**DSA 0216**

**COMPUTER VISION WITH OPEN CV**

**DAY 1**

1. Perform basic Image Handling and processing Operations to read an Image and Convert an Image to Gray-scale.

**Code:**

from PIL import Image, ImageDraw

**# Step 1: Generate a sample image**

image = Image.new("RGB", (100, 100), color="red")

draw = ImageDraw.Draw(image)

draw.rectangle([25, 25, 75, 75], fill="blue")

**# Step 2: Convert the image to gray-scale**

gray\_image = image.convert("L")

**# Save the gray-scale image**

gray\_image\_path = "/Computer vision/lab program/gray\_image.jpg"

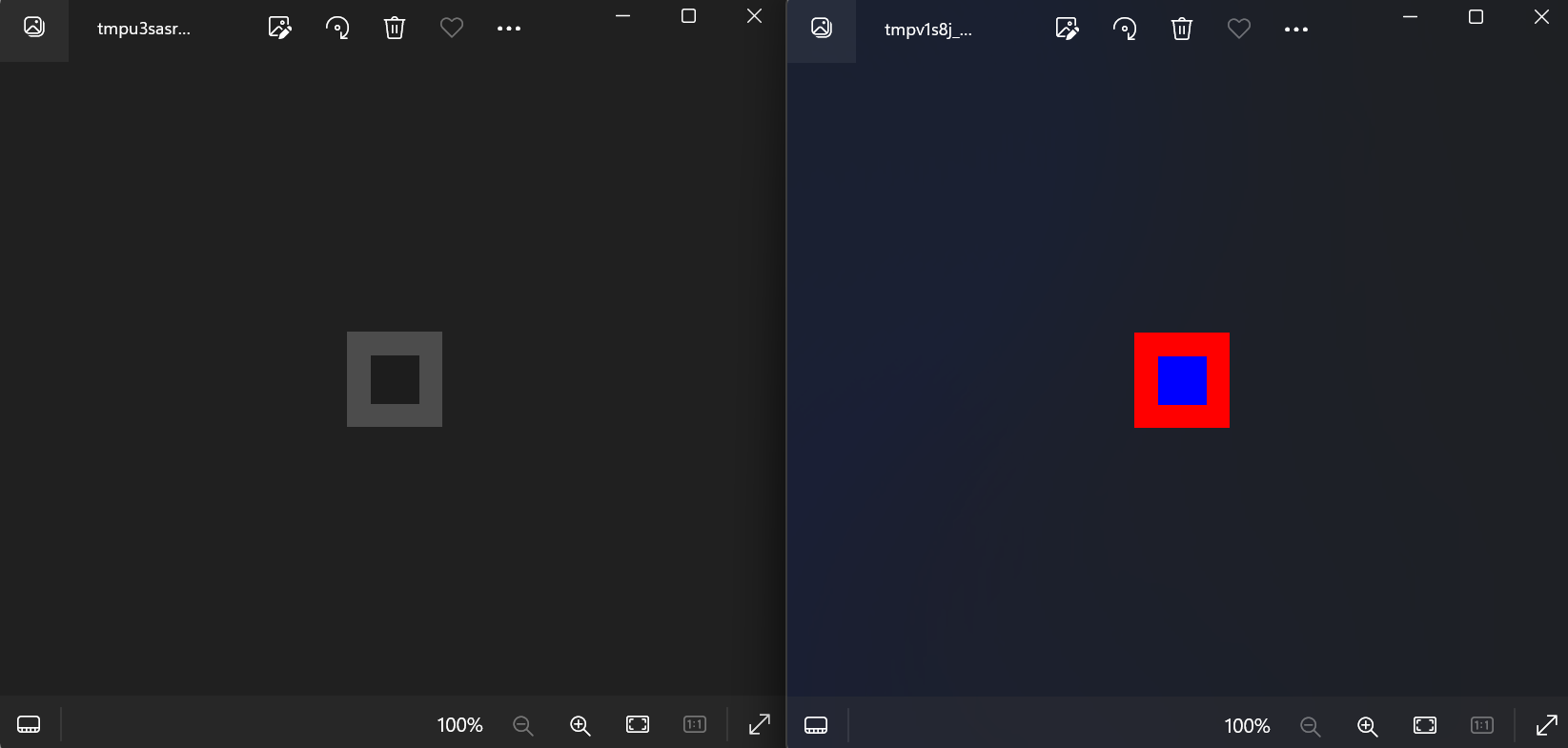
gray\_image.save(gray\_image\_path)

**# Display the images**

image.show(title="Original Image")

gray\_image.show(title="Gray-scale Image")

gray\_image\_path



1. Demonstrate how to read an image in Python and apply a Gaussian Blur to it using Open CV.

**Code:**

import cv2

import numpy as np

from matplotlib import pyplot as plt

**# Generate a sample image**

image = np.zeros((100, 100, 3), dtype=np.uint8)

image[25:75, 25:75] = [255, 0, 0] # Draw a blue square in the middle

**# Step 2: Apply Gaussian Blur**

blurred\_image = cv2.GaussianBlur(image, (15, 15), 0)

**# Display the original and blurred images using matplotlib**

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

**# Original image**

plt.subplot(1, 2, 1)

plt.title("Original Image")

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

plt.axis('off')

**# Blurred image**

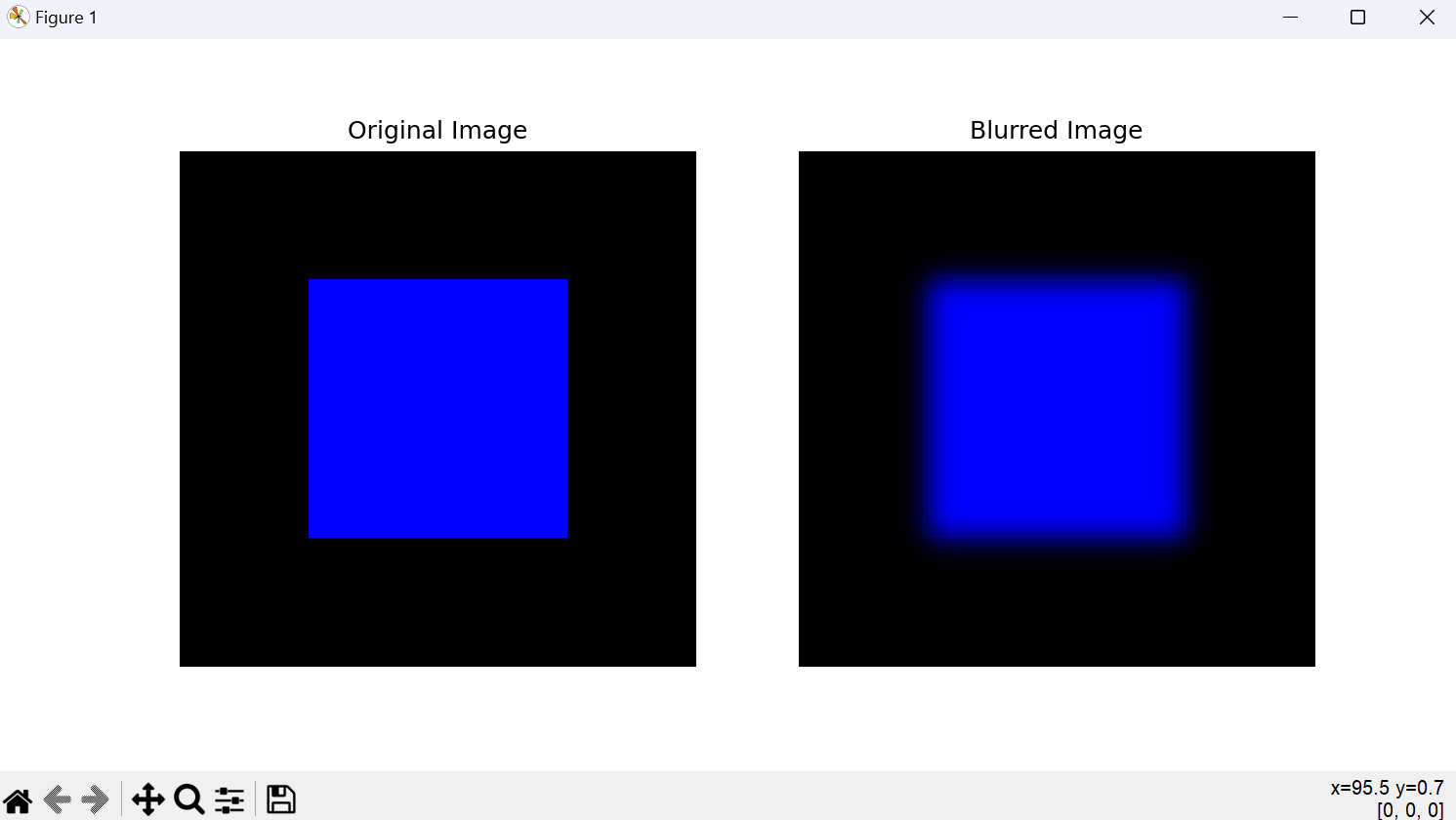
plt.subplot(1, 2, 2)

plt.title("Blurred Image")

plt.imshow(cv2.cvtColor(blurred\_image, cv2.COLOR\_BGR2RGB))

plt.axis('off')

plt.show()



1. Illustrate how you can use OpenCV to resize an image to both larger and smaller dimensions.

**Code:**

import cv2

import numpy as np

from matplotlib import pyplot as plt

**# Generate a sample image**

image = np.zeros((100, 100, 3), dtype=np.uint8)

image[25:75, 25:75] = [255, 0, 0] # Draw a blue square in the middle

**# Get the dimensions of the original image**

original\_height, original\_width = image.shape[:2]

**# Resize the image to smaller dimensions (half the original size**)

smaller\_image = cv2.resize(image, (original\_width // 2, original\_height // 2))

**# Resize the image to larger dimensions (double the original size)**

larger\_image = cv2.resize(image, (original\_width \* 2, original\_height \* 2))

**# Display the original, smaller, and larger images using matplotlib**

plt.figure(figsize=(15, 5))

**# Original image**

plt.subplot(1, 3, 1)

plt.title("Original Image")

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

plt.axis('off')

**# Smaller image**

plt.subplot(1, 3, 2)

plt.title("Smaller Image")

plt.imshow(cv2.cvtColor(smaller\_image, cv2.COLOR\_BGR2RGB))

plt.axis('off')

**# Larger image**

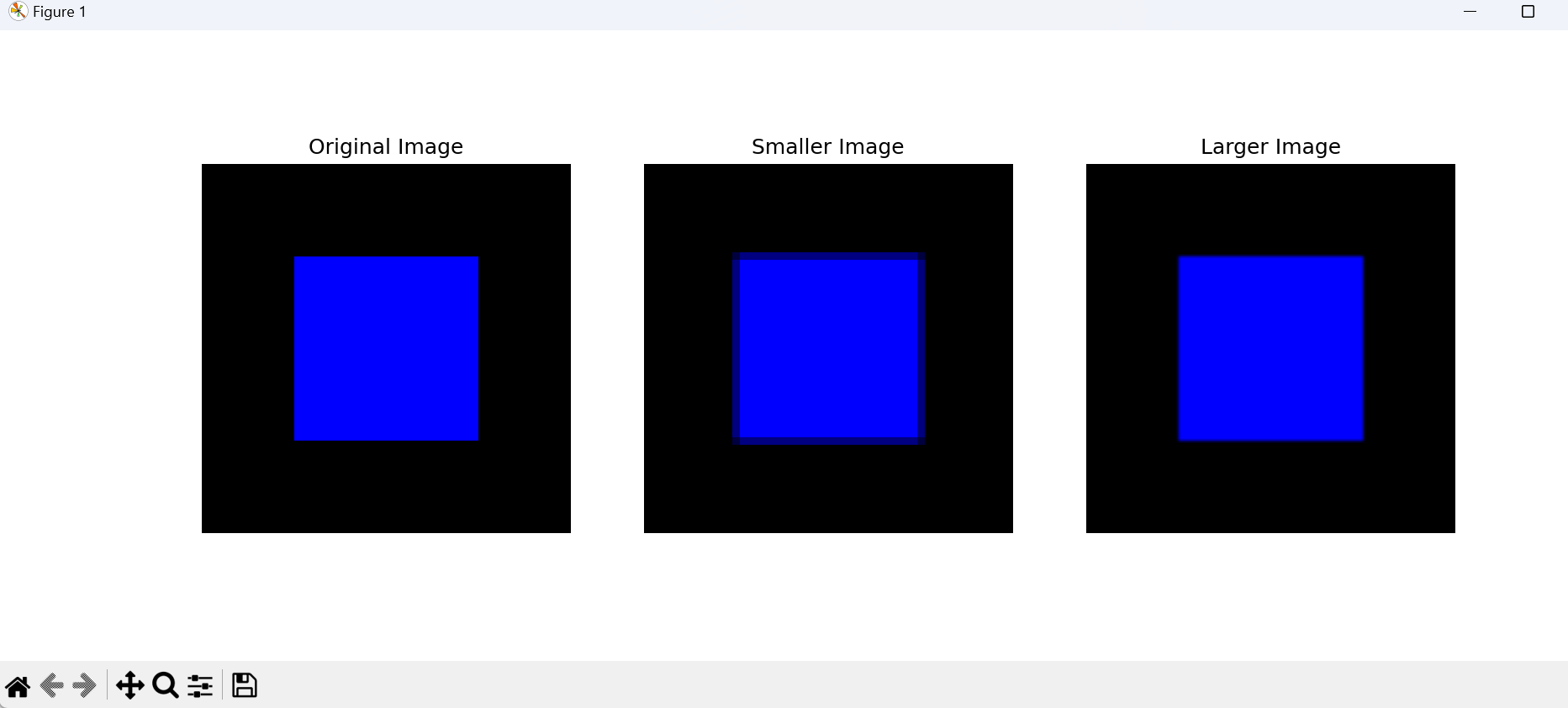
plt.subplot(1, 3, 3)

plt.title("Larger Image")

plt.imshow(cv2.cvtColor(larger\_image, cv2.COLOR\_BGR2RGB))

plt.axis('off')

plt.show()

****

1. Implement histogram equalization on the given image and compare it with the original image using Open CV.

**Code:**

import cv2

import numpy as np

from matplotlib import pyplot as plt

**# Generate a sample image with low contrast**

image = np.zeros((100, 100), dtype=np.uint8)

cv2.rectangle(image, (25, 25), (75, 75), 128, -1) # Draw a gray square in the middle

**# Step 2: Apply histogram equalization**

equalized\_image = cv2.equalizeHist(image)

**# Display the original and equalized images using matplotlib**

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

**# Original image**

plt.subplot(1, 2, 1)

plt.title("Original Image")

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

plt.axis('off')

**# Equalized image**

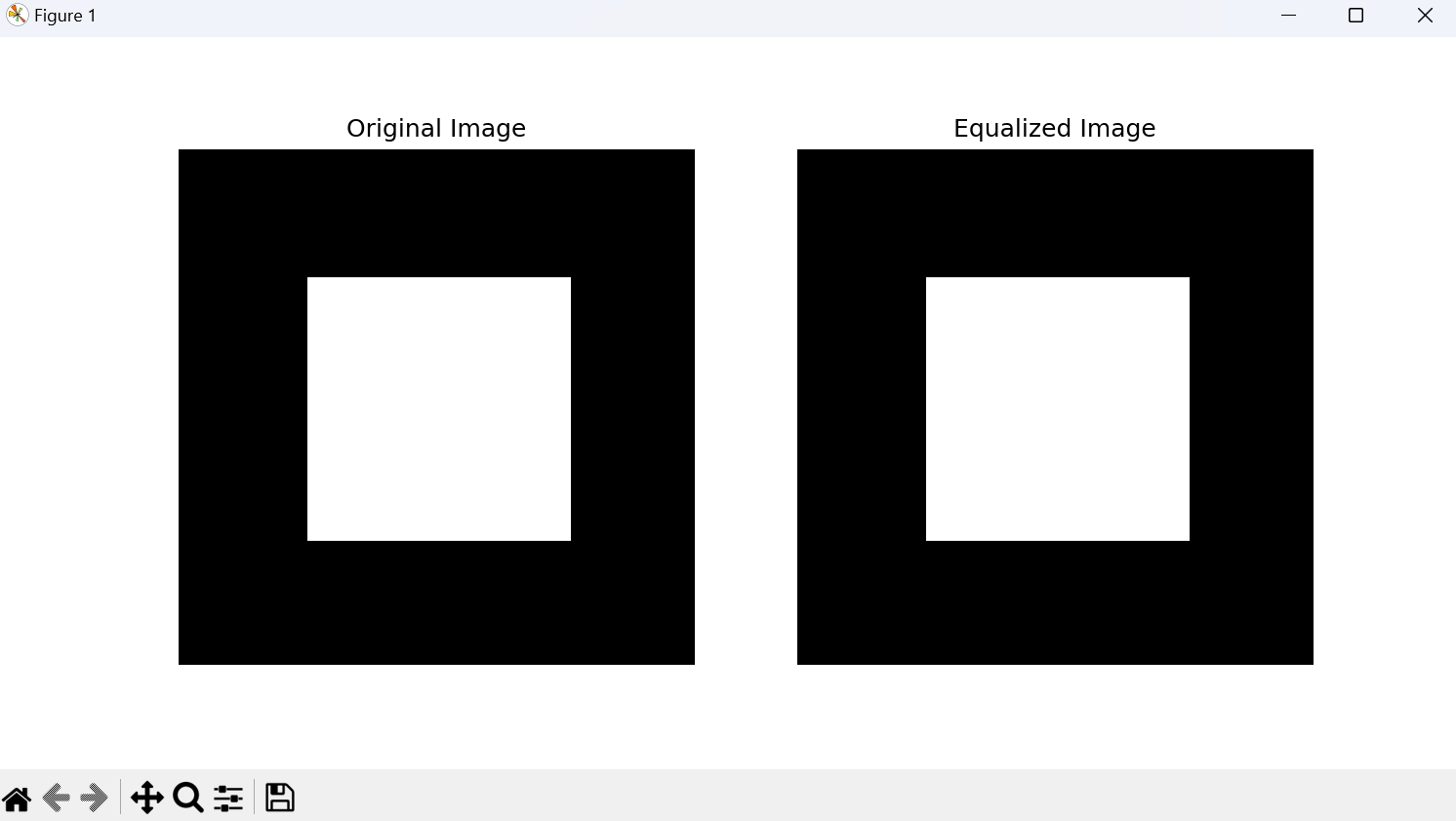
plt.subplot(1, 2, 2)

plt.title("Equalized Image")

plt.imshow(equalized\_image, cmap='gray')

plt.axis('off')

plt.show()



1. Write a Python function that uses OpenCV to analyze and plot the histogram of an image based on its color channels.

**Code:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

**# Load a sample image provided by OpenCV**

image\_path = cv2.samples.findFile('/Computer vision/lab program/gray\_image.jpg') # Adjust 'lena.jpg' if needed

image = cv2.imread(image\_path)

**# Split the image into its BGR components**

channels = cv2.split(image)

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

channel\_names = ('Blue', 'Green', 'Red')

**# Create a figure to display the histograms**

plt.figure()

plt.title('Color Histogram')

plt.xlabel('Bins')

plt.ylabel('Number of Pixels')

**# Iterate over the channels and plot each histogram**

for channel, color, name in zip(channels, colors, channel\_names):

**# Calculate the histogram for the current channel**

histogram = cv2.calcHist([channel], [0], None, [256], [0, 256])

**# Plot the histogram**

plt.plot(histogram, color=color, label=name)

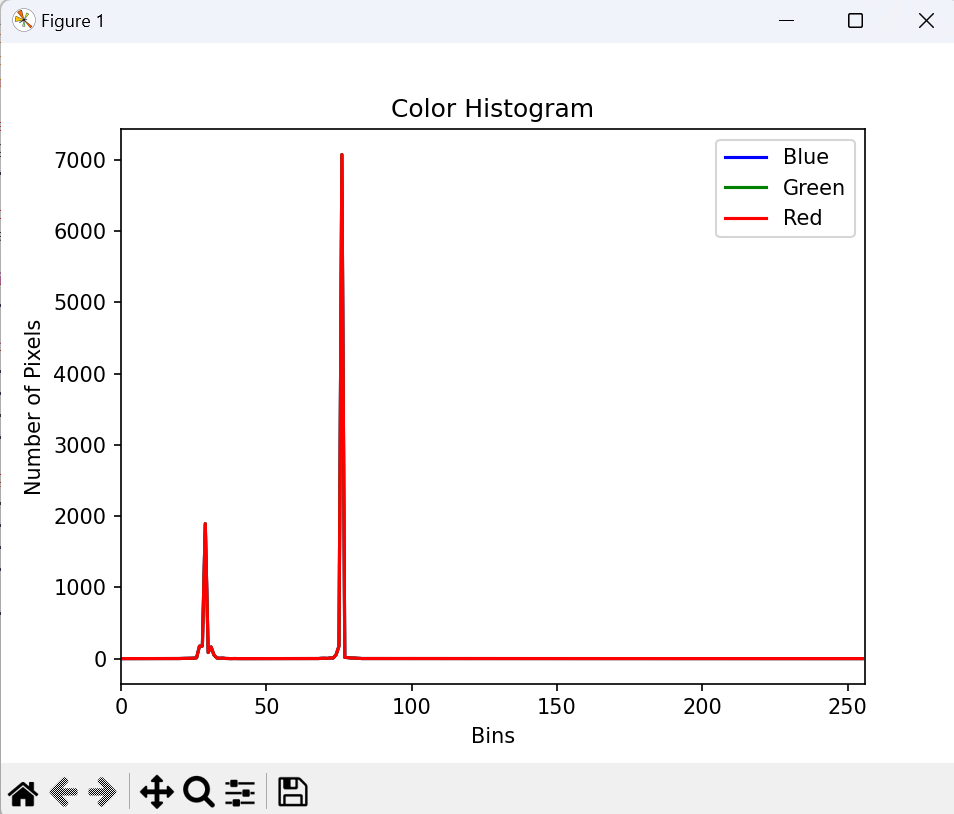
plt.xlim([0, 256])

**# Add a legend**

plt.legend()

**# Show the plot**

plt.show()

****

1. Show how to perform a 90-degree clockwise rotation on an image along the y-axis using OpenCV. Analyze how this transformation impacts the image's spatial orientation

**Code:**

import cv2

import matplotlib.pyplot as plt

**# Load the image**

image\_path = 'D:/photos/Telegram/IMG\_20220707\_084046\_638.jpg' # Replace with your image path

image = cv2.imread(image\_path)

**# Rotate the image 90 degrees clockwise**

rotated\_image = cv2.rotate(image, cv2.ROTATE\_90\_CLOCKWISE)

**# Display the original and rotated images**

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

**# Original Image**

plt.subplot(1, 2, 1)

plt.title("Original Image")

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

plt.axis('off')

**# Rotated Image**

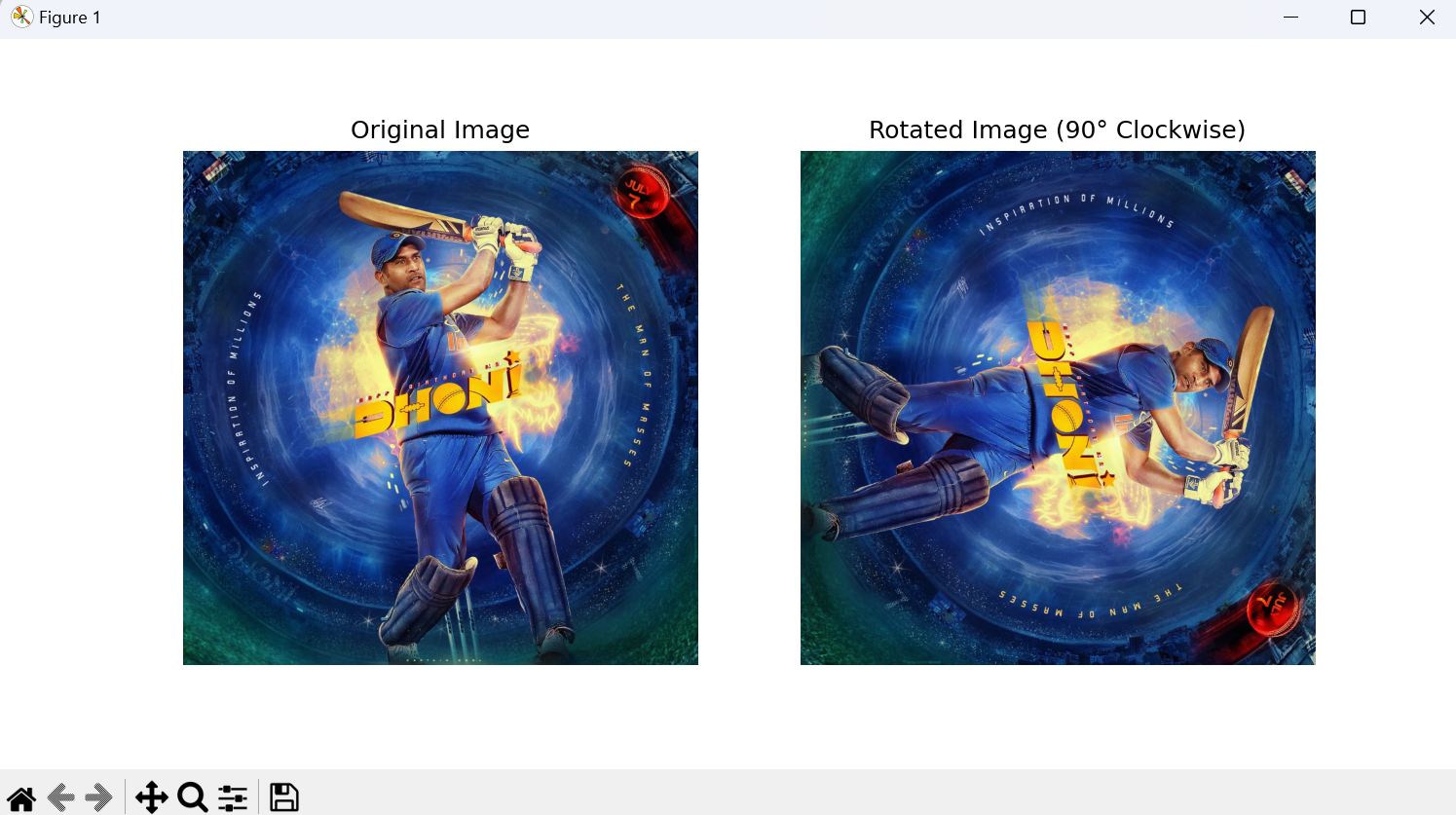
plt.subplot(1, 2, 2)

plt.title("Rotated Image (90° Clockwise)")

plt.imshow(cv2.cvtColor(rotated\_image, cv2.COLOR\_BGR2RGB))

plt.axis('off')

plt.show()



1. Demonstrate the steps to rotate an image 180 degrees clockwise along the y-axis using OpenCV. Discuss the changes observed in the image after this rotation.

**Code:**

import cv2

import matplotlib.pyplot as plt

**# Load the image**

image\_path = 'D:/photos/Telegram/IMG\_20220707\_084046\_638.jpg' # Replace with your image path

image = cv2.imread(image\_path)

**# Rotate the image 180 degrees clockwise**

rotated\_image = cv2.rotate(image, cv2.ROTATE\_180)

**# Display the original and rotated images**

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

**# Original Image**

plt.subplot(1, 2, 1)

plt.title("Original Image")

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

plt.axis('off')

**# Rotated Image**

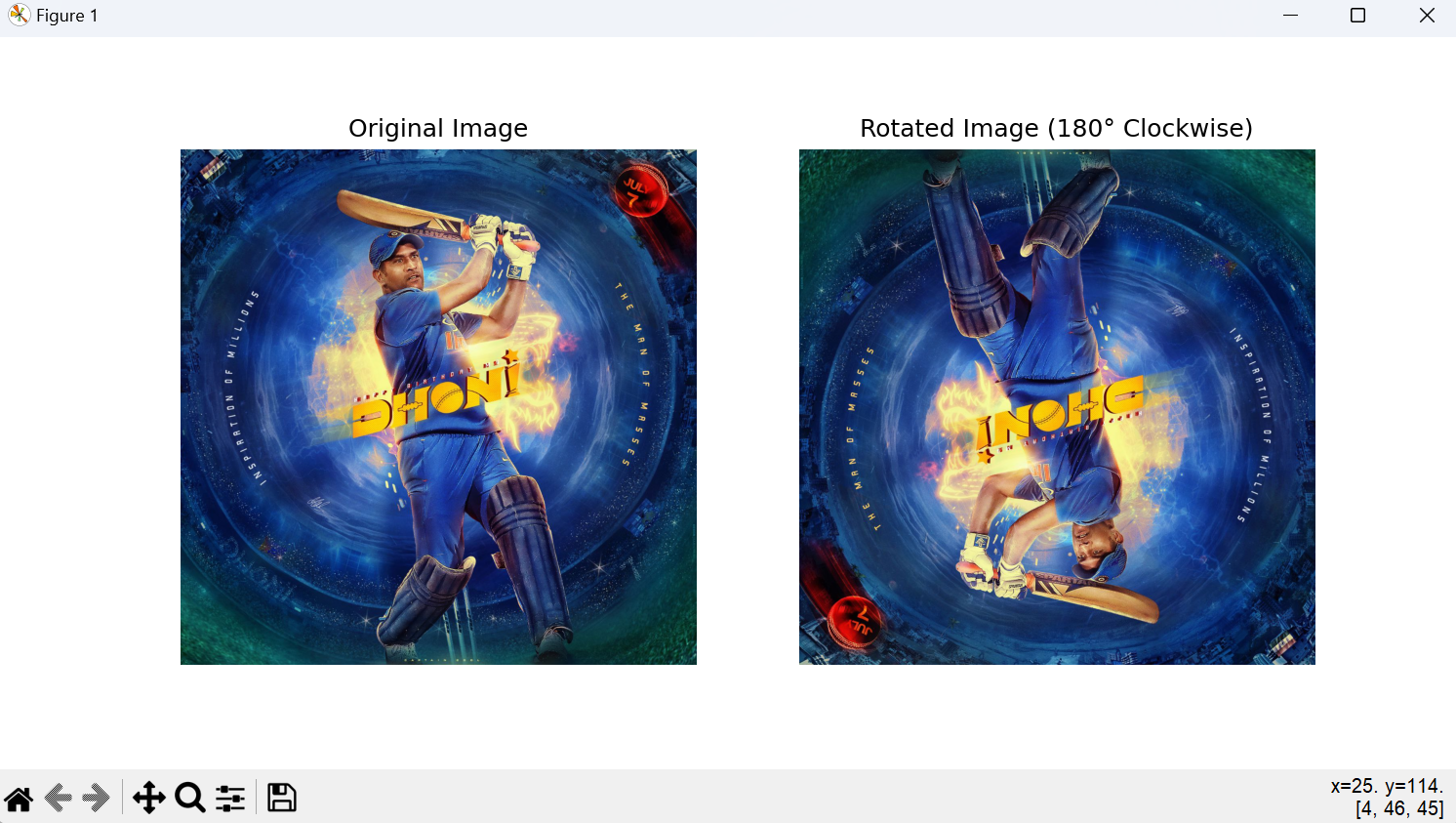
plt.subplot(1, 2, 2)

plt.title("Rotated Image (180° Clockwise)")

plt.imshow(cv2.cvtColor(rotated\_image, cv2.COLOR\_BGR2RGB))

plt.axis('off')

plt.show()



1. Implement a method to rotate an image 270 degrees clockwise along the y-axis using OpenCV. Evaluate how this rotation differs in effect from the 90-degree and 180-degree rotations.

**Code:**

import cv2

import matplotlib.pyplot as plt

**# Load the image**

image\_path = 'D:/photos/Telegram/IMG\_20220707\_084046\_638.jpg' # Replace with your image path

image = cv2.imread(image\_path)

**# Rotate the image 270 degrees clockwise (equivalent to 90 degrees counterclockwise)**

rotated\_image = cv2.rotate(image, cv2.ROTATE\_90\_COUNTERCLOCKWISE)

**# Display the original and rotated images**

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

**# Original Image**

plt.subplot(1, 2, 1)

plt.title("Original Image")

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

plt.axis('off')

**# Rotated Image**

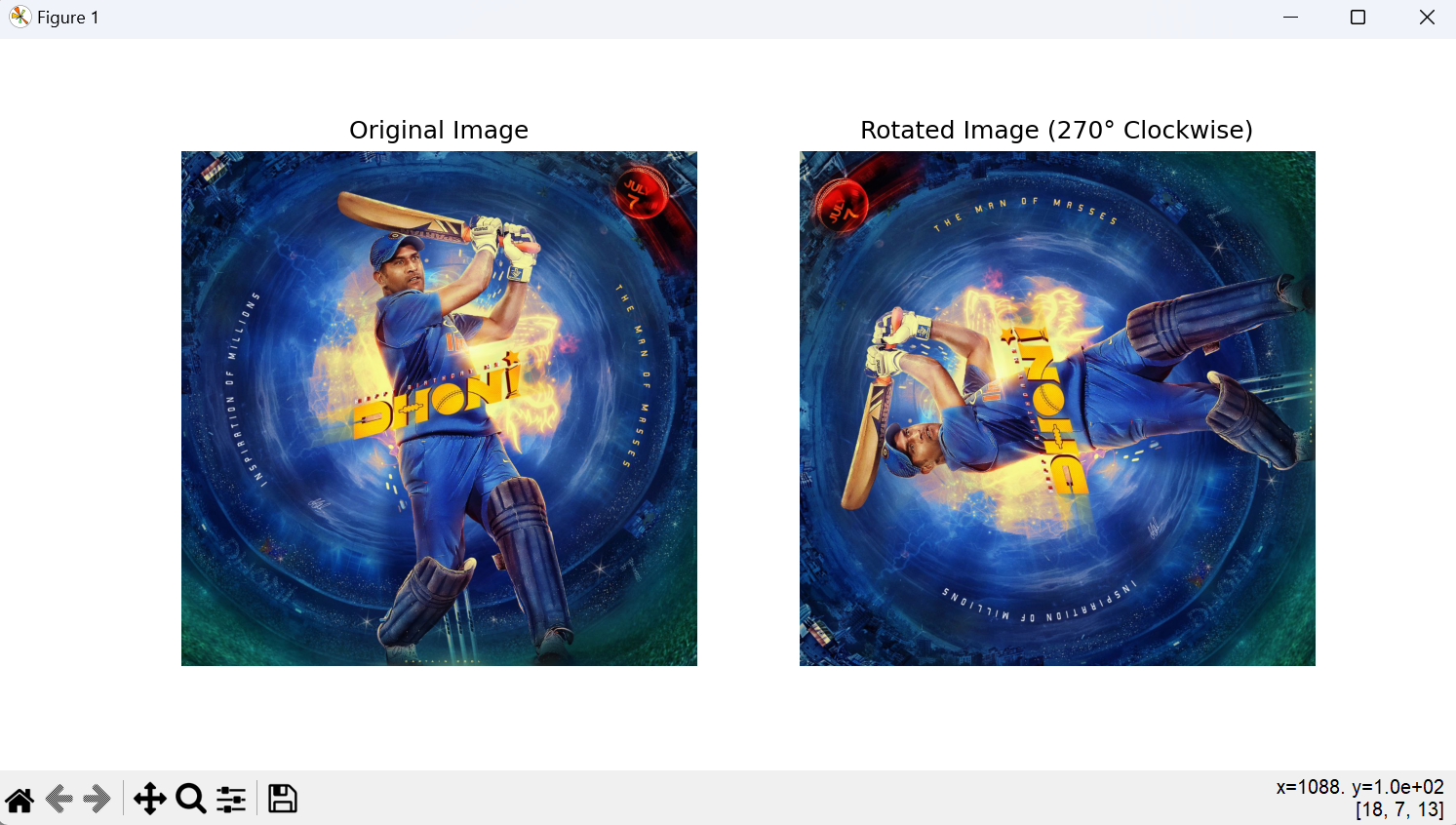
plt.subplot(1, 2, 2)

plt.title("Rotated Image (270° Clockwise)")

plt.imshow(cv2.cvtColor(rotated\_image, cv2.COLOR\_BGR2RGB))

plt.axis('off')

plt.show()



1. Write a Python function to apply erosion to an input image using OpenCV. Evaluate the impact of erosion on the image's structural elements.

**Code:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

**# Load the image in grayscale**

image\_path = '/Computer vision/lab program/gray\_image.jpg' # Replace with your image path

image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

**# Create the kernel for erosion**

kernel\_size = (5, 5) # Adjust the size of the kernel as needed

kernel = np.ones(kernel\_size, np.uint8)

**# Apply erosion to the image**

iterations = 1 # Number of times the erosion is applied

eroded\_image = cv2.erode(image, kernel, iterations=iterations)

**# Display the original and eroded images**

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

**# Original Image**

plt.subplot(1, 2, 1)

plt.title("Original Image")

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

plt.axis('off')

**# Eroded Image**

plt.subplot(1, 2, 2)

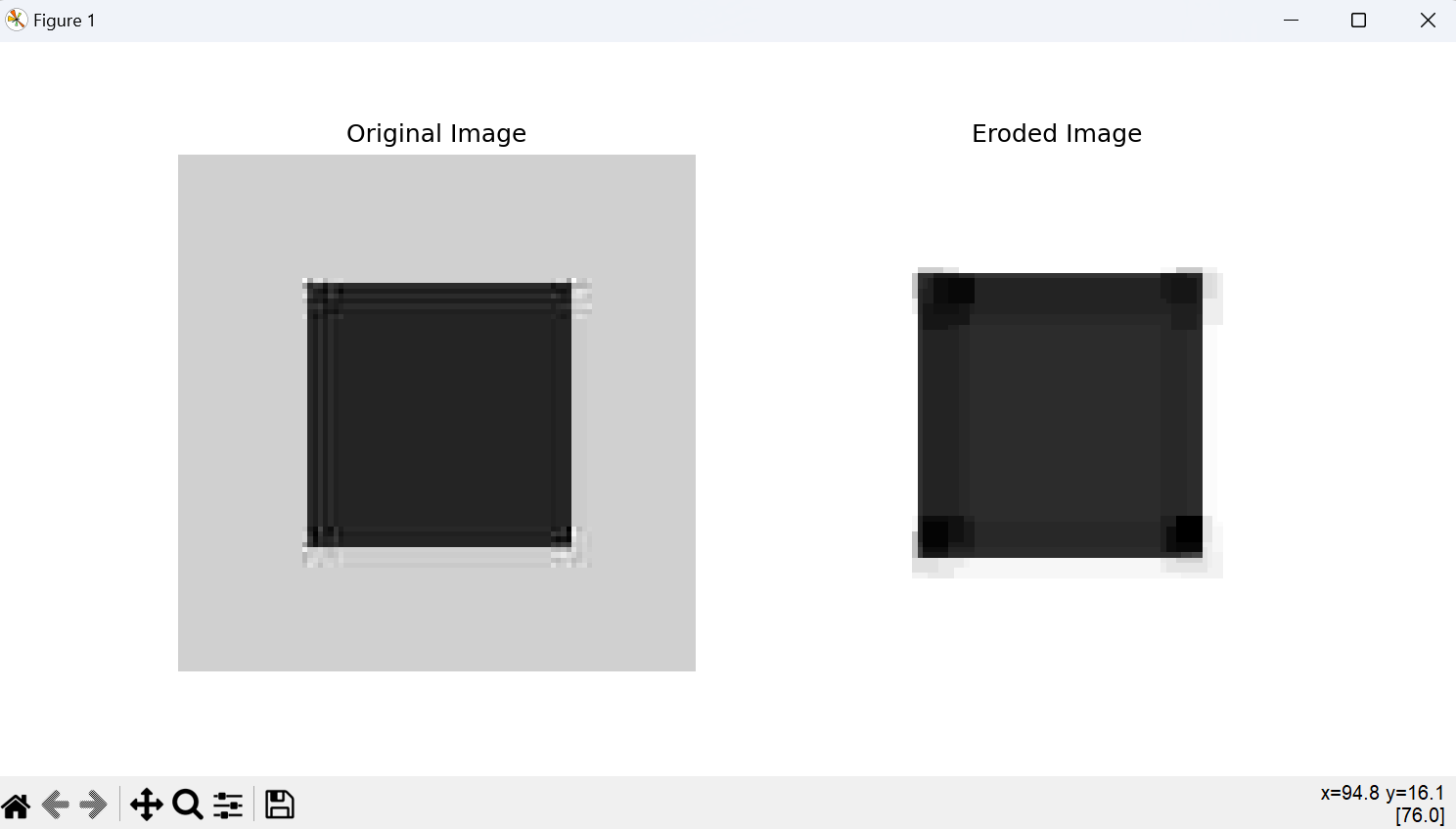
plt.title("Eroded Image")

plt.imshow(eroded\_image, cmap='gray')

plt.axis('off')

**# Show the plot**

plt.show()



1. Explain how to translate (shift) an image using OpenCV in Python. Create a Python function that performs this operation and justify scenarios where image translation might be necessary.

**Code:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

**# Load the image**

image\_path = 'D:/photos/Telegram/IMG\_20220707\_084046\_638.jpg' # Replace with your image path

image = cv2.imread(image\_path)

**# Get the dimensions of the image**

height, width = image.shape[:2]

**# Define the translation distances**

tx, ty = 100, 50 **# Shift right by 100 pixels and down by 50 pixels**

**# Define the translation matrix**

translation\_matrix = np.float32([[1, 0, tx], [0, 1, ty]])

**# Apply the translation to the image**

translated\_image = cv2.warpAffine(image, translation\_matrix, (width, height))

**# Display the original and translated images**

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

**# Original Image**

plt.subplot(1, 2, 1)

plt.title("Original Image")

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

plt.axis('off')

**# Translated Image**

plt.subplot(1, 2, 2)

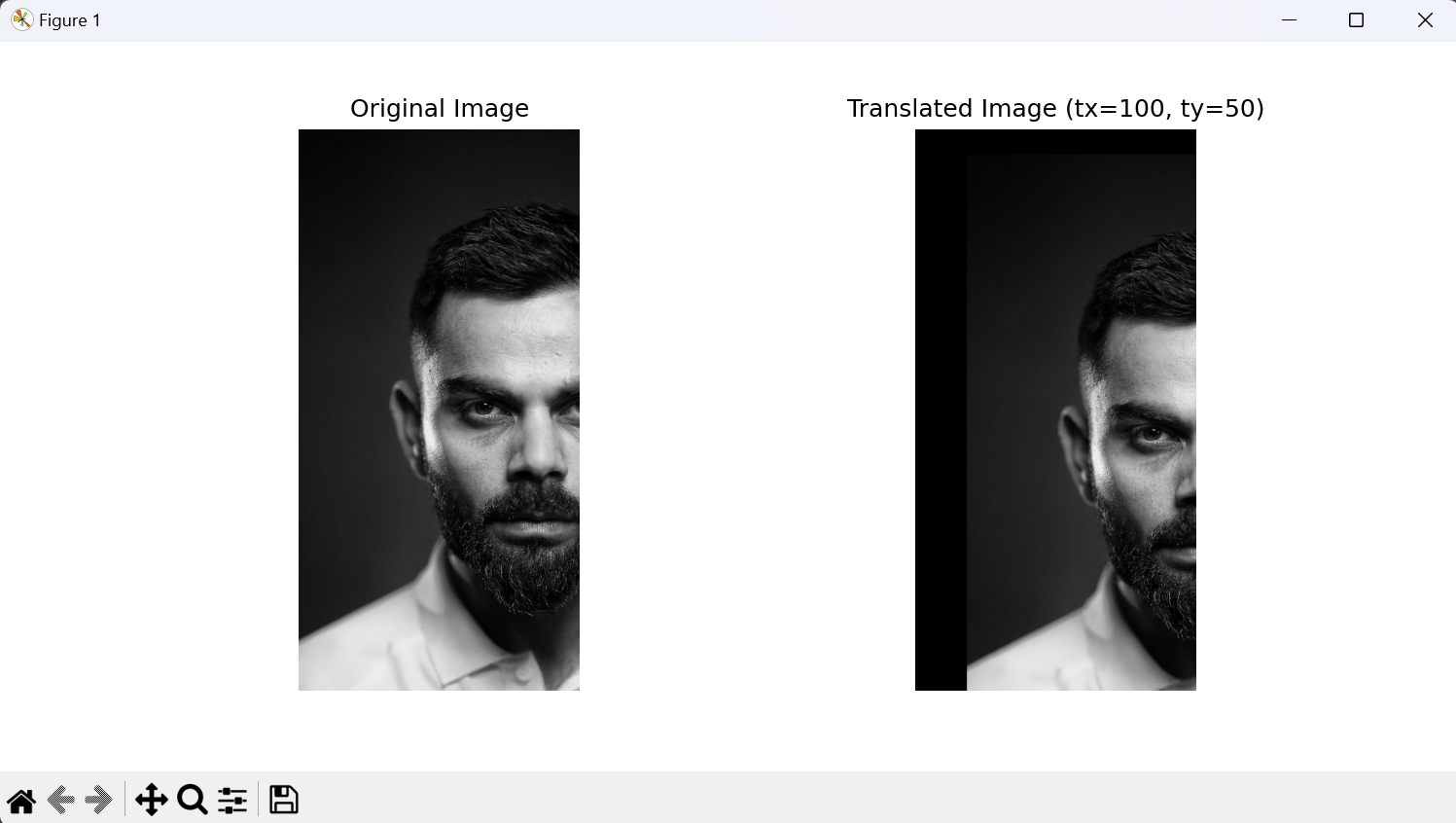
plt.title(f"Translated Image (tx={tx}, ty={ty})")

plt.imshow(cv2.cvtColor(translated\_image, cv2.COLOR\_BGR2RGB))

plt.axis('off')

**# Show the plot**

plt.show()



1. Write a Python Function To Implement smoothing or averaging filter in spatial domain using OpenCV.

**Code:**

import cv2

**# Path to the input image**

image\_path = "E:/Computer vision/lab program/blur image.jpg"

**# Read the input image**

image = cv2.imread(image\_path)

**# Define the kernel size for the averaging filter**

ksize = (5, 5)

**# Apply the averaging filter**

smoothed\_image = cv2.blur(image, ksize)

**# Display the original image**

cv2.imshow('Original Image', image)

**# Display the smoothed image**

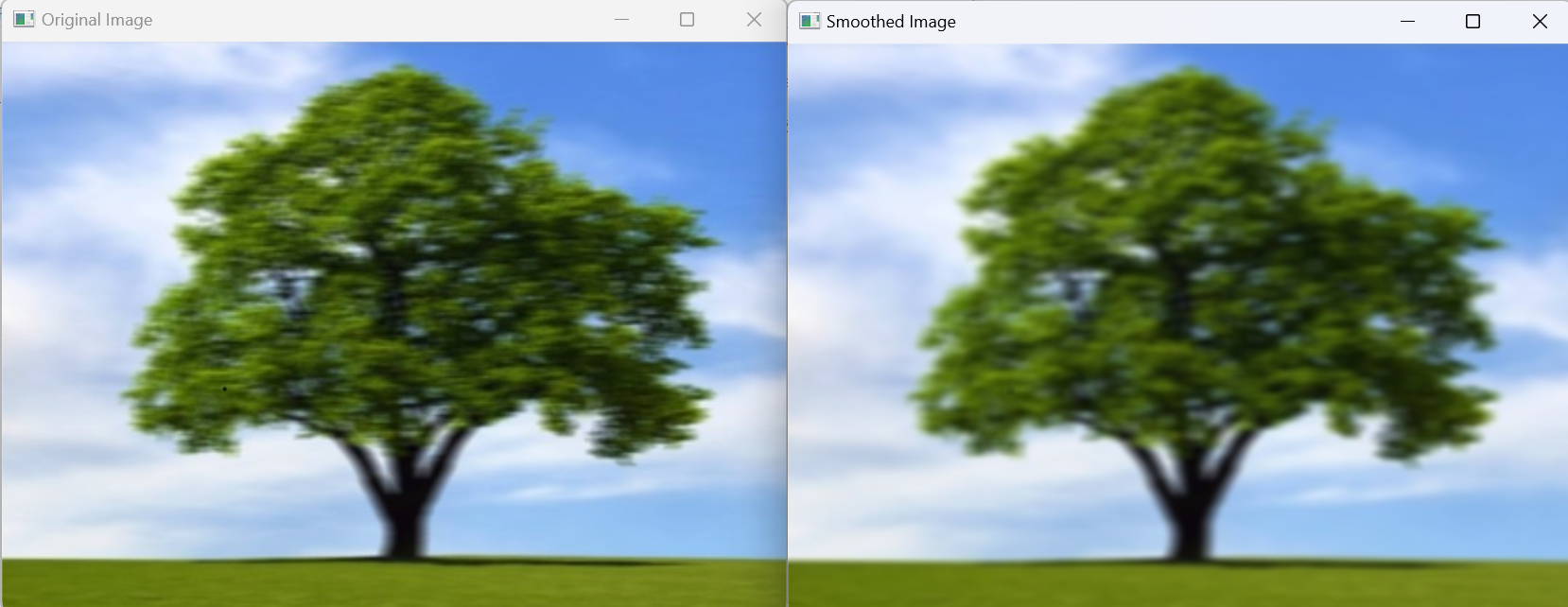
cv2.imshow('Smoothed Image', smoothed\_image)

**# Wait indefinitely until a key is pressed**

cv2.waitKey(0)

**# Destroy all OpenCV windows**

cv2.destroyAllWindows()



1. Demonstrate fill or modify a specific region of interest (ROI) in an image using OpenCV in Python? Describe the steps involved in identifying and processing the ROI.

**Code:**

import cv2

import numpy as np

**# Path to the input image**

image\_path = "E:/Computer vision/lab program/blur image.jpg"

**# Read the input image**

image = cv2.imread(image\_path)

**# Define the ROI coordinates and size**

x, y, w, h = 50, 50, 100, 100 # Example coordinates and dimensions

**# Extract the ROI**

roi = image[y:y+h, x:x+w]

**# Option 1: Fill the ROI with a specific color (e.g., green)**

fill\_color = (0, 255, 0) # Green color in BGR

image[y:y+h, x:x+w] = fill\_color

**# Option 2: Apply a smoothing filter to the ROI**

smoothed\_roi = cv2.blur(roi, (15, 15))

image[y:y+h, x:x+w] = smoothed\_roi

**# Display the original and modified images**

cv2.imshow('Original Image', image)

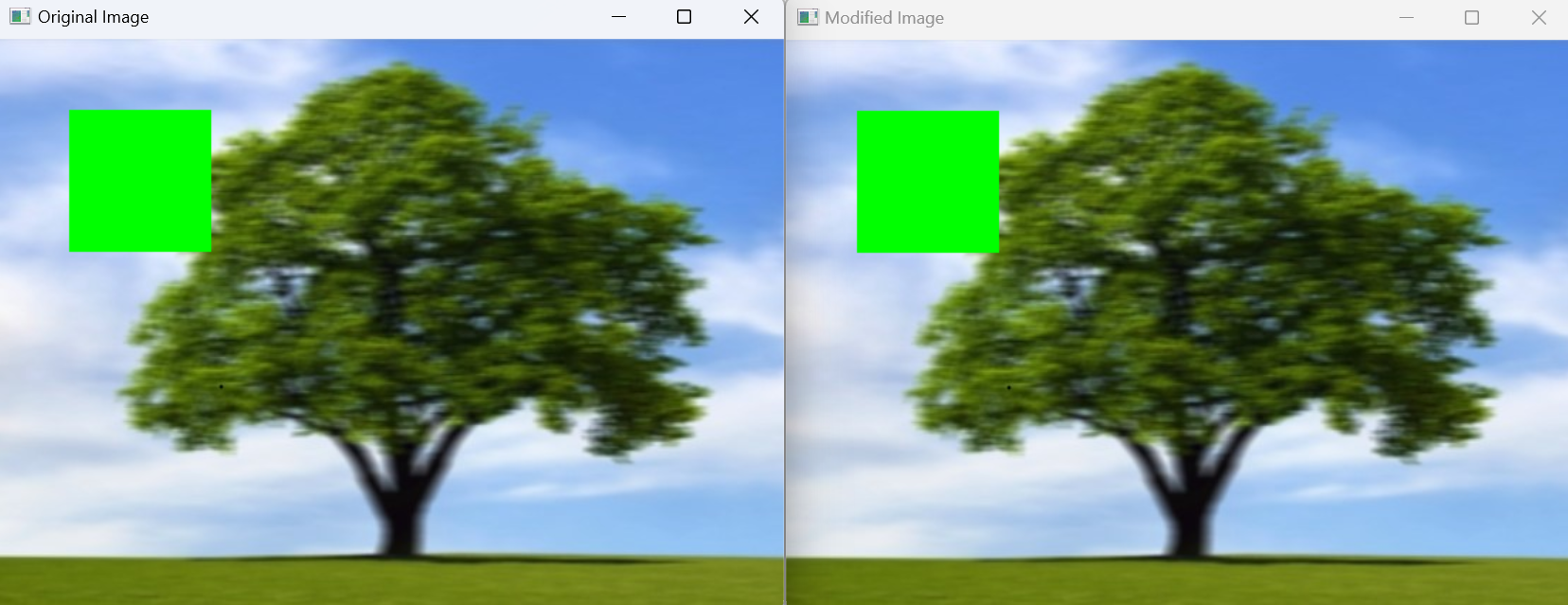
cv2.imshow('Modified Image', image)

**# Wait indefinitely until a key is pressed**

cv2.waitKey(0)

**# Destroy all OpenCV windows**

cv2.destroyAllWindows()



1. Write a python Function to Display the coordinates of the points clicked on the image using Python-OpenCV.

**Code:**

import cv2

**# Load an image**

img = cv2.imread("E:/Computer vision/lab program/blur image.jpg")

**# List to store clicked points**

clicked\_points = []

**# Callback function for mouse events**

def mouse\_callback(event, x, y, flags, param):

global clicked\_points

if event == cv2.EVENT\_LBUTTONDOWN:

clicked\_points.append((x, y))

print(f"Clicked at: ({x}, {y})")

font = cv2.FONT\_HERSHEY\_SIMPLEX

cv2.putText(img, f"{x},{y}", (x, y), font, 0.5, (255, 0, 0), 1, cv2.LINE\_AA)

cv2.imshow('image', img)

**# Display the image**

cv2.imshow('image', img)

**# Set mouse callback function**

cv2.setMouseCallback('image', mouse\_callback)

**# Wait for any key to be pressed**

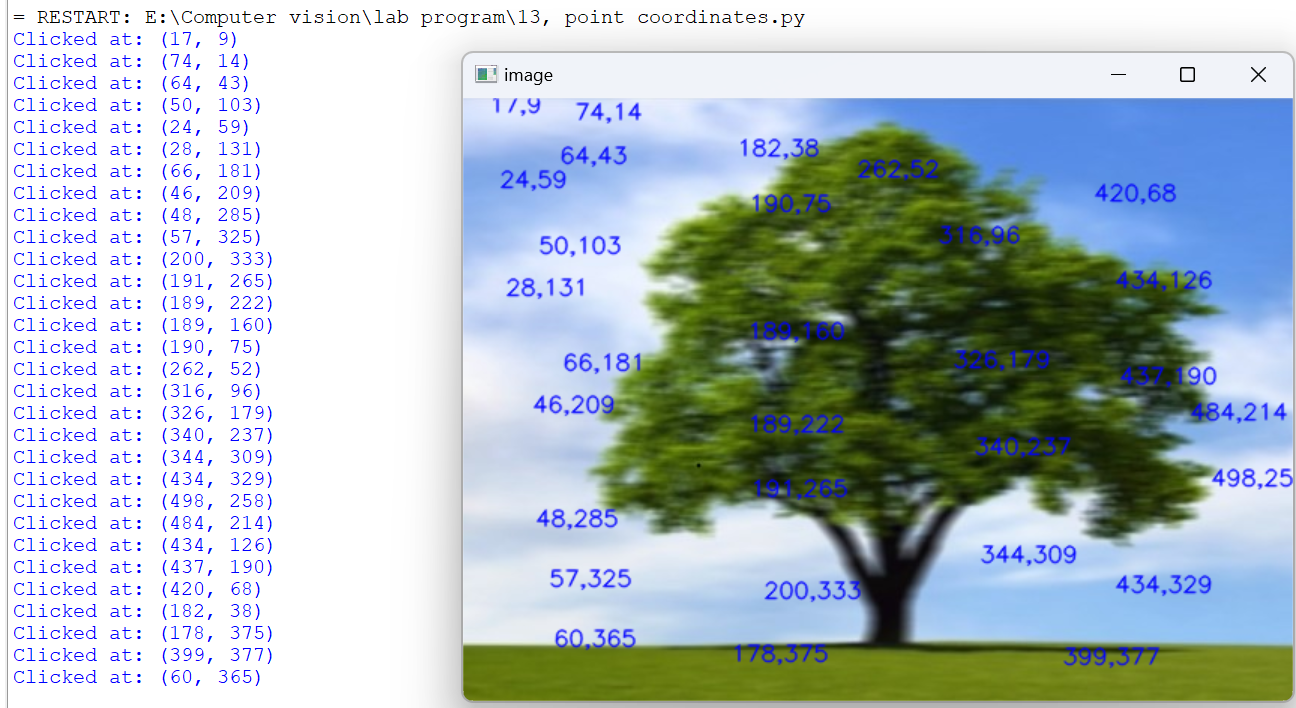
cv2.waitKey(0)

**# Print all clicked points after closing the window (optional)**

print("Clicked points:", clicked\_points)

**# Close all OpenCV windows**

cv2.destroyAllWindows()



1. Write a python Function to implement BGR color palette with trackbars.

**Code:**

import cv2

import numpy as np

**# Create a black image window**

img = 0 \* np.ones((300, 512, 3), np.uint8)

**# Create trackbars for Blue, Green, and Red**

cv2.namedWindow('BGR Color Palette')

cv2.createTrackbar('Blue', 'BGR Color Palette', 0, 255, lambda x: None)

cv2.createTrackbar('Green', 'BGR Color Palette', 0, 255, lambda x: None)

cv2.createTrackbar('Red', 'BGR Color Palette', 0, 255, lambda x: None)

while True:

**# Get current positions of trackbars**

blue = cv2.getTrackbarPos('Blue', 'BGR Color Palette')

green = cv2.getTrackbarPos('Green', 'BGR Color Palette')

red = cv2.getTrackbarPos('Red', 'BGR Color Palette')

**# Set the image color according to the trackbar values**

img[:] = [blue, green, red]

**# Display the image**

cv2.imshow('BGR Color Palette', img)

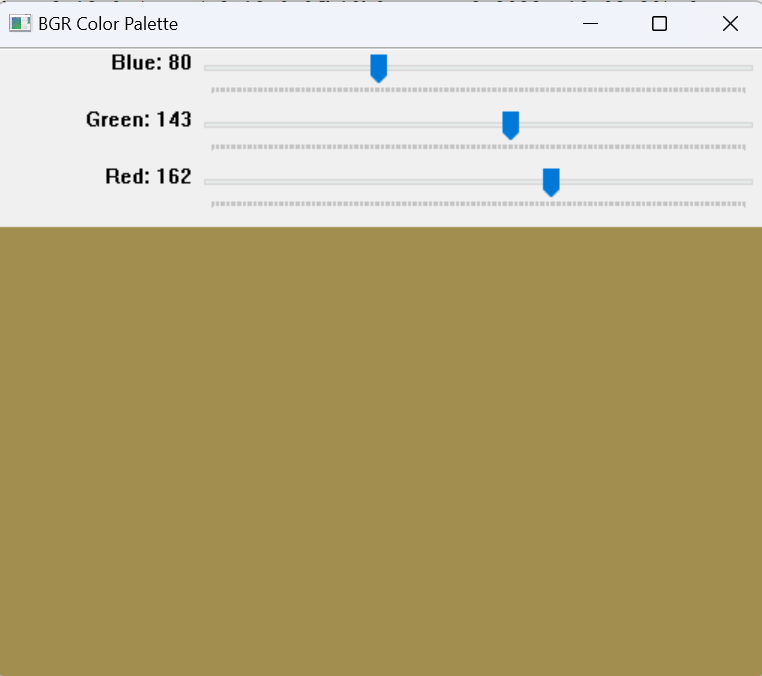
**# Check for ESC key press to exit**

if cv2.waitKey(1) & 0xFF == 27:

break

**# Close all OpenCV windows**

cv2.destroyAllWindows()



# Write a python Function to find the Co-ordinates of Contours using OpenCV.

**Code:**

import cv2

**# Load an image**

img = cv2.imread(“E:/Computer vision/lab program/blur image.jpg'')

**# Convert the image to grayscale**

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

**# Find contours**

contours, \_ = cv2.findContours(gray, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

**# List to store coordinates of all points in contours**

contour\_coordinates = []

**# Iterate through contours**

for contour in contours:

**# Iterate through points in contour**

for point in contour:

x, y = point[0] # point is an array of [[x, y]]

contour\_coordinates.append((x, y))

**# Print coordinates of contours**

print("Coordinates of Contours:")

for coord in contour\_coordinates:

print(coord)

