**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering**

**Computer Engineering Department**

Program: B.Tech. Integrated Sem IX

**Course: Image Processing**

w.e.f. 17st July, 2023

**Faculty:** Abhay Kolhe

LAB Manual

PART A

(PART A : TO BE REFFERED BY STUDENTS)

**Experiment No.01**

**A.1 Aim:**

To explore the requirements for Image Processing using Python functions and basic

Mathematical image operations.

**A.2 Prerequisite:**

1. Understanding of fundamental programming functions/commands and environment of Python,

2. Availability of Soft copy of your Photograph for experiment.

**A.3 Outcome:**

**After successful completion of this experiment students will be able to**

1. Perform basic operations on image(s) using Python functions/structures for programming. Imread, Imshow, dim, size, shape, resize
2. Understand the matrix representation of an Image
3. Write program segment to Perform following mathematical operations on one’s Photograph.
4. Addition of two images
5. Subtraction of one image from other
6. Multiplication of a value with an Image
7. Division of an Image

**A.4 Theory:**

**A.4.1**

**Image Processing Tutorial using OpenCV**

**Introduction:**

In this tutorial, you will explore the basics of image processing using OpenCV, one of the most popular libraries for computer vision tasks. OpenCV provides a wide range of functions and algorithms for manipulating and analyzing images. It will cover essential concepts and demonstrate how to perform common image processing operations using OpenCV in Python.

1. **Installation and Setup**

To begin, make sure you have Python installed on your system. You can install OpenCV using pip by running the following command:

**pip install opencv-python**

1. **Reading and Displaying Images:**

Let's start by reading and displaying an image using OpenCV:

**import cv2**

**# Load an image**

**image = cv2.imread('path\_to\_image.jpg')**

**# Display the image**

**cv2.imshow('Image', image)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

1. **Image Manipulation:** 
   1. **Resizing Images:**

To resize an image, you can use the **resize()** function:

**# Resize the image to a specific width and height**

**resized\_image = cv2.resize(image, (new\_width, new\_height))**

**example**

**import cv2**

**# Read the image**

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

**# Define the desired width and height**

**new\_width = 500**

**new\_height = 300**

**# Resize the image**

**resized\_image = cv2.resize(image, (new\_width, new\_height))**

**# Display the original and resized images**

**cv2.imshow('Original Image', image)**

**cv2.imshow('Resized Image', resized\_image)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

* 1. **Cropping Images:**

To crop a region of interest (ROI) from an image, specify the starting and ending coordinates:

**# Crop a region of interest from the image**

**cropped\_image = image[start\_y:end\_y, start\_x:end\_x]**

**example**

**import cv2**

**# Read the image**

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

**# Define the coordinates of the region of interest (ROI)**

**x = 100 # x-coordinate of the top-left corner of the ROI**

**y = 200 # y-coordinate of the top-left corner of the ROI**

**width = 300 # width of the ROI**

**height = 400 # height of the ROI**

**# Crop the image based on the defined ROI**

**cropped\_image = image[y:y+height, x:x+width]**

**# Display the original and cropped images**

**cv2.imshow('Original Image', image)**

**cv2.imshow('Cropped Image', cropped\_image)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

* 1. **Rotating Images:**

To rotate an image, you can use the **getRotationMatrix2D()** and **warpAffine()** functions:

**# Rotate the image by a certain angle**

**rotation\_matrix = cv2.getRotationMatrix2D(center, angle, scale)**

**rotated\_image = cv2.warpAffine(image, rotation\_matrix, (image.shape[1], image.shape[0]))**

**example**

**import cv2**

**# Read the image**

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

**# Get the image dimensions**

**height, width = image.shape[:2]**

**# Define the rotation angle in degrees**

**angle = 45**

**# Calculate the rotation matrix**

**rotation\_matrix = cv2.getRotationMatrix2D((width/2, height/2), angle, 1)**

**# Apply the rotation to the image**

**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()**

1. **Image Filtering:** 
   1. **Blurring Images:**

To apply blurring or smoothing to an image, you can use various types of blurring filters like Gaussian blur:

**# Apply Gaussian blur to the image**

**blurred\_image = cv2.GaussianBlur(image, (kernel\_size, kernel\_size), sigma)**

**example**

**import cv2**

**# Read the image**

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

**# Apply Gaussian blur to the image**

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

**# Display the blurred image**

**cv2.imshow('Blurred Image', blurred\_image)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

* 1. **Sharpening Images:**

To enhance the details in an image, you can apply sharpening filters like the Laplacian operator:

**# Apply sharpening to the image**

**sharpened\_image = cv2.Laplacian(image, cv2.CV\_64F)**

* 1. **Edge Detection:**

To detect edges in an image, you can use edge detection algorithms like the Canny edge detector:

**# Apply edge detection to the image**

**edges = cv2.Canny(image, threshold1, threshold2)**

**example**

**import cv2**

**# Read the image**

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

**# Convert the image to grayscale**

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

**# Apply Canny edge detection to the grayscale image**

**edges = cv2.Canny(gray\_image, 100, 200)**

**# Display the edges**

**cv2.imshow('Edges', edges)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

1. **Image Thresholding:**
   1. **Simple Thresholding:**

To convert an image into a binary form based on a fixed threshold, you can use simple thresholding:

**# Apply simple thresholding to the image**

**ret, thresholded\_image = cv2.threshold(image, threshold\_value, max\_value, cv2.THRESH\_BINARY)**

**example:**

**import cv2**

**# Read the image as grayscale**

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

**# Apply simple thresholding to the image**

**\_, thresholded\_image = cv2.threshold(image, 127, 255, cv2.THRESH\_BINARY)**

**# Display the thresholded image**

**cv2.imshow('Thresholded Image', thresholded\_image)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

**A.4.2**

**NumPy Tutorial: Introduction to Numerical Computing with Python**

**Introduction:**

NumPy (Numerical Python) is a powerful library for numerical computing in Python. It provides a multidimensional array object, mathematical functions, and tools for working with arrays. In this tutorial, we will explore the basics of NumPy and demonstrate how to perform common operations using its array objects.

1. **Installation and Setup:**

Before you begin, make sure you have NumPy installed. You can install it using pip:

**pip install numpy**

1. **Creating NumPy Arrays:**
   1. **1-Dimensional Arrays:**

To create a 1-dimensional NumPy array, you can pass a Python list or tuple to the **numpy.array()** function:

**import numpy as np**

**# Create a 1-dimensional array**

**arr1 = np.array([1, 2, 3, 4, 5])**

**print(arr1)**

**Ourput:**

**[1 2 3 4 5]**

* 1. **2-Dimensional Arrays:**

To create a 2-dimensional NumPy array, you can pass a nested list or a list of lists to the **numpy.array()** function:

**# Create a 2-dimensional array**

**arr2 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])**

**print(arr2)**

**Output:**

**[[1 2 3]**

**[4 5 6]**

**[7 8 9]]**

* 1. **Array Initialization:**

NumPy provides several functions for initializing arrays with specific values, such as **numpy.zeros()**, **numpy.ones()**, and **numpy.arange()**:

**# Create an array of zeros**

**zeros\_arr = np.zeros((3, 3))**

**print(zeros\_arr)**

**# Create an array of ones**

**ones\_arr = np.ones((2, 4))**

**print(ones\_arr)**

**# Create an array with a range of values**

**range\_arr = np.arange(0, 10, 2)**

**print(range\_arr)**

**output:**

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

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

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

**[[1. 1. 1. 1.]**

**[1. 1. 1. 1.]]**

**[0 2 4 6 8]**

1. **Basic Operations with NumPy Arrays:** 
   1. **Element-wise Operations:**

You can perform element-wise operations on NumPy arrays, such as addition, subtraction, multiplication, and division:

**arr1 = np.array([1, 2, 3])**

**arr2 = np.array([4, 5, 6])**

**# Perform element-wise operations**

**result = arr1 + arr2**

**print(result) # Output: [5 7 9]**

**result = arr1 - arr2**

**print(result) # Output: [-3 -3 -3]**

**result = arr1 \* arr2**

**print(result) # Output: [4 10 18]**

**result = arr1 / arr2**

**print(result) # Output: [0.25 0.4 0.5]**

* 1. **Array Arithmetic:**

NumPy provides various mathematical functions to perform arithmetic operations on arrays, such as **numpy.sum()**, **numpy.mean()**, **numpy.max()**, and **numpy.min()**:

**arr1 = np.array([1, 2, 3, 4, 5])**

**arr2 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])**

**# Perform array arithmetic**

**sum\_result = np.sum(arr1)**

**print(sum\_result) # Output: 15**

**mean\_result = np.mean(arr2)**

**print(mean\_result) # Output: 5.0**

**max\_value = np.max(arr1)**

**print(max\_value) # Output: 5**

**min\_value = np.min(arr2)**

**print(min\_value) # Output: 1**

* 1. **Aggregation Functions:**

NumPy also provides aggregation functions that work along a specific axis of the array, such as **numpy.sum()**, **numpy.mean()**, **numpy.max()**, and **numpy.min()**:

**# Aggregation functions**

**sum\_axis0 = np.sum(arr2, axis=0) # Sum along axis 0 (columns)**

**mean\_axis1 = np.mean(arr2, axis=1) # Mean along axis 1 (rows)**

**max\_axis0 = np.max(arr2, axis=0) # Maximum along axis 0 (columns)**

**min\_axis1 = np.min(arr2, axis=1) # Minimum along axis 1 (rows)**

**import numpy as np**

**# Create a 2-dimensional array**

**arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])**

**# Perform aggregation functions**

**sum\_axis0 = np.sum(arr, axis=0) # Sum along axis 0 (columns)**

**mean\_axis1 = np.mean(arr, axis=1) # Mean along axis 1 (rows)**

**max\_axis0 = np.max(arr, axis=0) # Maximum along axis 0 (columns)**

**min\_axis1 = np.min(arr, axis=1) # Minimum along axis 1 (rows)**

**# Print the results**

**print("Sum along axis 0:", sum\_axis0)**

**print("Mean along axis 1:", mean\_axis1)**

**print("Maximum along axis 0:", max\_axis0)**

**print("Minimum along axis 1:", min\_axis1)**

**output:**

**Sum along axis 0: [12 15 18]**

**Mean along axis 1: [2. 5. 8.]**

**Maximum along axis 0: [7 8 9]**

**Minimum along axis 1: [1 4 7]**

1. **Indexing and Slicing Arrays:**

NumPy arrays support indexing and slicing operations similar to Python lists:

**arr1 = np.array([1, 2, 3, 4, 5])**

**arr2 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])**

**# Indexing and Slicing**

**value = arr1[0]**

**print(value) # Output: 1**

**subarray = arr2[1, :]**

**print(subarray) # Output: [4 5 6]**

**subarray = arr2[:, 1]**

**print(subarray) # Output: [2 5 8]**

**subarray = arr2[1:3, 1:3]**

**print(subarray)**

**Output:**

**[[5 6]**

**[8 9]]**

1. **Shape Manipulation:** 
   1. **Changing Shape:**

You can change the shape of an array using the **numpy.reshape()** function:

**# Changing Shape**

**reshaped\_arr = np.reshape(arr1, (2, 3))**

* 1. **Reshaping Arrays:**

NumPy provides functions like **numpy.flatten()** and **numpy.ravel()** to flatten arrays:

**# Reshaping Arrays**

**flattened\_arr = arr2.flatten()**

**raveled\_arr = arr2.ravel()**

1. **Array Concatenation and Splitting:**
   1. **Concatenation:**

You can concatenate multiple arrays using **numpy.concatenate()** or **numpy.vstack()** for vertical stacking and **numpy.hstack()** for horizontal stacking:

**# Concatenation**

**concatenated\_arr = np.concatenate((arr1, arr2))**

**vertical\_stack = np.vstack((arr1, arr2))**

**horizontal\_stack = np.hstack((arr1, arr2))**

* 1. **Splitting:**

You can split an array into multiple smaller arrays using **numpy.split()** or **numpy.vsplit()** for vertical splitting and **numpy.hsplit()** for horizontal splitting:

**# Splitting**

**split\_arr = np.split(arr1, 2)**

**vertical\_split = np.vsplit(arr2, 3)**

**horizontal\_split = np.hsplit(arr2, 3)**

**#split example**

**import numpy as np**

**# Create a 1-dimensional array**

**arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])**

**# Split the array into three equal-sized parts**

**split\_arr = np.split(arr, 3)**

**# Print the split arrays**

**for subarray in split\_arr:**

**print(subarray)**

**output**

**[1 2 3]**

**[4 5 6]**

**[7 8 9]**

**#vsplit example**

**import numpy as np**

**# Create a 2-dimensional array**

**arr = np.array([[1, 2, 3, 4],**

**[5, 6, 7, 8],**

**[9, 10, 11, 12]])**

**# Vertically split the array into two parts**

**split\_arr = np.vsplit(arr, 2)**

**# Print the split arrays**

**for subarray in split\_arr:**

**print(subarray)**

**output**

**[[1 2 3 4]**

**[5 6 7 8]]**

**[[ 9 10 11 12]]**

**#hsplit example**

**import numpy as np**

**# Create a 2-dimensional array**

**arr = np.array([[1, 2, 3, 4],**

**[5, 6, 7, 8],**

**[9, 10, 11, 12]])**

**# Horizontally split the array into two parts**

**split\_arr = np.hsplit(arr, 2)**

**# Print the split arrays**

**for subarray in split\_arr:**

**print(subarray)**

**output**

**[[ 1 2]**

**[ 5 6]**

**[ 9 10]]**

**[[ 3 4]**

**[ 7 8]**

**[11 12]]**

**A.5 Procedure/Algorithm:**

**A.5.1 TASK 1:**

1. Create new file in python

2. Write a program to Read Image file and display the read image file.

3. Get familiarized with newly Image variable created with Imread () function, Image matrix its content, size, availability of 3 colour planes.

1. Modify the above program and use following functions on the same image and

observe the changes in workspace.

a. dim b. size c. resize d. shape

5. Observe the output and complete PART B of lab manual.

6. Save and close the file and name it as **EXP1\_Task1\_your Roll no.m** or

**EXP1\_Task1\_your Roll no.py** (as applicable)

**TASK 2:**

1. Create a new python file.
2. Read your input images (Your two different photographs).
3. Add two images, observe the output, and write your comments.
4. Subtract one image from another image, observe output, and write your comments.
5. Divide one image by 0.5, 1, 10, 25, 50 and observe output(both workspace and displayed output). Perform division of two images.
6. Multiply one image by 0.5, 0.75, 1, 10, and 20 and observe the output. Perform Multiplication of two images.
7. Complete PART B of lab manual.

Save and close the file and name it as **EXP1\_Task2\_your Roll no.m** or **EXP1\_Task2\_your Roll no.py** (as applicable)

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

***(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case there is no Black board access available)***

|  |  |
| --- | --- |
| Roll No.: C026 | Name: Anirbaan Ghatak |
| Class : B | Batch : B1 |
| Date of Experiment: 19/07/2023 | Date of Submission: ? |
| Grade : |  |

**B.1 Software Code written by student:**

***(Paste your Matlab code completed during the 2 hours of practical in the lab here)***

**B.2 Input and Output:**

**Input Images:**

**Task1:**



**Task2:**

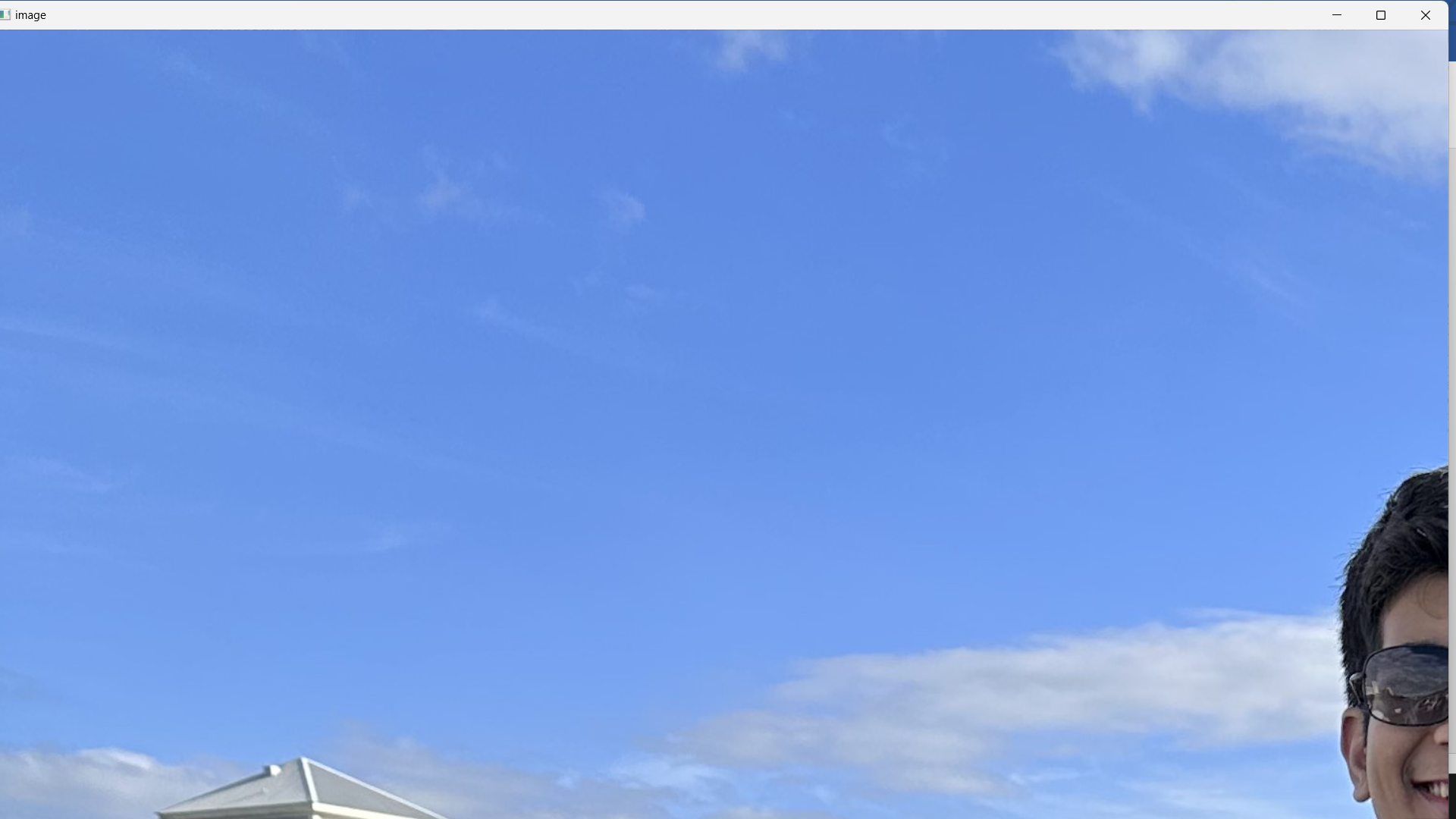


**Output Images:**

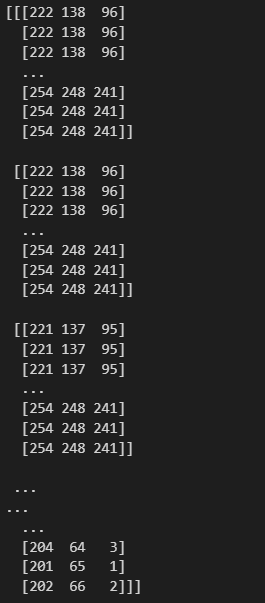
1. **For each functions used to read, show, size, resize, dim, shape the image.**

**TASK1:**

**Imread() and imshow():**

****

**Image Matrix its contents:**

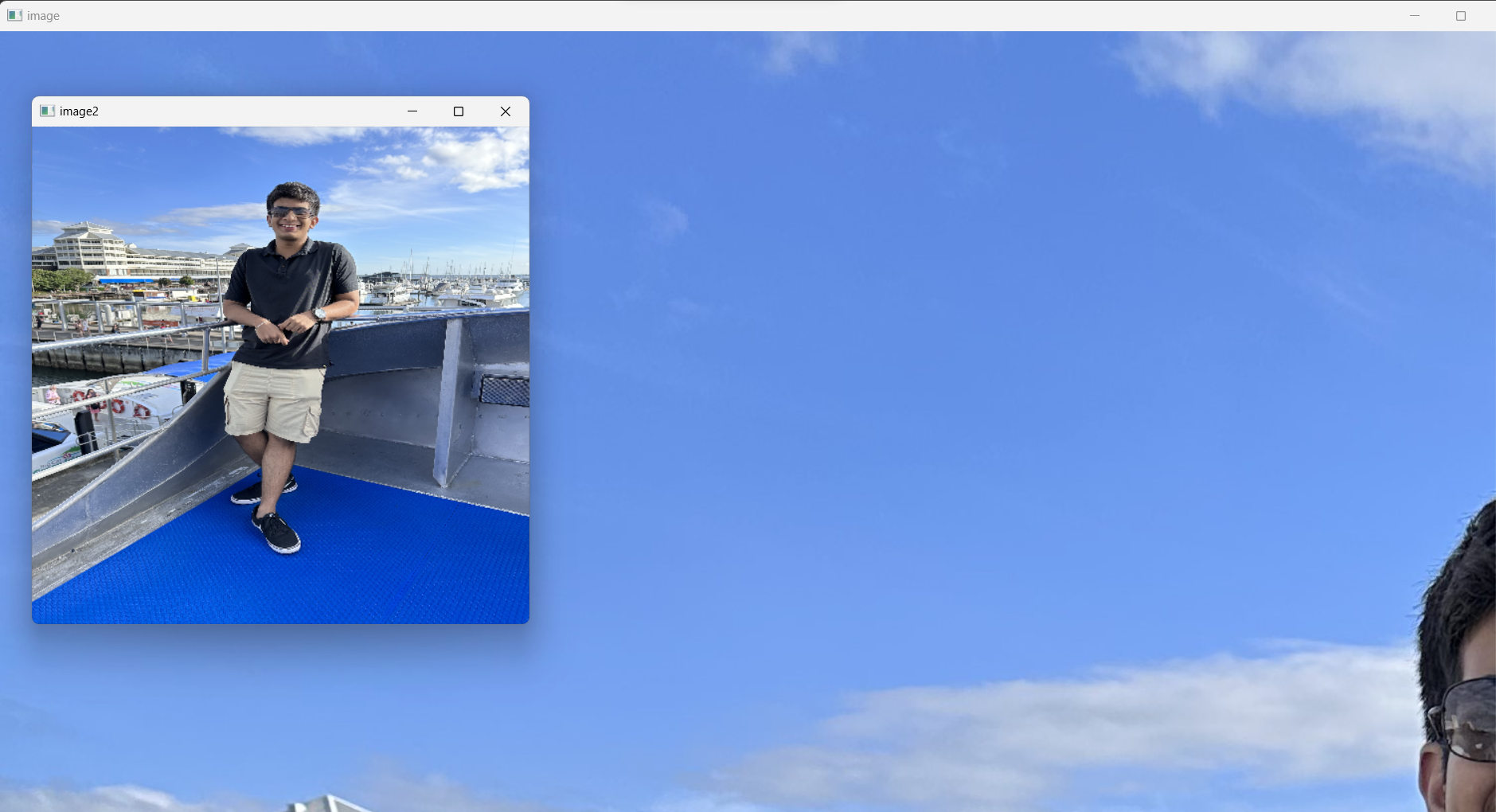
****

**Size:**

****

**Availability of 3 Color Planes:**

**Resize:**

****

1. **For each mathematical operations performed**
2. **For each Matlab/Python statements (if, while, for) used.**
3. **Workspace and matrix representation of an Image (For Matlab only)**

**B.3 Observations and learning:**

***(Students are expected to comment on the output obtained with clear observations and learning for each task/ sub part assigned)***

**B.4 Conclusion:**

*(****Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)***

**B.5 Question of Curiosity**

***(To be answered by student based on the practical performed and learning/observations)***

Q1: List out possible real-life applications of mathematical operations you have performed on the images

Ans:

Determining the dimensions of an image is essential for image processing pipelines as the images should all be of the same size before performing any set of operations on them.

Addition and subtraction are used for various image blending and manipulation tasks. In computer graphics, these techniques are employed to overlay one image on top of another to create effects like transparency or fading. In medical imaging, image subtraction is used for highlighting differences between images taken at different times, such as identifying changes in tumor size over time.

Dividing an image by a scalar values can be used for various purposes. It can be used to control image intensity and brightness, which is useful in adjusting exposure in photography

Q2: What output you can get if you perform following logical operations on images: AND/NAND, OR/NOR, XOR/XNOR and logical NOT. (For Python users – Use cv2.bitwise\_and, cv2.bitwise\_or, cv2.bitwise\_not, cv2.bitwise\_xor)