Practical no.: 1

Aim: Perform Geometric Transformations.

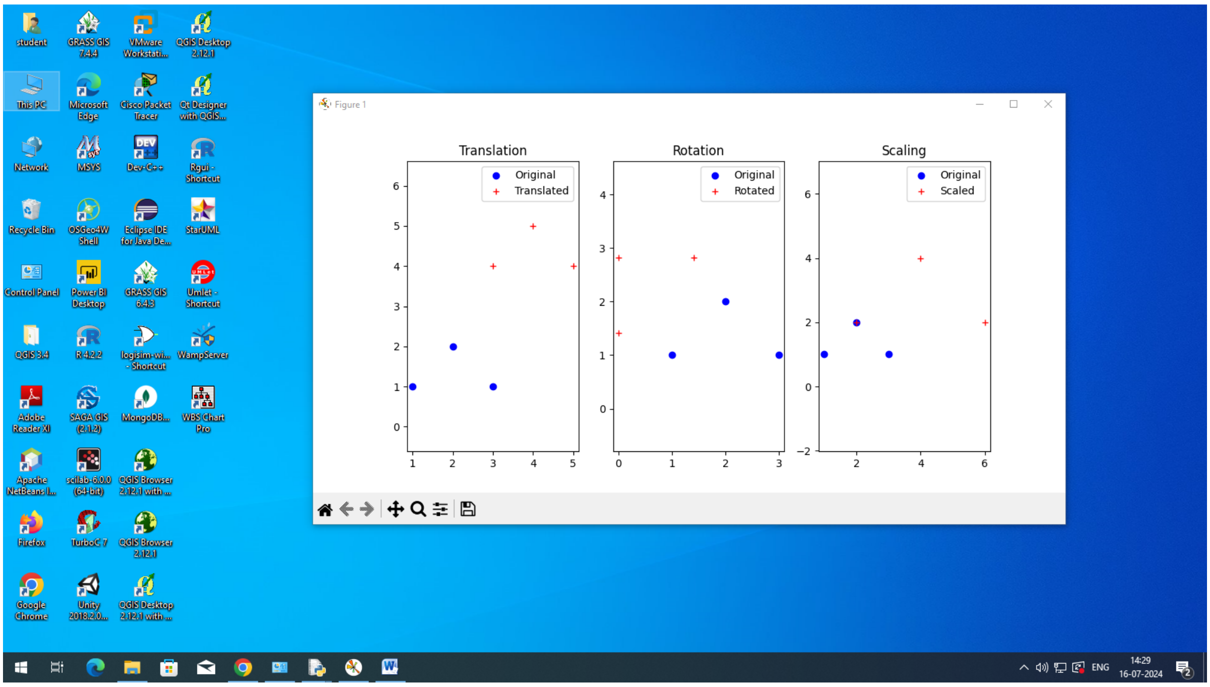
Commands:

```
pip install numpy
pip install matplotlib
```

Code:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.transforms as transforms

# Original points
points = np.array([[1, 1], [2, 2], [3, 1]])

# Translation
translation_matrix = np.array([[1, 0, 2], [0, 1, 3], [0, 0, 1]])  # Translation by (2, 3)
translated_points = np.dot(translation_matrix, np.hstack([points, np.ones((points.shape[0],
1))]).T).T[:, :2]

# Rotation
theta = np.pi / 4  # Rotation angle (45 degrees)
rotation_matrix = np.array([[np.cos(theta), -np.sin(theta), 0], [np.sin(theta), np.cos(theta), 0], [0, 0,
1]])
rotated_points = np.dot(rotation_matrix, np.hstack([points, np.ones((points.shape[0], 1))]).T).T[:, :2]

# Scaling
scaling_matrix = np.array([[2, 0, 0], [0, 2, 0], [0, 0, 1]])  # Scaling by a factor of 2
scaled_points = np.dot(scaling_matrix, np.hstack([points, np.ones((points.shape[0], 1))]).T).T[:, :2]

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

plt.subplot(1, 3, 1)
plt.title('Translation')
plt.plot(points[:, 0], points[:, 1], 'bo', label='Original')
plt.plot(translated_points[:, 0], translated_points[:, 1], 'r+', label='Translated')
plt.axis('equal')
plt.legend()
plt.subplot(1, 3, 2)
plt.title('Rotation')
plt.plot(points[:, 0], points[:, 1], 'bo', label='Original')
plt.plot(rotated_points[:, 0], rotated_points[:, 1], 'r+', label='Rotated')
plt.axis('equal')
plt.legend()
plt.subplot(1, 3, 3)
plt.title('Scaling')
plt.plot(points[:, 0], points[:, 1], 'bo', label='Original')
plt.plot(scaled_points[:, 0], scaled_points[:, 1], 'r+', label='Scaled')
plt.axis('equal')
plt.legend()
plt.show()
```

output:



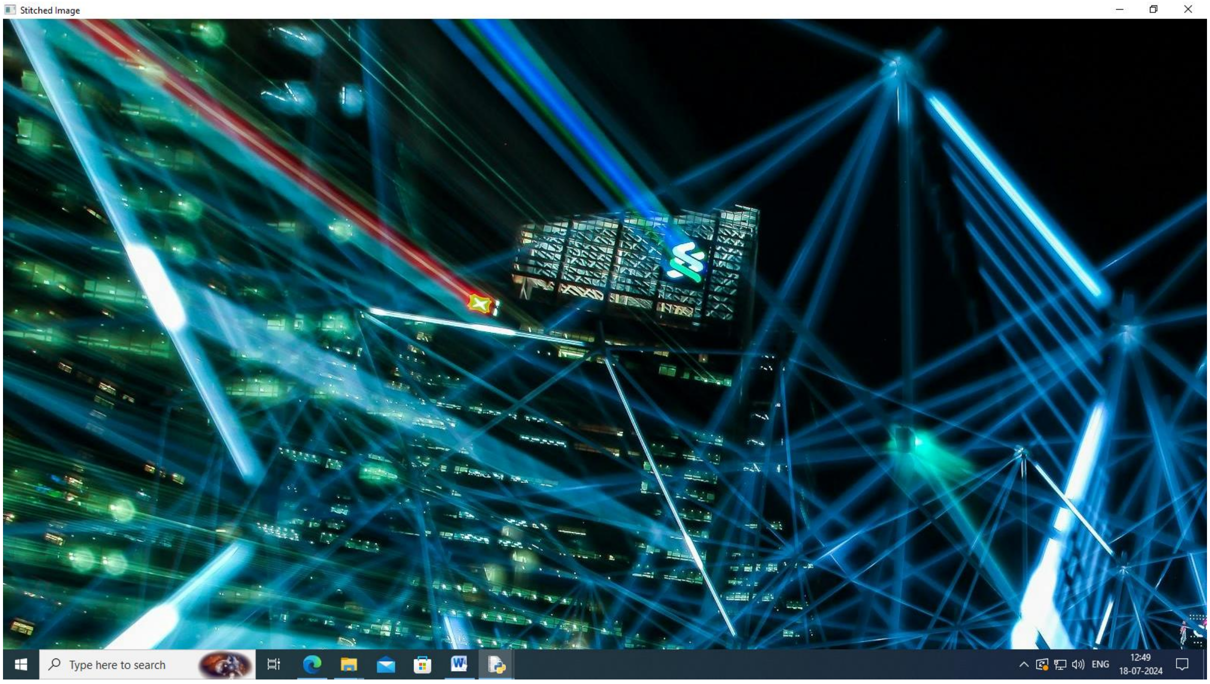
Practical no.: 2

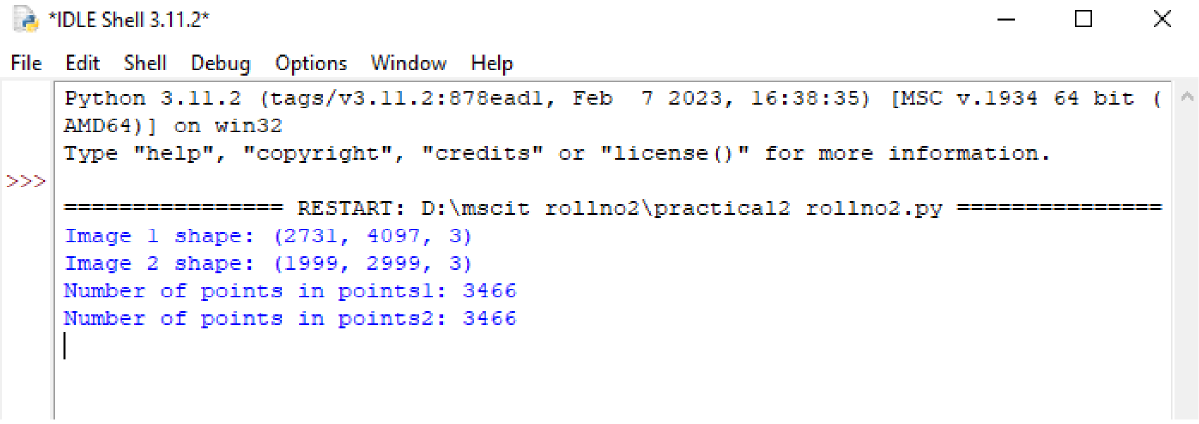
Aim: Perform Image Stitching .

Code:

```
import cv2
import numpy as np

image1 = cv2.imread("pex.jpg")
image2 = cv2.imread("pex1.jpg")
print("Image 1 shape:", image1.shape)
print("Image 2 shape:", image2.shape)

gray1 = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(image2, cv2.COLOR_BGR2GRAY)

sift = cv2.SIFT_create()
keypoints1, descriptors1 = sift.detectAndCompute(gray1, None)
keypoints2, descriptors2 = sift.detectAndCompute(gray2, None)

matcher = cv2.BFMatcher()
matches = matcher.match(descriptors1, descriptors2)
matches = sorted(matches, key=lambda x: x.distance)

points1 = np.float32([keypoints1[match.queryIdx].pt for match in matches]).reshape(-1, 1, 2)
print("Number of points in points1:", len(points1))
points2 = np.float32([keypoints2[match.trainIdx].pt for match in matches]).reshape(-1, 1, 2)
print("Number of points in points2:", len(points2))

homography, _ = cv2.findHomography(points1, points2, cv2.RANSAC)

height, width = gray2.shape
stitched_image = cv2.warpPerspective(image1, homography, (width, height))
stitched_image[0:image2.shape[0], 0:image2.shape[1]] = image2

cv2.imshow('Stitched Image', stitched_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output:





```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: D:\mscit rollno2\practical2 rollno2.py ================
Image 1 shape: (2731, 4097, 3)
Image 2 shape: (1999, 2999, 3)
Number of points in points1: 3466
Number of points in points2: 3466
```

Practical no.: 3

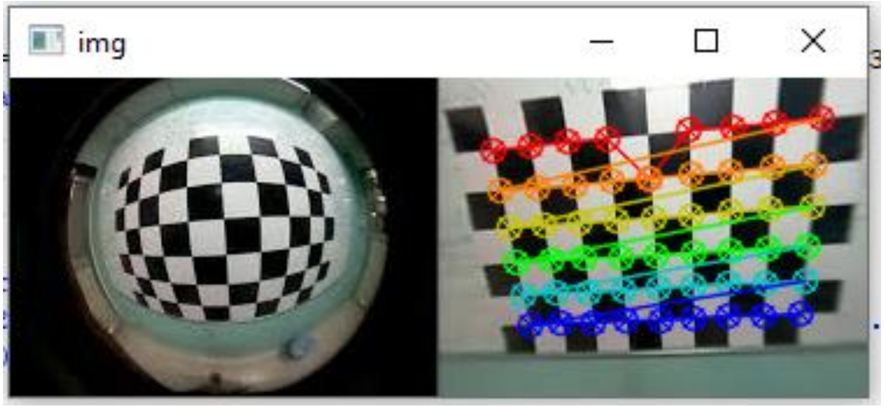
Aim: Perform Camera Calibration.

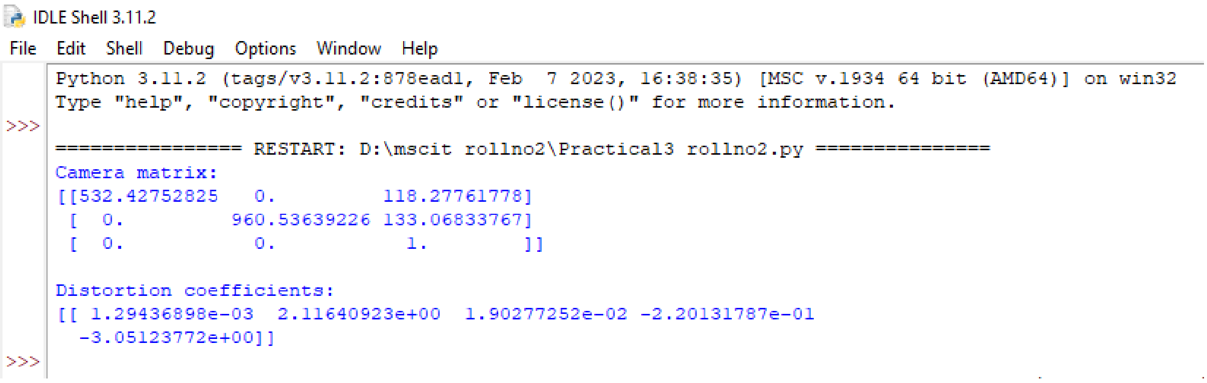
Code:

```
import numpy as np
import cv2
import glob

# Define the number of corners in the chessboard
num_corners_x = 9
num_corners_y = 6

# Prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
objp = np.zeros((num_corners_x * num_corners_y, 3), np.float32)
objp[:, :2] = np.mgrid[0:num_corners_x, 0:num_corners_y].T.reshape(-1, 2)

# Arrays to store object points and image points from all the images
objpoints = []  # 3d point in real world space
imgpoints = []  # 2d points in image plane.

# Load images
images = glob.glob("D:\mscit rollno2\calibration_images\*.jpg")

# Loop through images and find chessboard corners
for fname in images:
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Find the chessboard corners
    ret, corners = cv2.findChessboardCorners(gray, (num_corners_x, num_corners_y), None)

    # If found, add object points, image points (after refining them)
    if ret:
        objpoints.append(objp)

        corners2 = cv2.cornerSubPix(gray, corners, (11, 11), (-1, -1), criteria=(cv2.TERM_CRITERIA_EPS +
cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001))
        imgpoints.append(corners2)

        # Draw and display the corners
        img = cv2.drawChessboardCorners(img, (num_corners_x, num_corners_y), corners2, ret)
        cv2.imshow('img', img)
        cv2.waitKey()

cv2.destroyAllWindows()
```

```
# Perform camera calibration
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1], None, None)

# Save calibration results
np.savez('calibration.npz', mtx=mtx, dist=dist, rvecs=rvecs, tvecs=tvecs)

# Print calibration results
print("Camera matrix:")
print(mtx)
print("\nDistortion coefficients:")
print(dist)
```

Output:





```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: D:\mscit rollno2\Practical3 rollno2.py ===============
Camera matrix:
[[532.42752825   0.         118.27761778]
 [  0.         960.53639226 133.06833767]
 [  0.           0.           1.        ]]

Distortion coefficients:
[[ 1.29436898e-03  2.11640923e+00  1.90277252e-02 -2.20131787e-01
  -3.05123772e+00]]
>>>
```

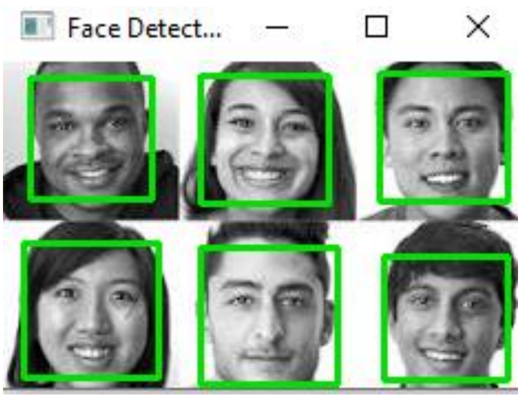
Practical no.: 4

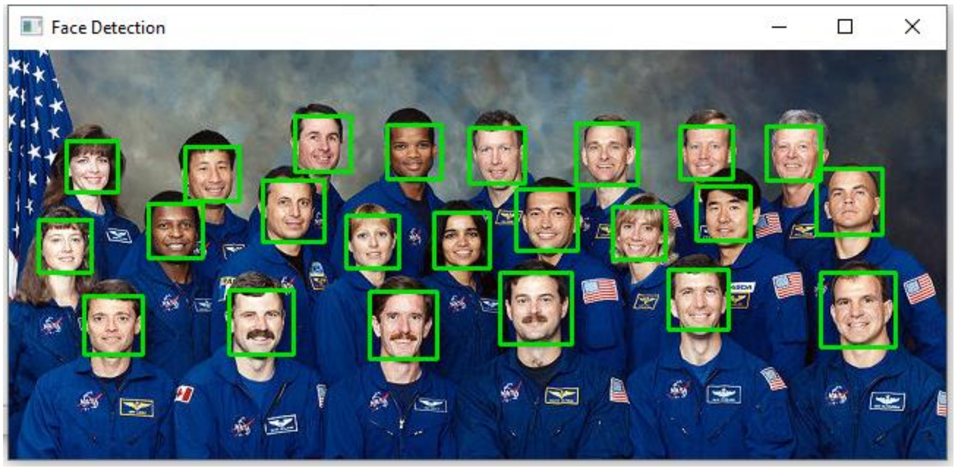
Aim: Perform Face Detection .

Code:

```
import cv2

# Load the pre-trained Haar Cascade face detector
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

# Load the image
image = cv2.imread("IMG1.jpg")

# Convert the image to grayscale (face detection works on grayscale images)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Detect faces in the image
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

# Draw rectangles around the faces
for (x, y, w, h) in faces:
   cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)

# Display the result
cv2.imshow('Face Detection', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output:





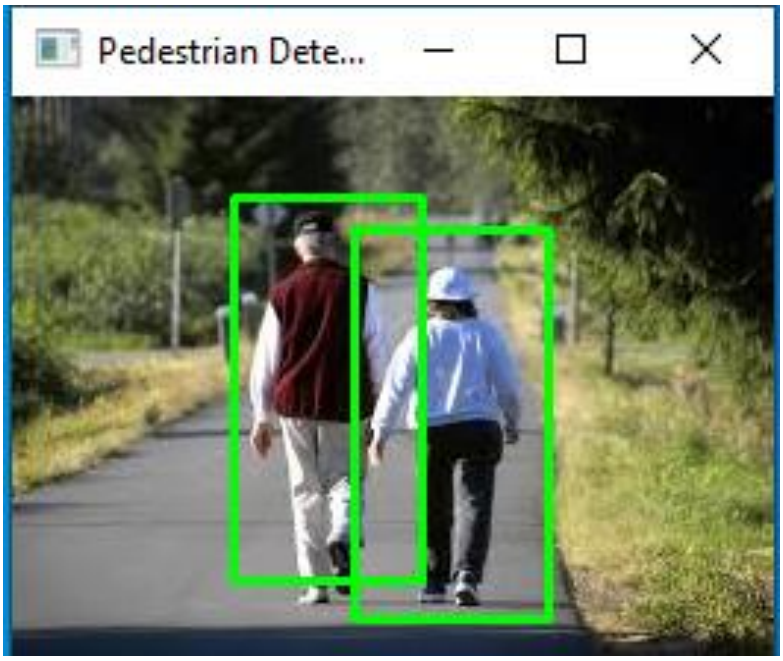
Practical no.: 5

Aim: Perform Pedestrian detection.

Code:

```
import cv2

# Load the pre-trained pedestrian detector
pedestrian_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_fullbody.xml')

# Load the input image ** walking ppl image
image = cv2.imread("D:\MscIT prt1 14\pedestrainimg.jpg")

# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Detect pedestrians in the image
pedestrians = pedestrian_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=1,
minSize=(5, 5))

# Draw rectangles around the detected pedestrians
for (x, y, w, h) in pedestrians:
   cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)

# Display the image with pedestrian detections
cv2.imshow('Pedestrian Detection', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output:

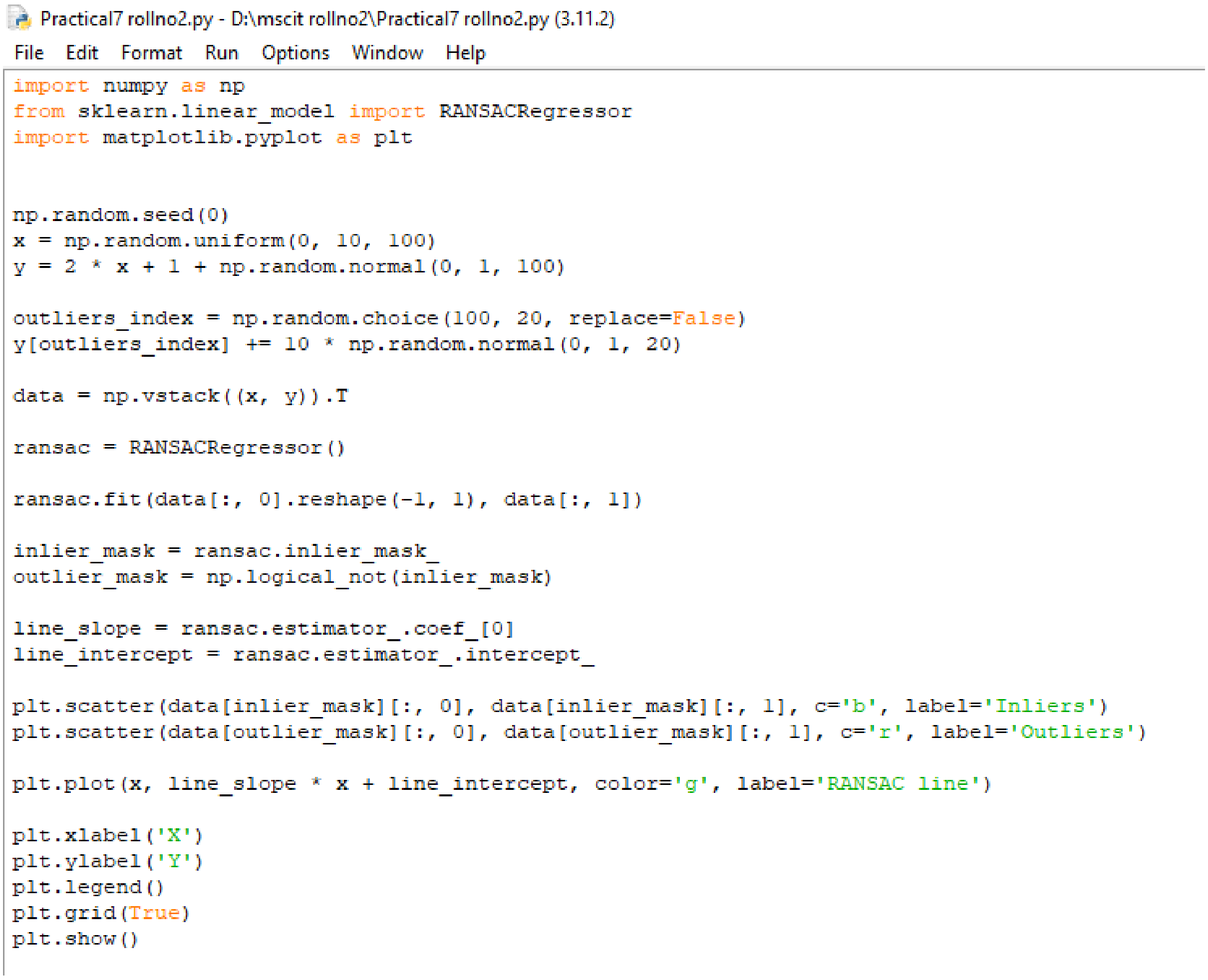


Practical no.: 6

Aim: Perform feature extraction using RANSAC.

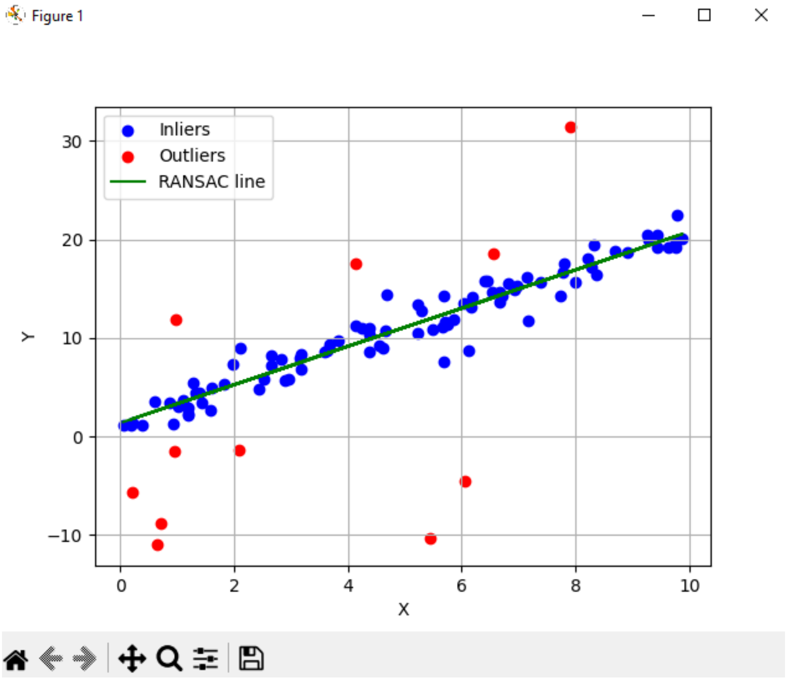
Code:

```
import numpy as np
from sklearn.linear_model import RANSACRegressor
import matplotlib.pyplot as plt

np.random.seed(0)
x = np.random.uniform(0, 10, 100)
y = 2 * x + 1 + np.random.normal(0, 1, 100)

outliers_index = np.random.choice(100, 20, replace=False)
y[outliers_index] += 10 * np.random.normal(0, 1, 20)

data = np.vstack((x, y)).T

ransac = RANSACRegressor()

ransac.fit(data[:, 0].reshape(-1, 1), data[:, 1])

inlier_mask = ransac.inlier_mask_
outlier_mask = np.logical_not(inlier_mask)

line_slope = ransac.estimator_.coef_[0]
line_intercept = ransac.estimator_.intercept_

plt.scatter(data[inlier_mask][:, 0], data[inlier_mask][:, 1], c='b', label='Inliers')
plt.scatter(data[outlier_mask][:, 0], data[outlier_mask][:, 1], c='r', label='Outliers')

plt.plot(x, line_slope * x + line_intercept, color='g', label='RANSAC line')

plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.grid(True)
plt.show()
```


Practical7 rollno2.py - D:\mscit rollno2\Practical7 rollno2.py (3.11.2)
File
Edit
Format
Run
Options
Window
Help

```
import numpy as np
from sklearn.linear_model import RANSACRegressor
import matplotlib.pyplot as plt


np.random.seed(0)
x = np.random.uniform(0, 10, 100)
y = 2 * x + 1 + np.random.normal(0, 1, 100)

outliers_index = np.random.choice(100, 20, replace=False)
y[outliers_index] += 10 * np.random.normal(0, 1, 20)

data = np.vstack((x, y)).T

ransac = RANSACRegressor()

ransac.fit(data[:, 0].reshape(-1, 1), data[:, 1])

inlier_mask = ransac.inlier_mask_
outlier_mask = np.logical_not(inlier_mask)

line_slope = ransac.estimator_.coef_[0]
line_intercept = ransac.estimator_.intercept_

plt.scatter(data[inlier_mask][:, 0], data[inlier_mask][:, 1], c='b', label='Inliers')
plt.scatter(data[outlier_mask][:, 0], data[outlier_mask][:, 1], c='r', label='Outliers')

plt.plot(x, line_slope * x + line_intercept, color='g', label='RANSAC line')

plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.grid(True)
plt.show()
```

Output:

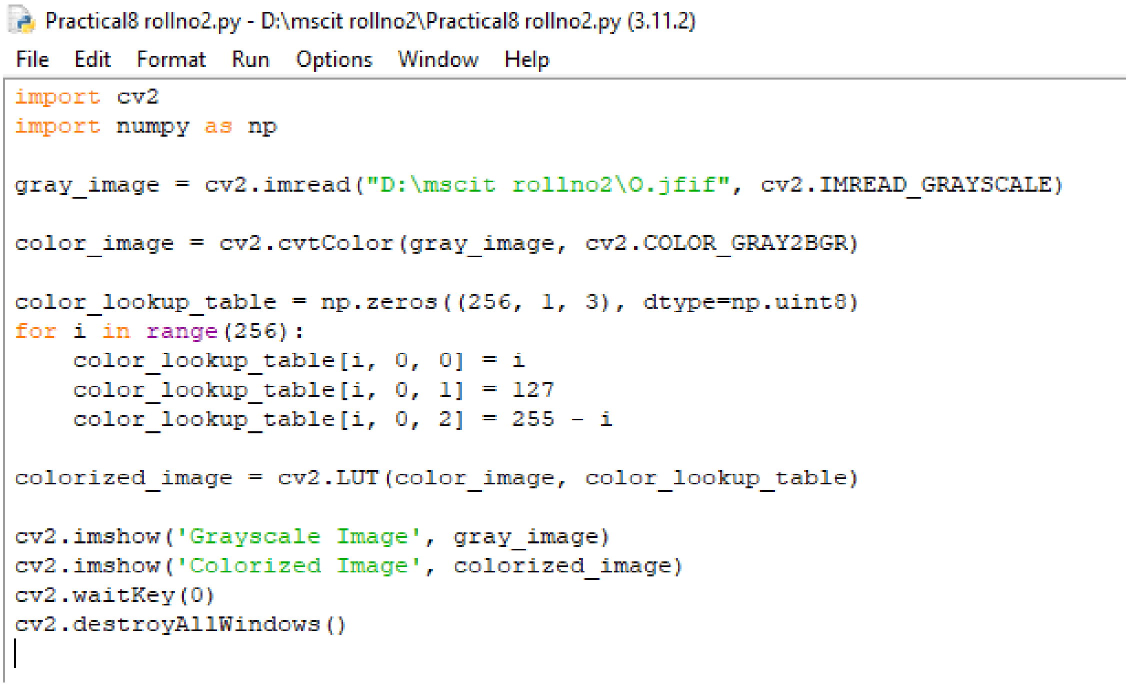


Practical no.: 7

Aim :Perform Colorization.

Code:

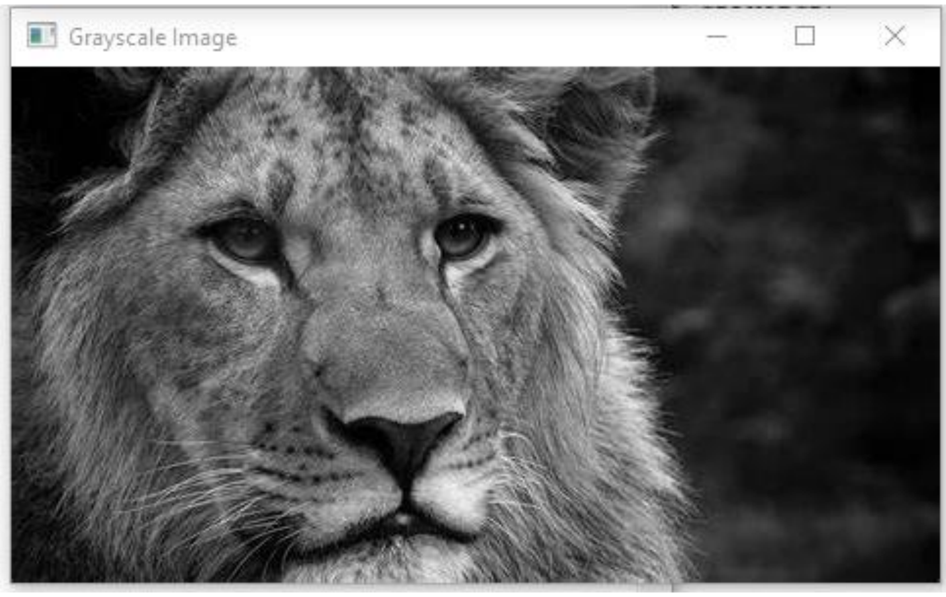
```
import cv2
import numpy as np

gray_image = cv2.imread("D:\mscit rollno2\O.jfif", cv2.IMREAD_GRAYSCALE)

color_image = cv2.cvtColor(gray_image, cv2.COLOR_GRAY2BGR)

color_lookup_table = np.zeros((256, 1, 3), dtype=np.uint8)
for i in range(256):
   color_lookup_table[i, 0, 0] = i
   color_lookup_table[i, 0, 1] = 127
   color_lookup_table[i, 0, 2] = 255 - i

colorized_image = cv2.LUT(color_image, color_lookup_table)

cv2.imshow('Grayscale Image', gray_image)
cv2.imshow('Colorized Image', colorized_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



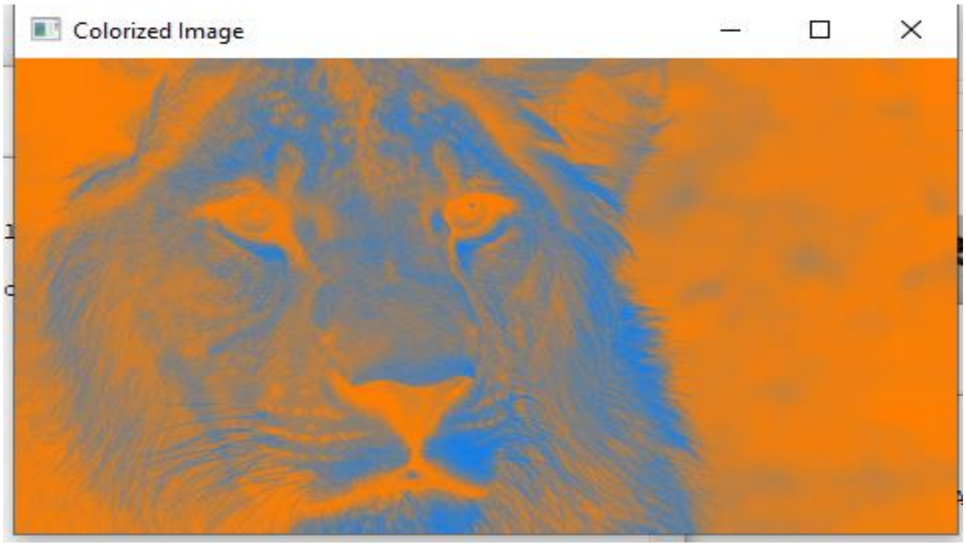
Practical8 rollno2.py - D:\mscit rollno2\Practical8 rollno2.py (3.11.2)
File Edit Format Run Options Window Help

```
import cv2
import numpy as np

gray_image = cv2.imread("D:\mscit rollno2\O.jfif", cv2.IMREAD_GRAYSCALE)

color_image = cv2.cvtColor(gray_image, cv2.COLOR_GRAY2BGR)

color_lookup_table = np.zeros((256, 1, 3), dtype=np.uint8)
for i in range(256):
    color_lookup_table[i, 0, 0] = i
    color_lookup_table[i, 0, 1] = 127
    color_lookup_table[i, 0, 2] = 255 - i

colorized_image = cv2.LUT(color_image, color_lookup_table)

cv2.imshow('Grayscale Image', gray_image)
cv2.imshow('Colorized Image', colorized_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Input image:



Output image:



Practical no.: 8A

Aim: Perform Image Matting and Composting.

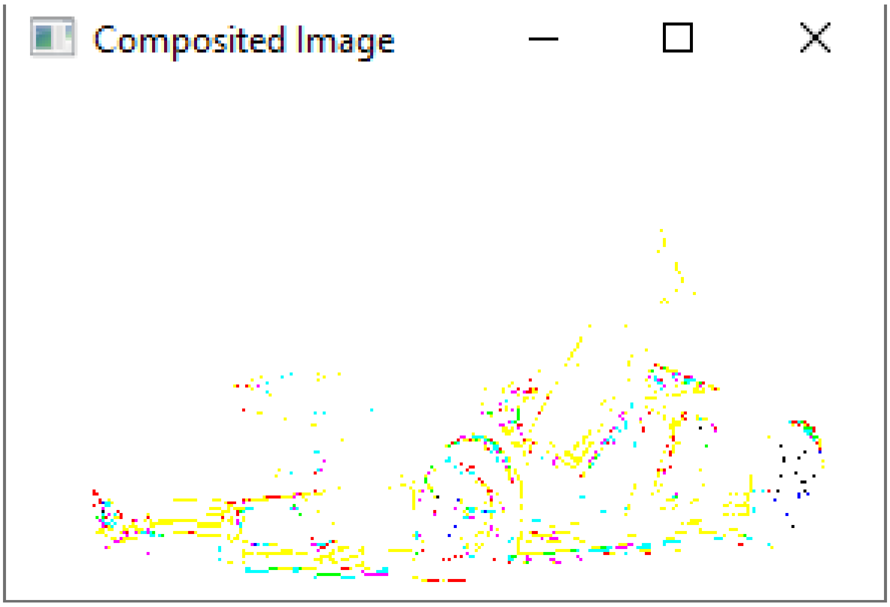
Code:

```
import cv2
import numpy as np
def estimate_alpha(image, trimap):
    # Placeholder function, replace with your matting algorithm implementation
    # This example simply sets alpha values based on trimap (e.g., foreground = 1, background = 0,
unknown = interpolated)
    alpha = np.zeros_like(trimap, dtype=np.float32)
    alpha[trimap == 255] = 1.0  # Foreground
    alpha[trimap == 0] = 0.0  # Background
    alpha[(trimap > 0) & (trimap < 255)] = 0.5  # Interpolated
    return alpha

def image_matting(image, trimap):
    # Convert image and trimap to float32
    image = image.astype(np.float32) / 255.0
    trimap = trimap.astype(np.float32) / 255.0

    # Estimate alpha matte using a matting algorithm
    # Replace this with your desired matting algorithm
    alpha = estimate_alpha(image, trimap)

    # Clip alpha values to [0, 1]
    alpha = np.clip(alpha, 0, 1)

    return alpha

def composit_foreground_background(foreground, background, alpha):
    # Resize background to match the foreground size
    background = cv2.resize(background, (foreground.shape[1], foreground.shape[0]))

    # Convert alpha to 3 channels
    alpha = np.stack((alpha, alpha, alpha), axis=2)

    # Composite foreground and background using alpha matte
    composited_image = alpha * foreground + (1 - alpha) * background

    return composited_image

# Example usage
if __name__ == "__main__":
    # Read foreground, background, and trimap images
    foreground = cv2.imread("model.jpg")
    background = cv2.imread("model.jpg")
    trimap = cv2.imread("model.jpg", cv2.IMREAD_GRAYSCALE)

    # Perform image matting
```

```
alpha = image_matting(foreground, trimap)

# Perform compositing
composited_image = composit_foreground_background(foreground, background, alpha)

# Display result
cv2.imshow("Composited Image", composited_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Input image:

Output image:

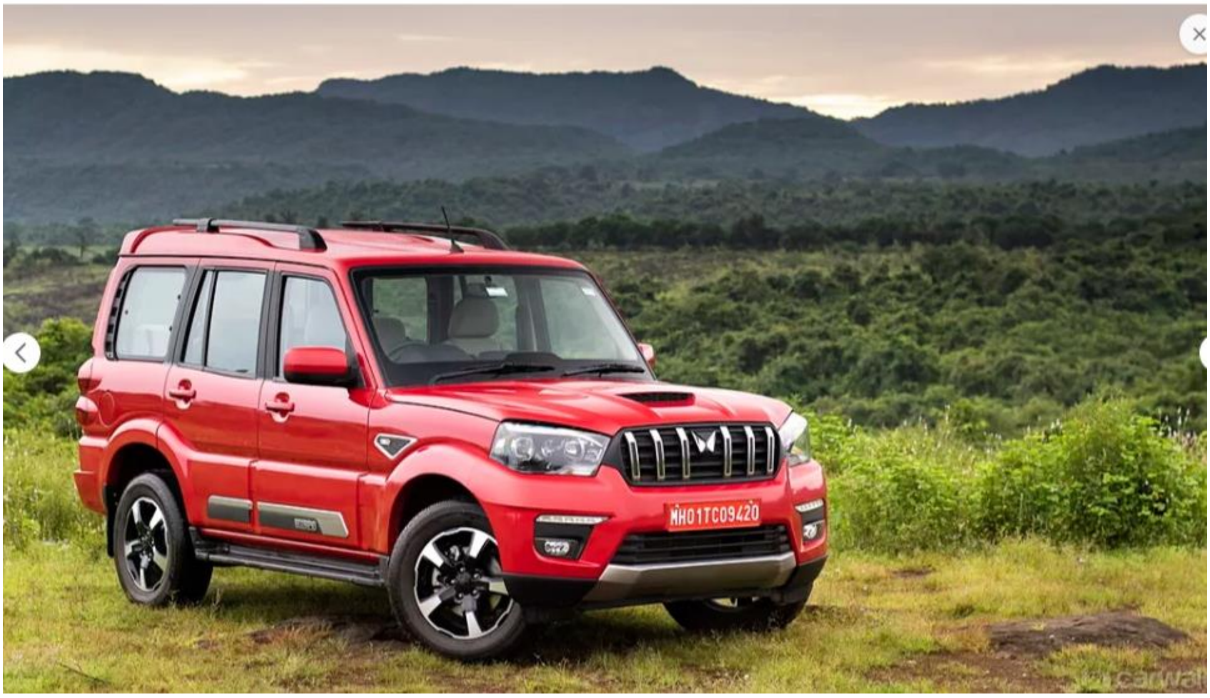


Practical no.: 8B

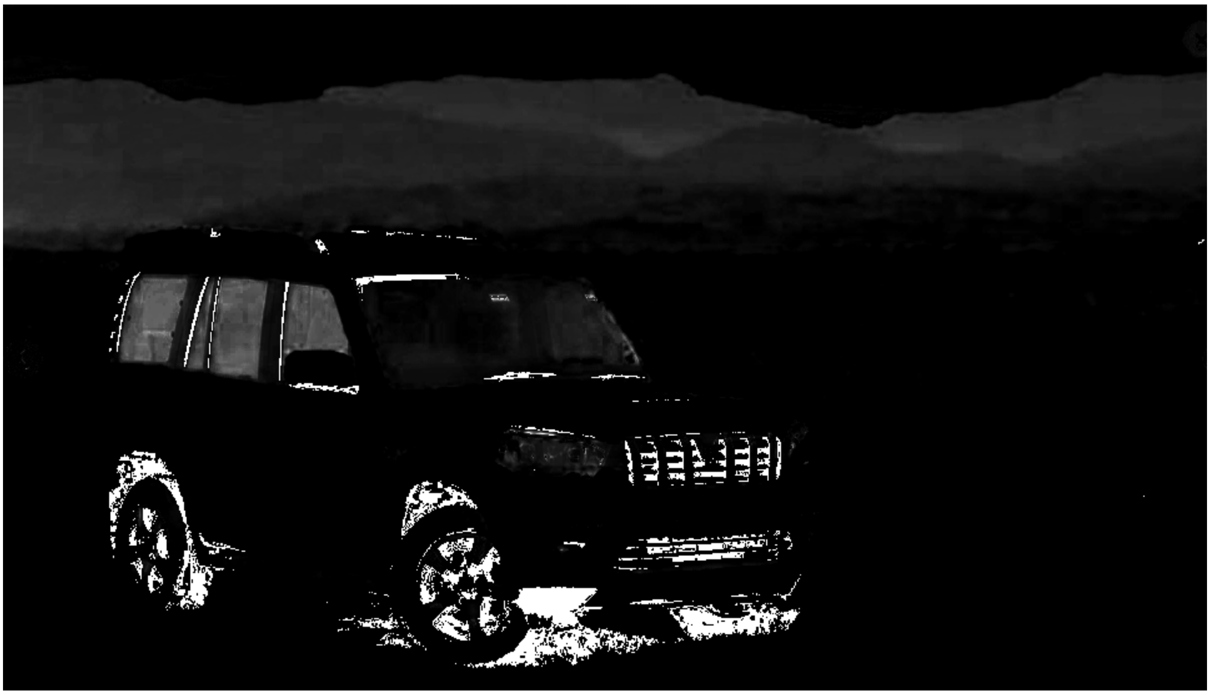
Code:

```
import cv2
import numpy as np

def estimate_alpha(image, trimap):
    # Convert to float
    image = image.astype(np.float32) / 255.0

    # Normalize trimap to [0, 1]
    trimap = trimap.astype(np.float32) / 255.0

    # Compute alpha matte using Closed-Form matting
    foreground = np.where(trimap > 0.95, 1.0, 0.0)  # Foreground mask
    alpha = np.where(trimap > 0.05, 1.0, 0.0)  # Alpha initialization
    for _ in range(5):  # Iterative refinement
        alpha = (image[:, :, 0] - image[:, :, 2] * alpha) / (1e-12 + foreground + (1.0 - trimap) * alpha)
        alpha = np.clip(alpha, 0, 1)

    return alpha

# Example usage
if __name__ == "__main__":
    # Read image and trimap
    image = cv2.imread("model.jpg")
    trimap = cv2.imread("model.jpg", cv2.IMREAD_GRAYSCALE)

    # Estimate alpha matte
    alpha = estimate_alpha(image, trimap)

    # Save or display alpha matte
    cv2.imshow("Alpha Matte", alpha)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Input image:



Output image:



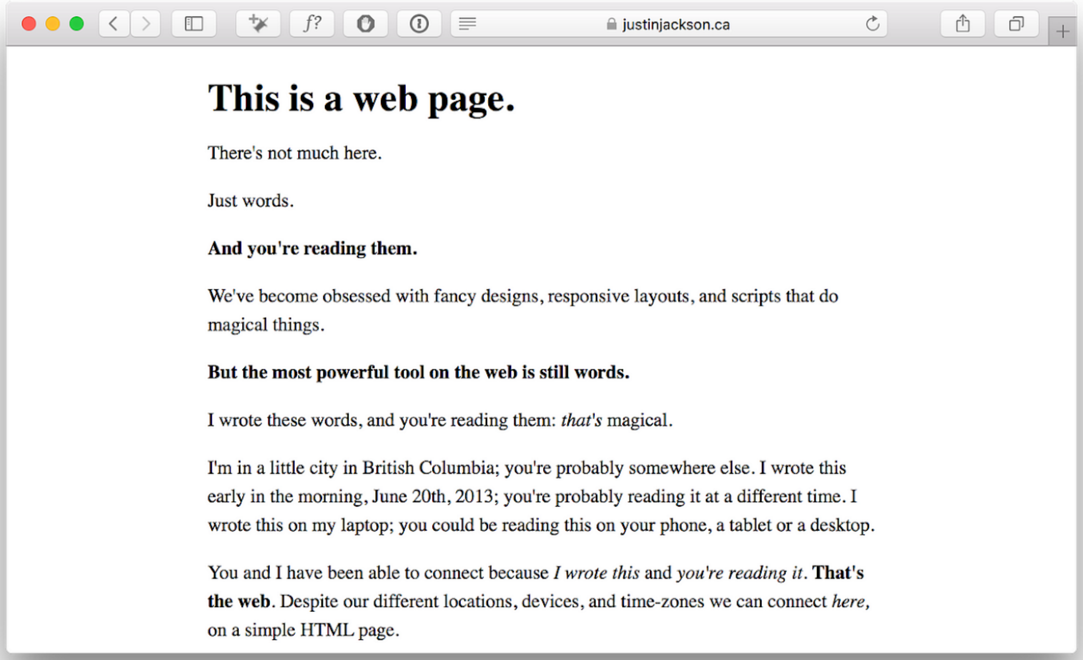
Practical no.: 9

Aim: Perform Text Detection and Recognition.

Code:

```
import pytesseract
import cv2

# Load the image
image_path = 'webpage.png'
image = cv2.imread(image_path)

# Perform OCR
text = pytesseract.image_to_string(image)

# Print the extracted text
print(text)
```

Input image:



This is a web page.

There's not much here.

Just words.

And you're reading them.

We've become obsessed with fancy designs, responsive layouts, and scripts that do magical things.

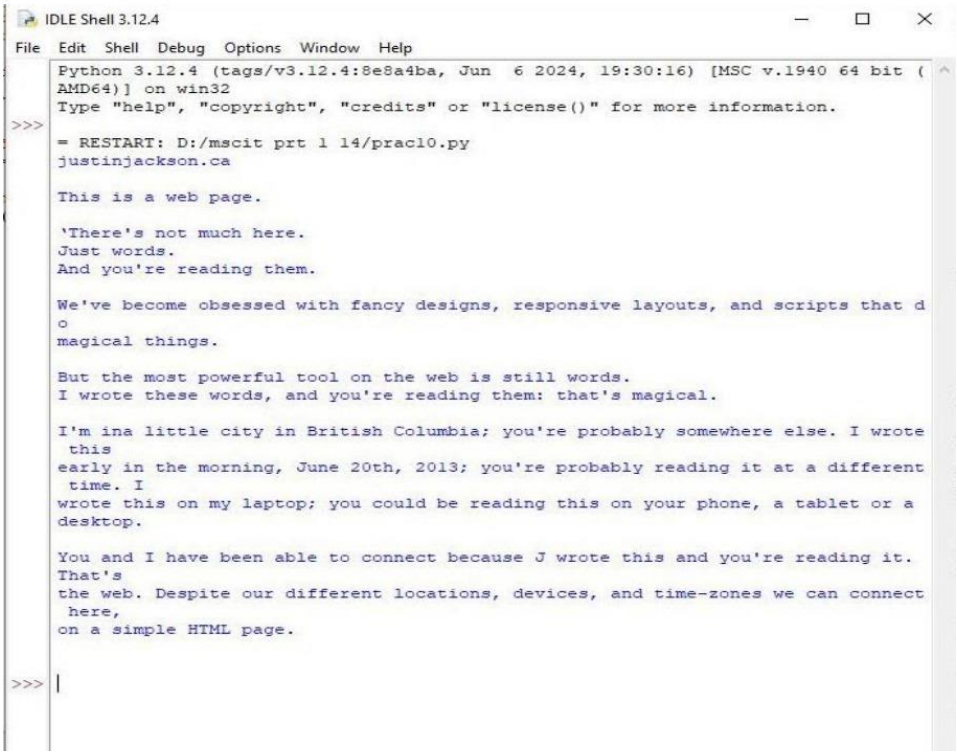
But the most powerful tool on the web is still words.

I wrote these words, and you're reading them: *that's* magical.

I'm in a little city in British Columbia; you're probably somewhere else. I wrote this early in the morning, June 20th, 2013; you're probably reading it at a different time. I wrote this on my laptop; you could be reading this on your phone, a tablet or a desktop.

You and I have been able to connect because *I wrote this* and *you're reading it*. **That's the web**. Despite our different locations, devices, and time-zones we can connect *here*, on a simple HTML page.

Output:



```
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: D:/mscit prt 1 14/prac10.py
justinjackson.ca

This is a web page.

'There's not much here.
Just words.
And you're reading them.

We've become obsessed with fancy designs, responsive layouts, and scripts that d
o
magical things.

But the most powerful tool on the web is still words.
I wrote these words, and you're reading them: that's magical.

I'm ina little city in British Columbia; you're probably somewhere else. I wrote
 this
early in the morning, June 20th, 2013; you're probably reading it at a different
 time. I
wrote this on my laptop; you could be reading this on your phone, a tablet or a
desktop.

You and I have been able to connect because J wrote this and you're reading it.
That's
the web. Despite our different locations, devices, and time-zones we can connect
 here,
on a simple HTML page.


>>> |
```