPy.
3. Read two input images of the same size in grayscale mode.
4. Perform image addition.
5. Perform image subtraction.
6. Perform image multiplication.
7. Perform bitwise operations (AND, OR, XOR).
8. Display original and processed images.
9. Stop the program.

Code:

```
import cv2
import numpy as np

img1 = cv2.imread('image1.jpg', 0)
img2 = cv2.imread('image2.jpg', 0)

add = cv2.add(img1, img2)
sub = cv2.subtract(img1, img2)
mul = cv2.multiply(img1, img2)

bit_and = cv2.bitwise_and(img1, img2)
bit_or = cv2.bitwise_or(img1, img2)
bit_xor = cv2.bitwise_xor(img1, img2)

cv2.imshow('Image 1', img1)
cv2.imshow('Image 2', img2)
cv2.imshow('Addition', add)
cv2.imshow('Subtraction', sub)
cv2.imshow('Multiplication', mul)
cv2.imshow('Bitwise AND', bit_and)
cv2.imshow('Bitwise OR', bit_or)
cv2.imshow('Bitwise XOR', bit_xor)
```

Result:
Image arithmetic operations were successfully performed using OpenCV.

Experiment 9: Image Transformation in OpenCV

Aim:
To perform image transformations such as scaling, translation, and rotation on a digital image using OpenCV.

Software Requirements:
- Python
- OpenCV
- NumPy

Algorithm:
1. Start the program.
2. Import required libraries such as OpenCV and NumPy.
3. Read the input image.
4. Perform image scaling (resizing).
5. Perform image translation using affine transformation.
6. Perform image rotation using rotation matrix.
7. Display original and transformed images.
8. Stop the program.

Code:

```
import cv2
import numpy as np

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

scaled = cv2.resize(img, None, fx=0.5, fy=0.5, interpolation=cv2.INTER_LINEAR)

rows, cols = img.shape[:2]
M_translate = np.float32([[1, 0, 50], [0, 1, 50]])
translated = cv2.warpAffine(img, M_translate, (cols, rows))

M_rotate = cv2.getRotationMatrix2D((cols/2, rows/2), 45, 1)
rotated = cv2.warpAffine(img, M_rotate, (cols, rows))

cv2.imshow('Original Image', img)
cv2.imshow('Scaled Image', scaled)
cv2.imshow('Translated Image', translated)
cv2.imshow('Rotated Image', rotated)
```

Result:
Image transformations were successfully performed using OpenCV.

Experiment 10: Object Detection

Aim:
To detect objects (faces) in an image using Haar Cascade Classifier in OpenCV.

Software Requirements:
- Python
- OpenCV
- NumPy

Algorithm:
1. Start the program.
2. Import required libraries such as OpenCV.
3. Load the pre-trained Haar Cascade classifier.
4. Read the input image and convert it to grayscale.
5. Detect objects using detectMultiScale() method.
6. Draw rectangles around detected objects.
7. Display the detected image.
8. Stop the program.

Code:

```
import cv2
face_cascade = cv2.CascadeClassifier(
    cv2.data.haarcascades + 'haarcascade_frontalface_default.xml'
)

img = cv2.imread('image.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

faces = face_cascade.detectMultiScale(gray, 1.1, 5)

for (x, y, w, h) in faces:
    cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)

cv2.imshow('Object Detection', img)
```

Result:
Objects were successfully detected using Haar Cascade Classifier in OpenCV.