COMPUTER VISION

MASTERS OF SCIENCE (INFORMATION TECHNOLOGY)

By

RIDDHI MITESH VARTAK

Seat No:

Under the esteemed guidance of

MRS. PROF. PALLAVI AHIRE

DEPARTMENT OF INFORMATION TECHNOLOGY

VIDYAVARDHINI’S

ANNASAHEB VARTAK COLLEGE OF ARTS, K.M.

COLLEGE OF COMMERCE,E.S.A. COLLEGE OF SCIENCE

(Affiliated to University of Mumbai)

VASAI (WEST)-401202,

DIST.PALGHAR MAHARASHTRA 2024-2025

DEPARTMENT OF INFORMATION TECHNOLOGY

VIDYAVARDHINI'S

ANNASAHEB VARTAK COLLEGE OF ARTS,K.M. COLLEGE OF COMMERCE,E.S.A. COLLEGE OF SCIENCE

(Affiliated to University of Mumbai)

VASAI (WEST)-401202,

DIST.PALGHAR MAHARASHTRA 2024-2025

CERTIFICATE

This is to certify that **Miss. Riddhi Mitesh Vartak** Exam Seat No. has satisfactorily completed practical's of the subject "Computer Vision" as part of the practical fulfilment of MSc.IT Sem II as prescribed by the university of Mumbai for the year 2024-2025.

Internal Guide (HOD)

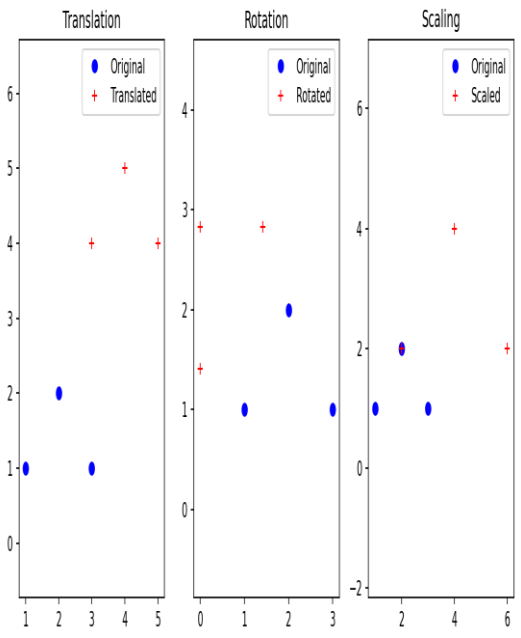
External Examiner

Date: College Seal

INDEX

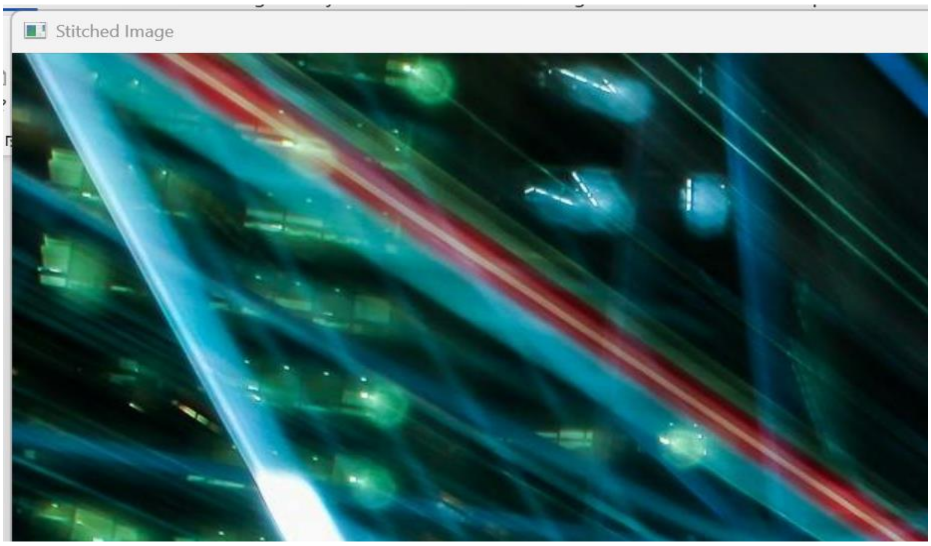
Practical No: 1: Perform Geometric transformation using Python

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.transforms as transforms
points = np.array([[1, 1], [2, 2], [3, 1]])
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]
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_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]
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()
```



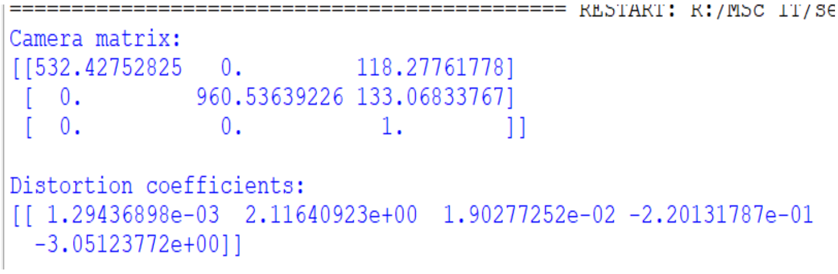
Practical 2: Perform Image Stitching

```
import cv2
import numpy as np
image1 = cv2.imread("R:\MSc IT\sem 2\computer vision\pex.jpg")
image2 = cv2.imread("R:\MSc IT\sem 2\computer vision\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()
```

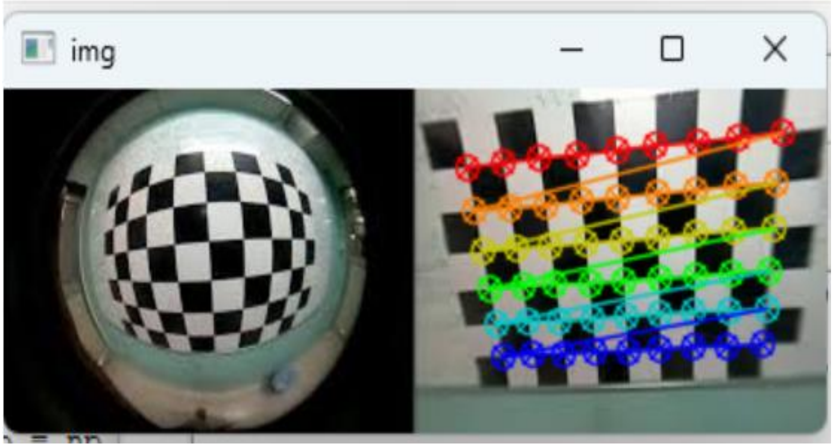


Practical No 3: Camera Calibration

```
objpoints = []
imgpoints = []
images = glob.glob("R:\MSc IT\sem 2\computer vision\calibration_images\*.jpg")
for fname in images:
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ret, corners = cv2.findChessboardCorners(gray, (num_corners_x, num_corners_y), None)
    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)
        img = cv2.drawChessboardCorners(img, (num_corners_x, num_corners_y), corners2, ret)
        cv2.imshow('img', img)
        cv2.waitKey(500)
cv2.destroyAllWindows()
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1], None, None)
np.savez('calibration.npz', mtx=mtx, dist=dist, rvecs=rvecs, tvecs=tvecs)
print("Camera matrix:")
print(mtx)
print("\nDistortion coefficients:")
print(dist)
```

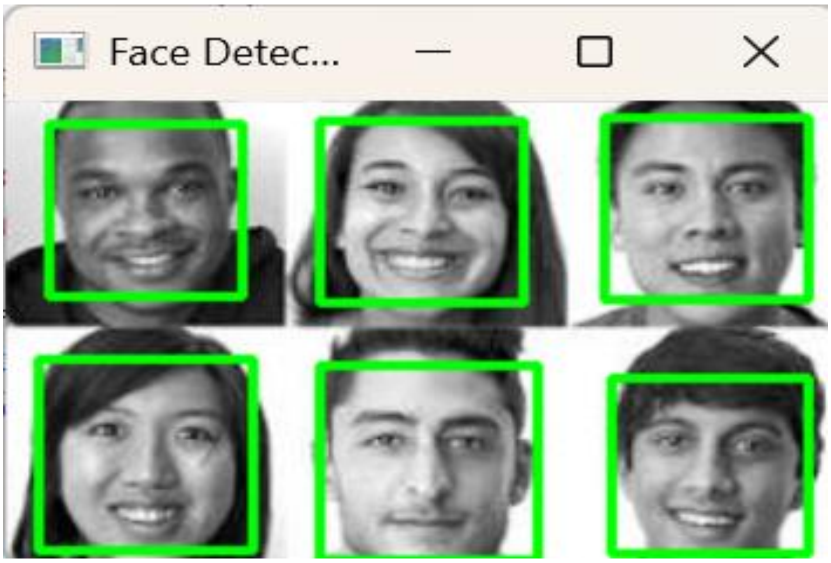


```
============================================ RESTART: R:/MSc IT/se
Camera matrix:
[[532.42752825   0.         118.27761778]
 [  0.         960.53639226 133.06833767]
 [  0.           0.           1.        ]]

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



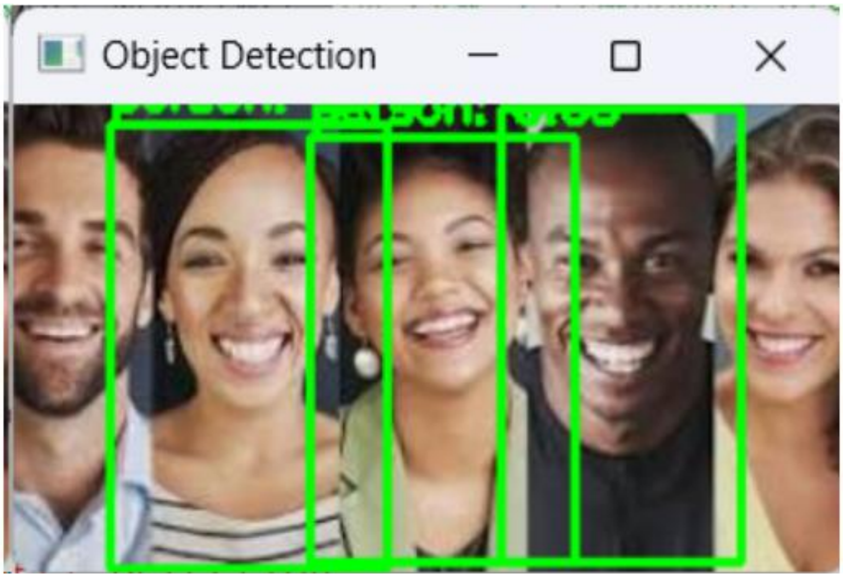
Practical 4: Face Detection

```
import cv2

face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

image = cv2.imread("R:\MSc IT\sem 2\computer vision\IMG1.jpg")

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

for (x, y, w, h) in faces:

    cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)

cv2.imshow('Face Detection', image)

cv2.waitKey(0)

cv2.destroyAllWindows()
```



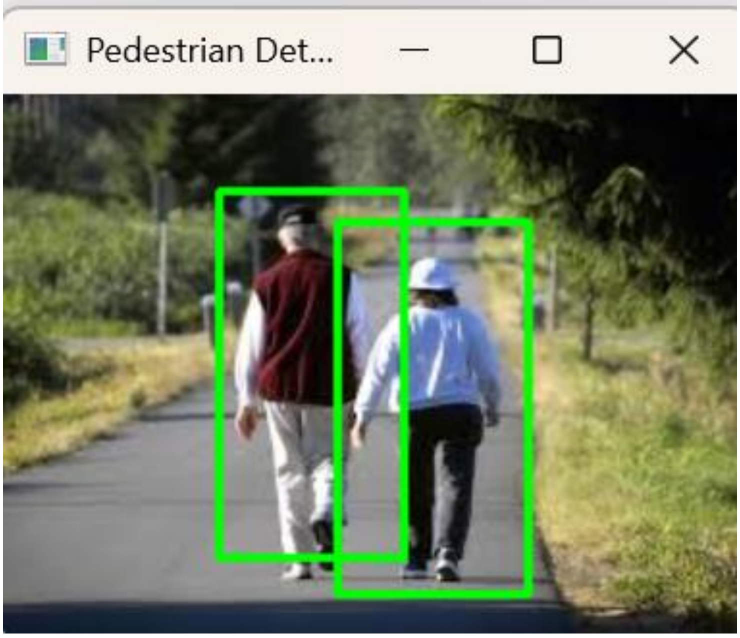
Practical 5: Object Detection

```
import numpy as np
import os
import tensorflow as tf
import cv2
MODEL_NAME = 'ssd_mobilenet_v2_coco_2018_03_29'
PATH_TO_CKPT = os.path.join(MODEL_NAME, 'frozen_inference_graph.pb')
NUM_CLASSES = 90
detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.compat.v1.GraphDef()
    with tf.io.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')
#PATH_TO_LABELS = os.path.join('data', 'mscoco_label_map.pbtxt')
PATH_TO_LABELS = "C:\PALLAVI WORK\MSC IT SEM 2 COMPUTER
VISION\mscoco_label_map.pbtxt"
category_index = {}
with open(PATH_TO_LABELS, 'r') as f:
    lines = f.readlines()
    for line in lines:
        if 'id:' in line:
            id_index = int(line.strip().split(':')[1])
        if 'display_name:' in line:
            name = line.strip().split(':')[1].strip().strip('"')
            category_index[id_index] = {'name': name}
def detect_objects(image):
    with detection_graph.as_default():
        with tf.compat.v1.Session(graph=detection_graph) as sess:
            # Expand dimensions since the model expects images to have shape: [1, None, None, 3]
            image_expanded = np.expand_dims(image, axis=0)
            image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
            boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
            scores = detection_graph.get_tensor_by_name('detection_scores:0')
            classes = detection_graph.get_tensor_by_name('detection_classes:0')
            num_detections = detection_graph.get_tensor_by_name('num_detections:0')
            (boxes, scores, classes, num_detections) = sess.run([boxes, scores, classes,
num_detections],feed_dict={image_tensor: image_expanded})
            for i in range(len(scores[0])):
                if scores[0][i] > 0.5:  # Adjust confidence threshold as needed
                    class_id = int(classes[0][i])
                    class_name = category_index[class_id]['name']
                    score = float(scores[0][i])
                    ymin, xmin, ymax, xmax = boxes[0][i]
                    (left, right, top, bottom) = (xmin * image.shape[1], xmax * image.shape[1], ymin *
image.shape[0], ymax * image.shape[0])
                    cv2.rectangle(image, (int(left), int(top)), (int(right), int(bottom)), (0, 255, 0), 2)
                    cv2.putText(image, '{}: {:.2f}'.format(class_name, score), (int(left), int(top - 5)),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
    return image
input_image = cv2.imread("C:\PALLAVI WORK\MSC IT SEM 2 COMPUTER VISION\IMG.jpg")
output_image = detect_objects(input_image)
cv2.imshow('Object Detection', output_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



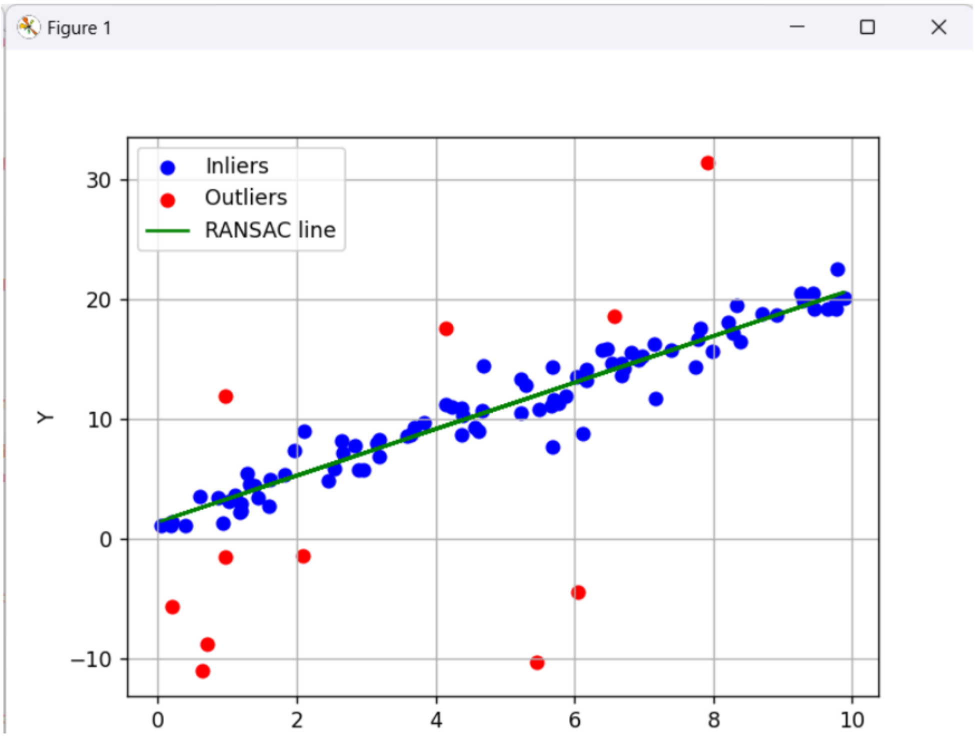
Practical 6: Pedestrian Detection

```
import cv2

pedestrian_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_fullbody.xml')

image = cv2.imread("R:\MSc IT\sem 2\computer vision\pedestrainimg.jpg")

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

pedestrians = pedestrian_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=1, minSize=(5, 5))

for (x, y, w, h) in pedestrians:

    cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)

cv2.imshow('Pedestrian Detection', image)

cv2.waitKey(0)

cv2.destroyAllWindows()
```



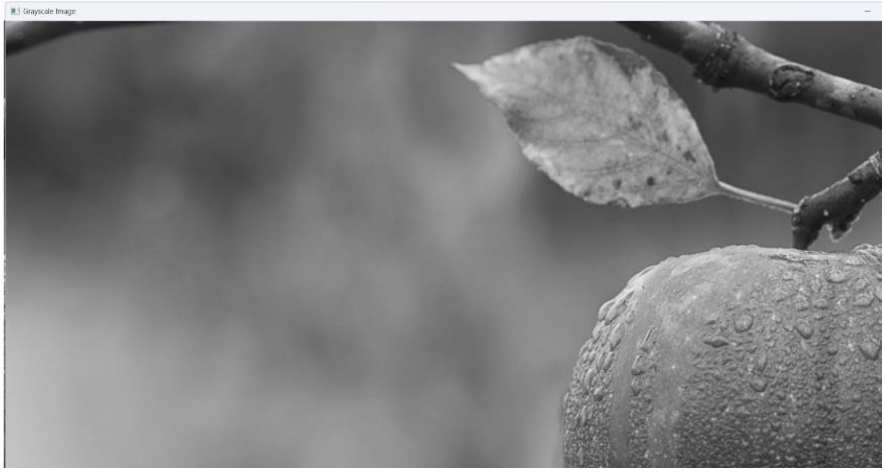
Practical 7: Feature Extraction using RANSAC

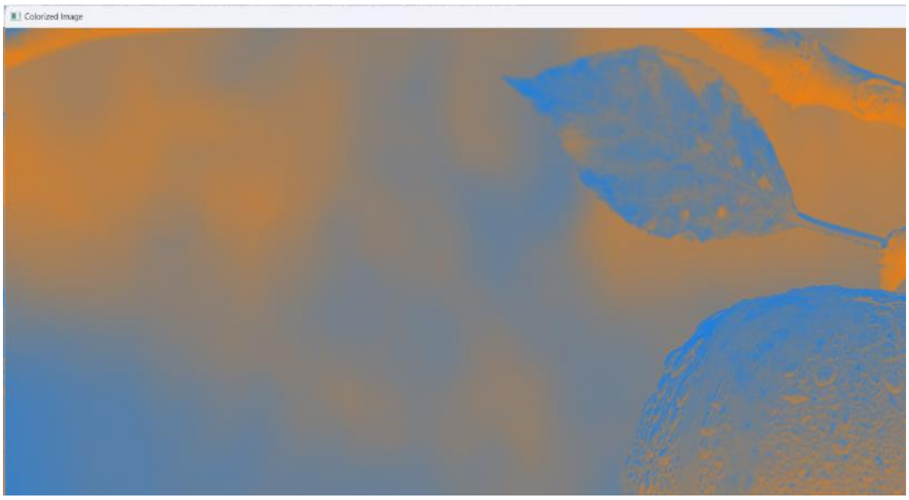
```
import numpy as np
from sklearn.linear_model import RANSACRegressor
import matplotlib.pyplot as plt
np.random.seed(0)
x = np.random.uniform(0, 10, 100)
y = 2 * x + 1 + np.random.normal(0, 1, 100)
outliers_index = np.random.choice(100, 20, replace=False)
y[outliers_index] += 10 * np.random.normal(0, 1, 20)
data = np.vstack((x, y)).T
ransac = RANSACRegressor()
ransac.fit(data[:, 0].reshape(-1, 1), data[:, 1])
inlier_mask = ransac.inlier_mask_
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()
```



Practical 8: Colorization of an image

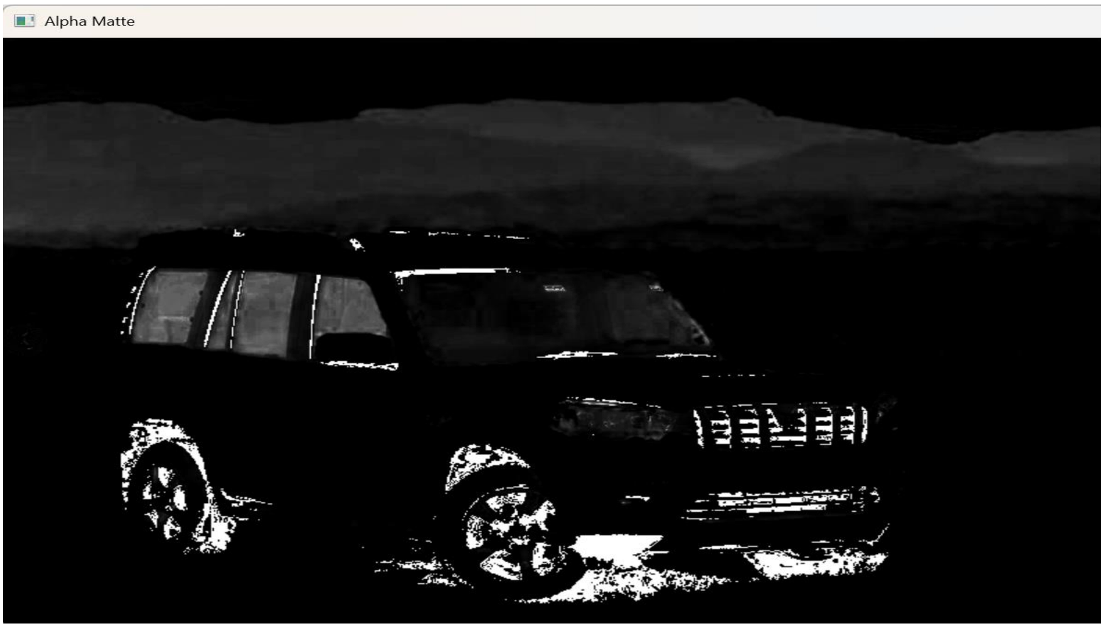
```
import cv2

import numpy as np

gray_image = cv2.imread("R:\MSc IT\sem 2\computer vision\grayimg.jpeg", 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()
```





Practical 9: Image Matting

```
import cv2
import numpy as np
def estimate_alpha(image, trimap):
    image = image.astype(np.float32) / 255.0
    trimap = trimap.astype(np.float32) / 255.0
    foreground = np.where(trimap > 0.95, 1.0, 0.0)
    alpha = np.where(trimap > 0.05, 1.0, 0.0)
    for _ in range(5):
        alpha = (image[:, :, 0] - image[:, :, 2] * alpha) / (1e-12 + foreground + (1.0 - trimap) * alpha)
        alpha = np.clip(alpha, 0, 1)
     return alpha
if __name__ == "__main__":
    image = cv2.imread("R:\MSc IT\sem 2\computer vision\model.jpg")
    trimap = cv2.imread("R:\MSc IT\sem 2\computer vision\model.jpg", cv2.IMREAD_GRAYSCALE)
    alpha = estimate_alpha(image, trimap)
    cv2.imshow("Alpha Matte", alpha)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```


Alpha Matte