**----------------------------------LAB\_01----------------------------------**

**# convert images from one color-space to another,**

**# like BGR ↔ Gray, BGR ↔ HSV,**

img = cv2.imread("D:\\SLIIT\\4th\_year\\2nd semester\\IUP\LABS\\lab02\\sunflower.jpg", 1)

cv2.imshow('Original Image',img)

**##convert bgr to gray**

imgray =cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

cv2.imshow('Gray Image',imgray)

**#convert to hsv**

imghsv = cv2.cvtColor(img,cv2.COLOR\_BGR2HSV)

cv2.imshow('HSV IMAGE',imghsv)

**# convert to binary image**

res, binary = cv2.threshold(imgray, 120, 255, cv2.THRESH\_BINARY)

cv2.imshow('Binary Image', binary)

cv2.waitKey()

cv2.destroyAllWindows()

**//display images**

fig,ax = plt.subplots(1,4, figsize = (15, 15))

ax[0].imshow(cv2.cvtColor(img, cv2.COLOR\_BGR2RGB))

ax[0].axis('on')

ax[1].imshow(cv2.cvtColor(imgray, cv2.COLOR\_BGR2RGB))

ax[1].axis('off')

ax[2].imshow(cv2.cvtColor(imghsv, cv2.COLOR\_BGR2RGB))

ax[2].axis('on')

ax[3].imshow(cv2.cvtColor(binary, cv2.COLOR\_BGR2RGB))

ax[3].axis('on')

**# 4. Split and merge channels**

b, g, r = cv2.split(image) # Splitting channels

merged\_image = cv2.merge([r, g, b]) # Merging channels back

**# 5. Resize/rotate image**

resized\_image = cv2.resize(image, (width // 2, height // 2)) # Resizing image

rotation\_matrix = cv2.getRotationMatrix2D((width // 2, height // 2), 45, 1) # Rotation matrix

rotated\_image = cv2.warpAffine(image, rotation\_matrix, (width, height)) # Rotating image

**# 6. Draw circle/rectangle/lines**

cv2.circle(image, (200, 200), 50, (0, 255, 0), 2) # Drawing a circle

cv2.rectangle(image, (300, 100), (400, 200), (255, 0, 0), 2) # Drawing a rectangle

cv2.line(image, (100, 100), (400, 400), (0, 0, 255), 2) # Drawing a line

**# 7. Write text on image**

font = cv2.FONT\_HERSHEY\_SIMPLEX

cv2.putText(image, 'OpenCV', (10, 50), font, 2, (255, 255, 255), 2, cv2.LINE\_AA)

**# 8. Convert color to grayscale/binary/negative image**

gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY) # Grayscale

ret, binary\_image = cv2.threshold(gray\_image, 127, 255, cv2.THRESH\_BINARY) # Binary

negative\_image = cv2.bitwise\_not(image) # Negative

**# Displaying images**

cv2.imshow('Original Image', image)

cv2.imshow('Grayscale Image', gray\_image)

cv2.imshow('Binary Image', binary\_image)

cv2.imshow('Negative Image', negative\_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

**----------------------------------LAB\_02----------------------------------**

**#Read image**

img = cv2.imread("D:\\4th yr 2nd sem\\IUP\_Lab\\flower.jpeg", 1)

cv2.imshow('Original Image',img)

**#convert bgr to gray**

imgray =cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

cv2.imshow('Gray Image',imgray)

**#convert to hsv**

imghsv = cv2.cvtColor(img,cv2.COLOR\_BGR2HSV)

cv2.imshow('HSV IMAGE',imghsv)

cv2.waitKey()

cv2.destroyAllWindows()

import cv2

**# Load the video**

video = cv2.VideoCapture("folder path")

**# Read frame by frame**

while video.isOpened():

ret, frame = video.read() **#Video Read**

**# Check if the frame was read successfully**

if not ret:

break **# Exit the loop if no frame is read**

**# Convert BGR to HSV color space**

videoHSV = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)

**# Resize the video frame**

originalVideo = cv2.resize(frame, (400, 400))

cv2.imshow('OriginalVideo', originalVideo)

if cv2.waitKey(1) == ord('q'):

break

video.release()

cv2.destroyAllWindows()

**# Convert BGR to HSV color space**

videoHSV = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)

**# Define the upper range and the lower range of the blue color**

lower\_blue = np.array([110, 50, 50])

upper\_blue = np.array([130, 255, 255])

**# Threshold the HSV image for the range of blue color**

maskVideo = cv2.inRange(videoHSV, lower\_blue, upper\_blue)

**# Extract the blue color object alone**

result = cv2.bitwise\_and(frame, frame, mask=maskVideo)

**----------------------------------LAB\_03----------------------------------**

**//////Histogram/////**

import cv2

import numpy as np

import matplotlib.pyplot as plt

img = cv2.imread("D:\\4th yr 2nd sem\\IUP\_Lab\\flower.jpeg", 0)

**# Find frequency of pixels in range 0 - 255 – histogram Open CV**

hist = cv2.calcHist([img], [0], None, [256], [0, 256])

**# Show the plotting graph of an image**

plt.plot(hist) plt.show()

**# Find frequency of pixels in range 0 - 255 – Histogram numpy**

hist, bins = np.histogram(img.ravel(), 256, [0, 256])

**#Find Blue, Grean, Red colors**

img = cv2.imread("D:\\4th yr 2nd sem\\IUP\_Lab\\flower.jpeg")

color = ('b', 'g', 'r')

for i,col in **enumerate**(color): **#iterate chanels**

histr = cv2.calcHist([img], [i], None, [256], [0,256])

plt.plot(histr, color = col)

plt.xlim([0,256])

plt.show()

**----------------------------------LAB\_04----------------------------------**  
**/////Dark image equalization/////**

dark\_img = cv.imread("D:\\4th yr 2nd sem\\IUP\_Lab\\flower.jpeg", 0)

**# Create histogram of the image**

dark\_img\_hist = cv.calcHist([dark\_img], [0], None, [256], [0, 256])

**# Apply histogram equalization**

equalized\_dark\_img = cv.equalizeHist(dark\_img)

**# Create histogram of the equalized image**

equalized\_dark\_img\_hist = cv.calcHist([equalized\_dark\_img], [0], None, [256], [0, 256])

**# Plot the dark\_img\_hist & equalized\_dark\_img\_hist**

**# : plt.subplot(211) - 1st parameter value means Row, 2nd parameter value means column, 3rd parameter value means image no**

plt.subplot(211), plt.plot(dark\_img\_hist)

plt.subplot(212), plt.plot(equalized\_dark\_img\_hist)

**# Equalized Image window will be appered firstly, After closing that window** **histogram graphs will be displyed**

result = np.hstack((dark\_img, equalized\_dark\_img))

cv.imshow("Equalized Image", result)

cv.waitKey(0)

cv.destroyAllWindows()

**/////color image equalization/////**

color\_img = cv.imread("D:\\4th yr 2nd sem\\IUP\_Lab\\flower.jpeg", 1)

**# Convert image from RGB to HSV**

color\_img\_hsv = cv.cvtColor(color\_img, cv.COLOR\_RGB2HSV)

**# Histogram equalization on V channel**

color\_img\_hsv[:,:,2] = cv.equalizeHist(color\_img\_hsv[:,:,2])

**# Convert image from HSV to RGB**

color\_img\_rgb = cv.cvtColor(color\_img\_hsv, cv.COLOR\_HSV2RGB)

**# Create histogram of the image**

color\_img\_hist = cv.calcHist([color\_img], [0], None, [256], [0, 256])

color\_img\_rgb\_hist = cv.calcHist([color\_img\_rgb], [0], None, [256], [0, 256])

**# Plot the color\_img\_hist & equalized\_color\_img\_hist**

**# : plt.subplot(211) - 1st parameter value means Row, 2nd parameter value means column, 3rd parameter value means image no**

plt.subplot(211), plt.plot(color\_img\_hist)

plt.subplot(212), plt.plot(color\_img\_rgb\_hist)

**# Equalized Image window will be appered firstly, After closing that window histogram graphs will be displyed**

result = np.hstack((color\_img, color\_img\_rgb))

cv.imshow("Equalized Image", result)

cv.waitKey(0)

cv.destroyAllWindows()

**----------------------------------LAB\_05----------------------------------**

**//// Apply negative transformation for color image/////////**

**# 1st Method**

**# negative\_img = 255 - img**

**# 2nd Method**

negative\_img = cv.bitwise\_not(img)

**//// Apply power-low/gamma transformation for color image/////////**

**# Apply 2.2 gamma value**

**# gamma\_2\_2 = np.array(255\*(img/255)\*\*2.2, dtype='uint8')**

gamma1 = 2.2

**# Apply 0.4 gamma value**

**# gamma\_0\_4 = np.array(255\*(img/255)\*\*0.4, dtype='uint8')**

gamma2 = 0.4

gamma\_2\_2 = np.power(img, gamma1)

gamma\_0\_4 = np.power(img, gamma2)

**//// Apply Log transformation for color image/////////**

if img is None:

print("Error loading image")

else:

**# Apply log transformation**

**# Using the second method**

img\_float = img.astype(float) **# Convert to float for precision in log calculation**

log\_img = (np.log(img\_float + 1) / np.log(1 + np.max(img\_float))) \* 255

**# Handle division by zero and other errors silently**

np.seterr(divide='ignore', invalid='ignore')

**# Clip the transformed values to be in the proper range and convert to uint8**

log\_img = np.clip(log\_img, 0, 255).astype(np.uint8)

**# Show the original image**

plt.figure(figsize=(10, 5))

plt.subplot(121)

plt.title("Original Image")

plt.imshow(img, cmap="gray")

plt.axis('off')

**# Show the log transformed image**

plt.subplot(122)

plt.title("Log Transformed Image")

plt.imshow(log\_img, cmap="gray")

plt.axis('off')

plt.show()

**----------------------------------LAB\_06----------------------------------**

**# Apply custom averaging filter (Lowpass Filter) using filter2D()**

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

dst = **cv.filter2D**(noise\_img, -1, kernal)

result = np.hstack((noise\_img, dst))

**# cv.imshow("Filtered Image", result)**

**# cv.waitKey();**

plt.imshow(cv.cvtColor(result, cv.COLOR\_BGR2RGB)) plt.show()

**# Apply averaging filter (Lowpass Filter) : (3 \* 3 filter)**

dst = **cv.blur**(noise\_img, (3, 3))

result = np.hstack((noise\_img, dst))

**# cv.imshow("Filtered Image", result)**

**# cv.waitKey();**

plt.imshow(cv.cvtColor(result, cv.COLOR\_BGR2RGB)) plt.show()

**# Apply averaging filter (Lowpass Filter) : (3 \* 3 box filter)**

dst = **cv.boxFilter**(noise\_img, -1, (3, 3))

result = np.hstack((noise\_img, dst))

**# cv.imshow("Filtered Image", result)**

**# cv.waitKey();**

plt.imshow(cv.cvtColor(result, cv.COLOR\_BGR2RGB)) plt.show()

**# Apply averaging filter : (Median value = 3)**

dst = **cv.medianBlur**(noise\_img, 3)

result = np.hstack((noise\_img, dst))

**# cv.imshow("Filtered Image", result)**

**# cv.waitKey();**

plt.imshow(cv.cvtColor(result, cv.COLOR\_BGR2RGB)) plt.show()

**# Apply averaging filter : (Median value = 3)**

dst = cv.**GaussianBlur**(noise\_img, (11, 11), 0)

result = np.hstack((noise\_img, dst))

**# cv.imshow("Filtered Image", result)**

**# cv.waitKey();**

plt.imshow(cv.cvtColor(result, cv.COLOR\_BGR2RGB)) plt.show()

**# Apply custom averaging filter (Lowpass Filter) using filter2D()**

kernal = np.ones((15, 15), np.float32)/225

dst = cv.filter2D(noise\_img, -1, kernal)

**# Apply thresholding operator to highlight largest object**

ret, thresh = cv.threshold(dst,100,255,cv.THRESH\_BINARY)

**# result = np.hstack((noise\_img, dst, thresh\_img))**

result = np.hstack((noise\_img, dst, thresh))

**# cv.imshow("Filtered Image", result)**

**# cv.waitKey();**

plt.imshow(cv.cvtColor(result, cv.COLOR\_BGR2RGB)) plt.show()

**----------------------------------LAB\_07----------------------------------**

**#Importing Image, ImageFilter and ImageOps modules from PIL package**

from PIL import Image, ImageFilter, ImageOps

**# Load the image**

image = Image.open("D:\\4th yr 2nd sem\\IUP\_Lab\\noise.jpg")

**# Apply Min and Max filter**

min\_filter = image.filter(ImageFilter.MinFilter(3))

max\_filter = min\_filter.filter(ImageFilter.MaxFilter(3))

**# Apply Grayscale and Edge detection**

grayscale = ImageOps.grayscale(max\_filter)

edges = grayscale.filter(ImageFilter.FIND\_EDGES)

**# Show the original and processed images**

plt.imshow(image) plt.show()

plt.imshow(min\_filter) plt.show()

plt.imshow(max\_filter) plt.show()

plt.imshow(grayscale, 'gray') plt.show()

plt.imshow(edges, 'gray') plt.show()

**----------------------------------LAB\_08----------------------------------**

**# Apply Laplacian Filter - 8bits not enough for that - That is why we use cv.CV\_64F for 64bits(Convertian Flag)**

laplacian = cv.Laplacian(img, cv.CV\_64F)

**# Scaling the output - (Ignore negative values)**

laplacianAbs = cv.convertScaleAbs(laplacian)

**# Display the result**

plt.imshow(img, 'gray') plt.show()

plt.imshow(laplacian, 'gray') plt.show()

plt.imshow(laplacianAbs, 'gray') plt.show()

**# Apply GaussianBlur filter**

gaussianBlur = cv.GaussianBlur(img, (3, 3), 0)

**# Apply Laplacian Filter - 8bits not enough for that - That is why we use cv.CV\_64F for 64bits(Convertian Flag)**

laplacian = cv.Laplacian(img, cv.CV\_64F)

laplacianGaussian = cv.Laplacian(gaussianBlur, cv.CV\_64F)

**# Scaling the output - (Ignore negative values)**

laplacianAbs = cv.convertScaleAbs(laplacian)

laplacianAbsGaussian = cv.convertScaleAbs(laplacianGaussian)

**# Display the results**

plt.imshow(img, 'gray') plt.show()

plt.imshow(laplacian, 'gray') plt.show()

plt.imshow(laplacianAbs, 'gray') plt.show()

plt.imshow(gaussianBlur, 'gray') plt.show()

plt.imshow(laplacianGaussian, 'gray') plt.show()

plt.imshow(laplacianAbsGaussian, 'gray') plt.show()

**# Apply Sobel Operator Sobelx vertical sobely horizontal**

sobelX = cv.Sobel(img, cv.CV\_64F, 1, 0, 3) //vertical

sobelY = cv.Sobel(img, cv.CV\_64F, 0, 1, 3) //horizontal

**# Scaling the output - (Ignore negative values)**

sobelXAbs = cv.convertScaleAbs(sobelX)

sobelYAbs = cv.convertScaleAbs(sobelY)

**# Display the result**

plt.imshow(img, 'gray') plt.show()

plt.imshow(sobelX, 'gray') plt.show()

plt.imshow(sobelXAbs, 'gray') plt.show()

plt.imshow(sobelY, 'gray') plt.show()

plt.imshow(sobelYAbs, 'gray') plt.show()

**# Apply the Laplacian operator**

laplacian = cv.Laplacian(img, cv.CV\_64F, ksize=3) **# ksize is the kernel size,** **usually 3, 5, or 7**

**# Convert to absolute values to handle negative edges**

laplacian\_abs = cv.convertScaleAbs(laplacian)

**# Display the original and Laplacian-transformed images**

plt.subplot(1, 2, 1) **# 1 row, 2 columns, 1st subplot**

plt.title("Original Image")

plt.imshow(img, cmap='gray') **# Display the image in grayscale**

plt.subplot(1, 2, 2) # 2nd subplot

plt.title("Laplacian")

plt.imshow(laplacian\_abs, cmap='gray')

plt.show()

**----------------------------------LAB\_09----------------------------------**

img = cv.imread('D:\\4th yr 2nd sem\\IUP\_Lab\\Lab 9 Images-20240426\\binary2.PNG',0)

**#create a structuring elelment**

kernal = np.ones((7,7),np.uint8)

#apply **erosion** operation

erodeImg = cv.erode(img,kernal,iterations = 1)

plt.imshow(img,cmap='gray') plt.show()

plt.imshow(erodeImg,cmap='gray') plt.show()

**#create a structuring elelment**

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

#apply dilation operation

erodeImg = cv.dilate(img,kernal,iterations = 1)

plt.imshow(img,cmap='gray') plt.show()

plt.imshow(erodeImg,cmap='gray') plt.show()

**#opening - erosion followed by dilation**

**#good for removing external noise**

**#create a structuring elelment**

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

**#apply erosion operation**

erodeImg = cv.erode(img,kernal,iterations = 1)

openingImg = cv.dilate(erodeImg,kernal,iterations = 1)

plt.imshow(img,cmap='gray') plt.show()

plt.imshow(openingImg,cmap='gray') plt.show()

**#closing - dilation followed by erotion # This is dilation followed by erosion.**

**It is useful in closing small holes or gaps in an image.**

**#create a structuring elelment**

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

dilatedImg = cv.dilate(img,kernal,iterations = 1)

**#apply erosion operation**

closingImg = cv.erode(dilatedImg,kernal,iterations = 1)

plt.imshow(img,cmap='gray') plt.show()

plt.imshow(dilatedImg,cmap='gray') plt.show()

plt.imshow(closingImg,cmap='gray') plt.show()

**----------------------------------Q1----------------------------------**

import cv2

import matplotlib.pyplot as plt

import numpy as np

**# Load the image in grayscale**

img = cv2.imread('D:\\SLIIT\\4th\_year\\2nd semester\\IUP\\LABS\\lab01\\sunflower.jpg', 0)

**# Calculate histogram**

hist1 = cv2.calcHist([img], [0], None, [256], [0, 256])

**# Calculate the Cumulative Distribution Function (CDF)**

CDF = np.cumsum(hist1, axis=None, dtype=None, out=None) #no of gray levels

**# Constants for histogram equalization**

L = 65536

**# Calculate total number of pixels**

M, N = img.shape

total\_pixels = M \* N

**# Normalize the CDF**

cdf\_normalized = (CDF - CDF.min()) \* (L-1) / (total\_pixels - 1)

**# Apply histogram equalization**

equalized = cv2.equalizeHist(img)

**# Create a negative of the equalized image**

equalized\_neg = cv2.bitwise\_not(equalized)

**# Apply binary thresholding**

\_, binaryimg = cv2.threshold(equalized\_neg, 100, 255, cv2.THRESH\_BINARY)

**# Stack original and binary images horizontally for comparison**

result = np.hstack((img, binaryimg))

**# Display the result**

plt.imshow(result, cmap='gray')

plt.show()

**---------------------------------- Mask ----------------------------------**

import cv2 as cv

import matplotlib.pyplot as plt

import numpy as np

**# Load the image in grayscale**

img = cv.imread('D:\\SLIIT\\4th\_year\\2nd semester\\IUP\\LABS\\lab01\\sunflower.jpg', 0)

**# Compute the histogram**

img\_hist = cv.calcHist([img], [0], None, [256], [0, 256])

**# Compute the Cumulative Distribution Function (CDF) and normalize it**

cdf = np.cumsum(img\_hist, axis=None, dtype=None, out=None)

cdf\_normalized = (cdf - cdf.min()) / (cdf.max() - cdf.min()) \* 255

cdf\_normalized = cdf\_normalized.astype('uint8')

**# Map the original grayscale pixels through the normalized CDF**

img2 = cdf\_normalized[img]

**# Apply histogram equalization**

equalized\_img = cv.equalizeHist(img2)

equalized\_img2\_hist = cv.calcHist([equalized\_img], [0], None, [256], [0, 256])

**# Plot the input and output histograms**

plt.subplot(2, 2, 1)

plt.plot(img\_hist)

plt.title('Original Histogram')

plt.subplot(2, 2, 2)

plt.plot(equalized\_img2\_hist)

plt.title('Equalized Histogram')

**# Extract the image region of interest**

mask = np.zeros(img.shape[:2], dtype=np.uint8)

mask[220:400, 240:390] = 255

masked\_img = cv.bitwise\_and(equalized\_img, equalized\_img, mask=mask)

**# Display the input and output image**

plt.subplot(2, 2, 3)

plt.imshow(img, cmap='gray')

plt.title('Original Image')

plt.subplot(2, 2, 4)

plt.imshow(masked\_img, cmap='gray')

plt.title('Masked Equalized Image')

plt.show()

**----------------------------------** **Contrast stretching----------------------------------**

import cv2 as cv

import numpy as np

import matplotlib.pyplot as plt

**# Load the image**

inputImg = cv.imread('D:\\SLIIT\\4th\_year\\2nd semester\\IUP\\LABS\\q\\image.png', cv.IMREAD\_GRAYSCALE)

**# If the image is not loaded, raise an error**

if inputImg is None:

raise ValueError("Image not loaded correctly")

**# width and height of image**

row, column = inputImg.shape

**# Create an zeros array to store the output image**

outputImg = np.zeros((row, column), dtype=np.uint8)

**# Specify the mini, maxi, mino, maxo values**

mini, maxi = np.percentile(inputImg, (5, 95))

mino, maxo = 0, 255

**# Loop over the input image**

for i in range(row):

for j in range(column):

**# Compute contrast stretching**

outputImg[i, j] = (inputImg[i, j] - mini) \* ((maxo - mino) / (maxi - mini)) + mino

# Compute the input and output histograms

hist\_in, bins\_in = np.histogram(inputImg.flatten(), 256, [0, 256])

hist\_out, bins\_out = np.histogram(outputImg.flatten(), 256, [0, 256])

**# Display the input and output images**

plt.figure(figsize=(10, 8))

plt.subplot(2, 2, 1)

plt.imshow(inputImg, cmap='gray')

plt.title('Input Image (P)')

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

plt.subplot(2, 2, 2)

plt.imshow(outputImg, cmap='gray')

plt.title('Output Image (Q)')

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

**# Display the input and output histograms**

plt.subplot(2, 2, 3)

plt.plot(hist\_in)

plt.title('Input Histogram (R)')

plt.xlim([0, 256])

plt.ylim([0, np.max(hist\_in) + 100])

plt.subplot(2, 2, 4)

plt.plot(hist\_out)

plt.title('Output Histogram (S)')

plt.xlim([0, 256])

plt.ylim([0, np.max(hist\_out) + 100])

plt.tight\_layout()

plt.show()

---------------------------------- **CDF**----------------------------------

Even out the brightness: It makes dark images brighter and bright images darker so details can be seen more clearly.

Improve contrast: It makes the image look better by increasing the difference between light and dark areas, so everything isn't just shades of gray.

Prepare for analysis: It sets up the image nicely for further processing, like finding edges or other important features.

**----------------------------------** **Contrast stretching----------------------------------**

Enhance Details: Make subtle details more visible, especially in areas that were too dark or too light.

Improve Visibility: Enhance the overall sharpness and clarity, making it easier to see features and important information in the image.

Standardize Images: Adjust images to have a similar look, which is important for comparing images or for processing them in batch operations.

Optimize for Display: Make sure the full range of tones can be displayed on a screen or printed, utilizing the medium's full potential.