**CO543: Image Processing**

**Lab 1**

Ranage R.D.P.R. - E/19/310

**1. Implement the following functions on your own without using Python OpenCV.**

The following is the original image that was used to implement the functions in Part 1 of the lab tasks.



As can be seen below, we changed the original image to grayscale before moving on to the routines for task 1.

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

#Funtion to convert image to grayscale

def img\_gray(img):

    return np.mean(img, axis=2)

gray\_img = img\_gray(img)

fig, ax = plt.subplots(1, 2, figsize=(10, 5))

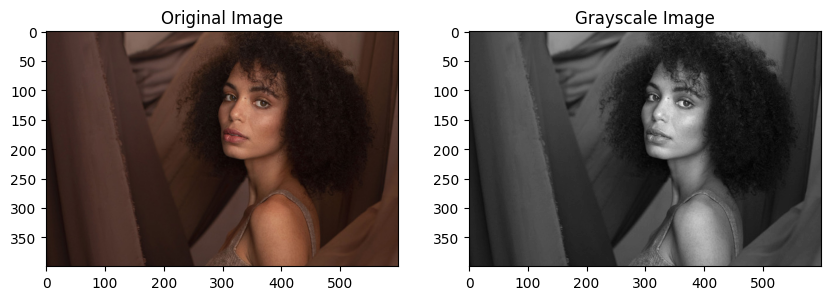
ax[0].imshow(img )

ax[0].set\_title('Original Image')

ax[1].imshow( gray\_img, cmap='gray')

ax[1].set\_title('Grayscale Image')

plt.show()

****

**a). imcomplement(I) - Inverts I**

#Function to invert the image

def imcomplement(img):

    return 1 - img

img\_inv = imcomplement(gray\_img)

fig, ax = plt.subplots(1, 2, figsize=(10, 5))

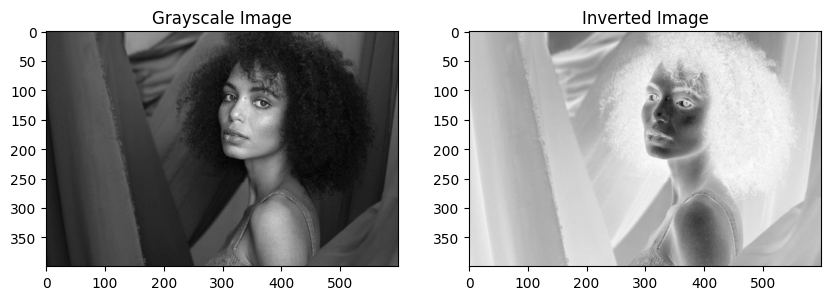
ax[0].imshow(gray\_img, cmap='gray')

ax[0].set\_title('Grayscale Image')

ax[1].imshow( img\_inv, cmap='gray')

ax[1].set\_title('Inverted Image')

plt.show()

****

**b). flipud(I) - Flips image along x-axis**

#Function to flip the image up down

def flipud(img):

    return np.flipud(img)

img\_flipud = flipud(gray\_img)

fig, ax = plt.subplots(1, 2, figsize=(10, 5))

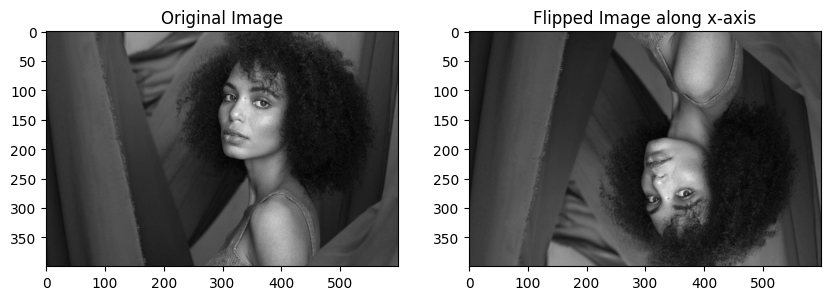
ax[0].imshow(gray\_img, cmap='gray')

ax[0].set\_title('Grayscale Image')

ax[1].imshow( img\_flipud, cmap='gray')

ax[1].set\_title('Flipped Image along x-axis')

plt.show()

****

**c). fliplr(I) - Flips image along y-axis**

#Function to flip the image left right

def fliplr(img):

    return np.fliplr(img)

img\_fliplr = fliplr(gray\_img)

fig, ax = plt.subplots(1, 2, figsize=(10, 5))

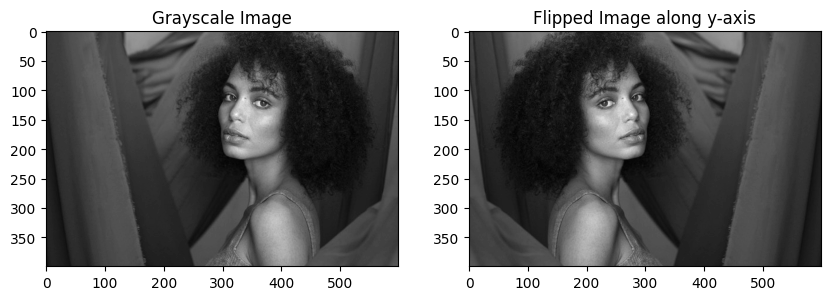
ax[0].imshow(gray\_img, cmap='gray')

ax[0].set\_title('Grayscale Image')

ax[1].imshow( img\_fliplr, cmap='gray')

ax[1].set\_title('Flipped Image along y-axis')

plt.show()

****

**d). imresize(I,[x y]) with nearest-neighbour interpolation**

#Function to resize the image with nearest neighbor interpolation

def imresize(image, new\_shape):

    x\_new, y\_new = new\_shape

    x\_old, y\_old = image.shape

    # Calculate scaling factors

    scale\_x = x\_old / x\_new

    scale\_y = y\_old / y\_new

    # Initialize empty array for the resized image

    resized\_image = np.empty((x\_new, y\_new), dtype=image.dtype)

    # Iterate over each pixel in the new image

    for i in range(x\_new):

        for j in range(y\_new):

            # Find the nearest pixel in the original image

            nearest\_x = int(i \* scale\_x)

            nearest\_y = int(j \* scale\_y)

            # Assign the pixel value from the original image to the resized image

            resized\_image[i, j] = image[nearest\_x, nearest\_y]

    return resized\_image

img\_resize = imresize(gray\_img, (400,900))

fig, ax = plt.subplots(1, 2, figsize=(10, 5))

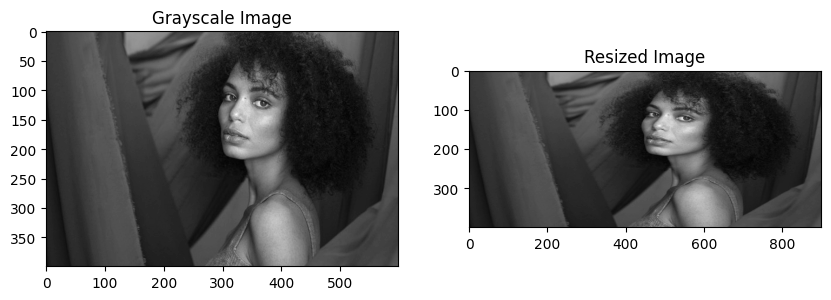
ax[0].imshow(gray\_img, cmap='gray')

ax[0].set\_title('Grayscale Image')

ax[1].imshow( img\_resize, cmap='gray')

ax[1].set\_title('Resized Image')

plt.show()

****

**2. Implement the 4 geometric transformation functions using OpenCV in addition to the given example.**

**a). Translation Transformation**

import cv2 as cv

import numpy as np

from matplotlib import pyplot as plt

rows, cols = gray\_img.shape

# Define the Translation Transformation

tx = 50

ty = 100

M = np.float32([[1, 0, tx], [0, 1, ty]])

# Perform the Transformation

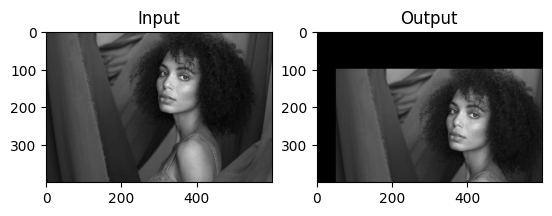
dst = cv.warpAffine(gray\_img, M, (cols, rows))

# Visualize the original and the transformed images

plt.subplot(121), plt.imshow(gray\_img, cmap='gray'), plt.title('Input')

plt.subplot(122), plt.imshow(dst, cmap='gray'), plt.title('Output')

plt.show()

****

**b). Similarity Transformation**

# Define the Similarity Transformation

scale\_factor = 0.5

rotation\_angle = 30

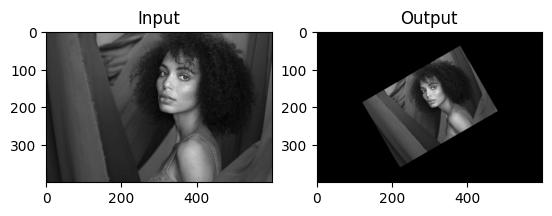
M = cv.getRotationMatrix2D((cols/2, rows/2), rotation\_angle, scale\_factor)

dst = cv.warpAffine(gray\_img, M, (cols, rows))

plt.subplot(121), plt.imshow(gray\_img, cmap='gray'), plt.title('Input')

plt.subplot(122), plt.imshow(dst, cmap='gray'), plt.title('Output')

plt.show()

****

**c). Euclidean Transformation**

# Define the Euclidean Transformation

rotation\_angle = 30

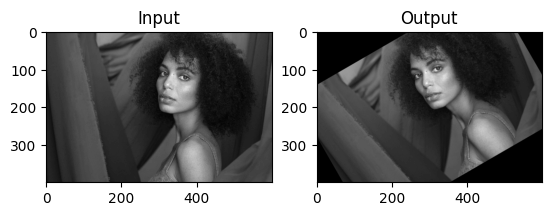
M = cv.getRotationMatrix2D((cols/2, rows/2), rotation\_angle, 1)

dst = cv.warpAffine(gray\_img, M, (cols, rows))

plt.subplot(121), plt.imshow(gray\_img, cmap='gray'), plt.title('Input')

plt.subplot(122), plt.imshow(dst, cmap='gray'), plt.title('Output')

plt.show()

****

**d). Projective Transformation**

# Define the Projective Transformation

pts1 = np.float32([[56, 65], [368, 52], [28, 387], [389, 390]])

pts2 = np.float32([[0, 0], [300, 0], [0, 300], [200, 300]])

M = cv.getPerspectiveTransform(pts1, pts2)

dst = cv.warpPerspective(gray\_img, M, (cols, rows))

plt.subplot(121), plt.imshow(gray\_img, cmap='gray'), plt.title('Input')

plt.subplot(122), plt.imshow(dst, cmap='gray'), plt.title('Output')

plt.show()

