COMPUTER VISION PRACTICALS

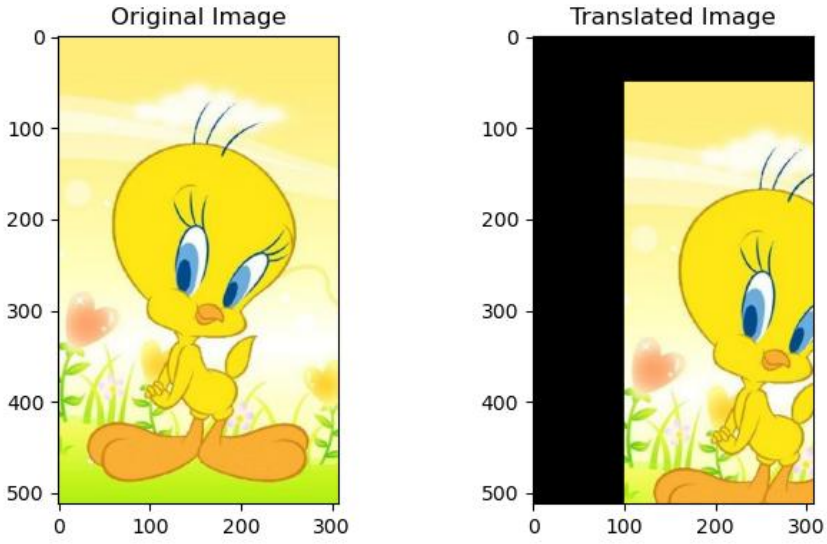
Practical – 1 (a)

Aim: Image Scaling

CODE:

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
img = cv2.imread("C:/Users/Student/Downloads/tweety.jpg")
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
rows, cols, channels = img_rgb.shape
M = np.float32([[1, 0, 100], [0, 1, 50]])
dst = cv2.warpAffine(img_rgb, M, (cols, rows))
fig, axs = plt.subplots(1, 2, figsize=(7, 4))
axs[0].imshow(img_rgb)
axs[0].set_title('Original Image')
axs[1].imshow(dst)
axs[1].set_title('Translated Image')
plt.tight_layout()
plt.show()
```

OUTPUT:



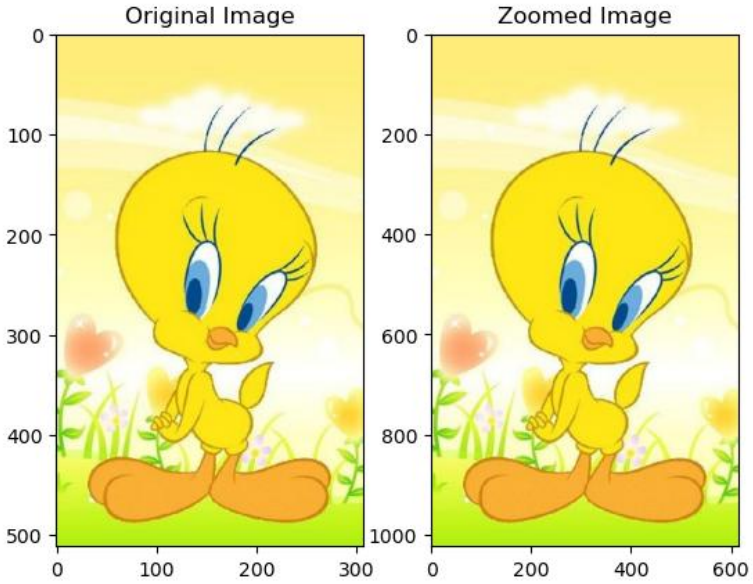
Practical – 1 (b)

Aim: Image Shrinking.

CODE:

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
img = cv2.imread("C:/Users/Student/Downloads/tweety.jpg")
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
rows, cols, channels = img_rgb.shape
resize_img = cv2.resize(img_rgb, (0, 0), fx=2, fy=2, interpolation=cv2.INTER_CUBIC)
plt.subplot(121), plt.imshow(img_rgb), plt.title('Original Image')
plt.subplot(122), plt.imshow(resize_img), plt.title('Zoomed Image')
plt.show()
```

OUTPUT:



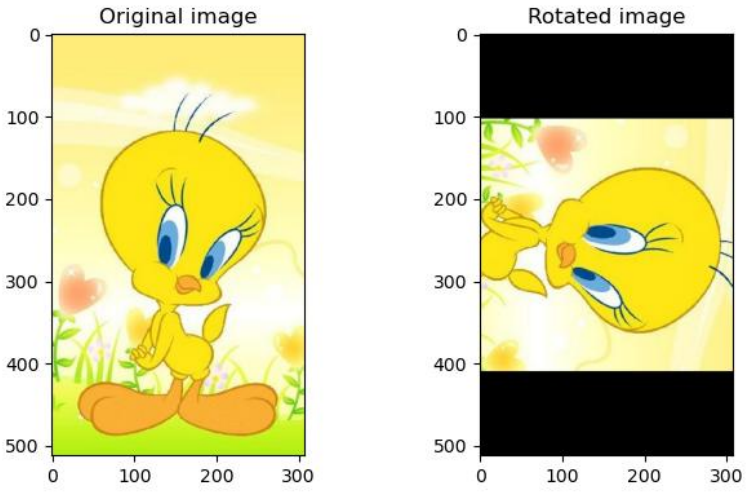
Practical – 1 (c)

Aim: Image Rotation.

CODE:

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
img = cv2.imread("C:/Users/Student/Downloads/tweety.jpg")
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
rows, cols, channels = img_rgb.shape
center = (cols // 2, rows // 2)
angle = -90
scale = 1
rotation_matrix = cv2.getRotationMatrix2D(center, angle, scale)
rotated_image = cv2.warpAffine(img_rgb, rotation_matrix, (cols, rows))
fig, axs = plt.subplots(1, 2, figsize=(7, 4))
axs[0].imshow(img_rgb)
axs[0].set_title("Original image")
axs[1].imshow(rotated_image)
axs[1].set_title("Rotated image")
plt.tight_layout()
plt.show()
```

OUTPUT:



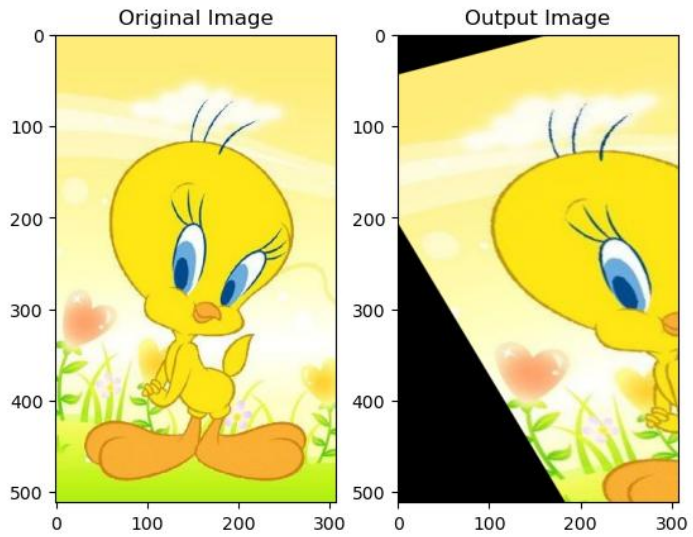
Practical – 1 (d)

Aim: Affine Transformation.

CODE:

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
img = cv2.imread("C:/Users/Student/Downloads/tweety.jpg")
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
rows, cols, channels = img_rgb.shape
pts1 = np.float32([[50, 50], [200, 50], [50, 200]])
pts2 = np.float32([[10, 100], [200, 50], [100, 250]])
M = cv2.getAffineTransform(pts1, pts2)
dst = cv2.warpAffine(img_rgb, M, (cols, rows))
plt.subplot(121), plt.imshow(img_rgb), plt.title('Original Image')
plt.subplot(122), plt.imshow(dst), plt.title('Output Image')
plt.show()
```

OUTPUT:



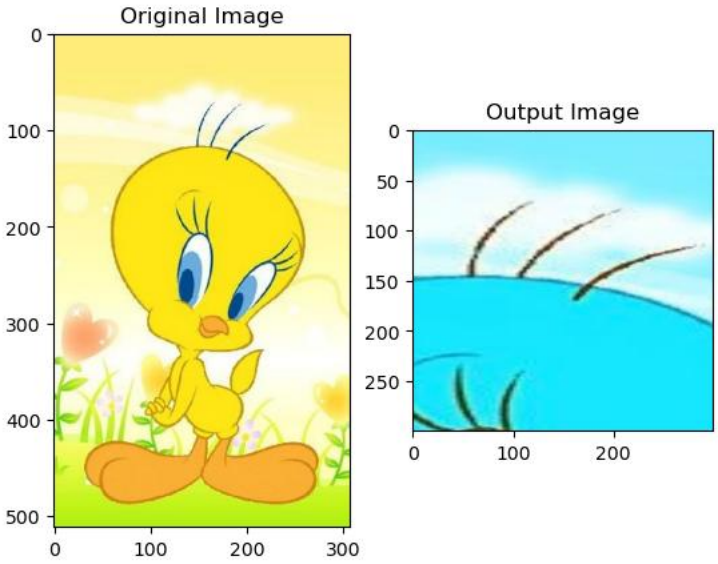
Practical – 1 (e)

Aim: Perspective Transformation.

CODE:

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
img = cv2.imread("C:/Users/Student/Downloads/tweety.jpg")
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
rows, cols, channels = img_rgb.shape
pts1 = np.float32([[133, 34], [226, 16], [133, 206], [226, 219]])
pts2 = np.float32([[0, 0], [300, 0], [0, 300], [300, 300]])
M = cv2.getPerspectiveTransform(pts1, pts2)
dst = cv2.warpPerspective(img, M, (300, 300))
plt.subplot(121), plt.imshow(img_rgb), plt.title('Original Image')
plt.subplot(122), plt.imshow(dst), plt.title('Output Image')
plt.show()
```

OUTPUT:



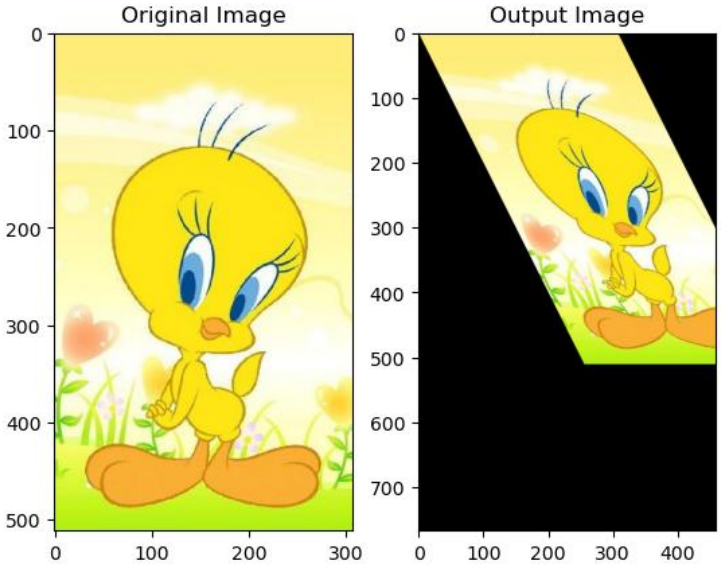
Practical – 1 (f)

Aim: Shearing X-axis.

CODE:

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
img = cv2.imread("C:/Users/Student/Downloads/tweety.jpg")
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
rows, cols, channels = img_rgb.shape
M = np.float32([[1, 0.5, 0], [0, 1, 0], [0, 0, 1]])
dst = cv2.warpPerspective(img_rgb, M, (int(cols * 1.5), int(rows * 1.5)))
plt.subplot(121), plt.imshow(img_rgb), plt.title('Original Image')
plt.subplot(122), plt.imshow(dst), plt.title('Output Image')
plt.show()
```

OUTPUT:



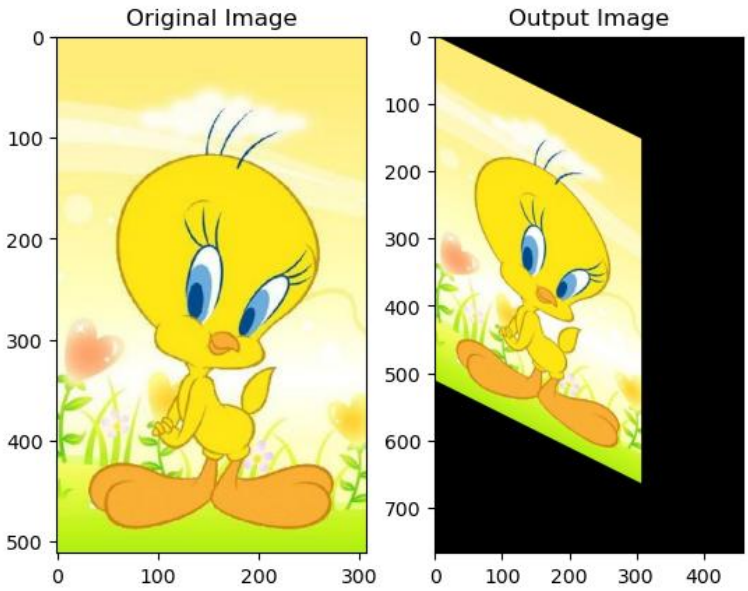
Practical – 1 (g)

Aim: Shearing Y-axis.

CODE:

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
img = cv2.imread("C:/Users/Student/Downloads/tweety.jpg")
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
rows, cols, channels = img_rgb.shape
M = np.float32([[1, 0, 0], [0.5, 1, 0], [0, 0, 1]])
dst = cv2.warpPerspective(img_rgb, M, (int(cols * 1.5), int(rows * 1.5)))
plt.subplot(121), plt.imshow(img_rgb), plt.title('Original Image')
plt.subplot(122), plt.imshow(dst), plt.title('Output Image')
plt.show()
```

OUTPUT:



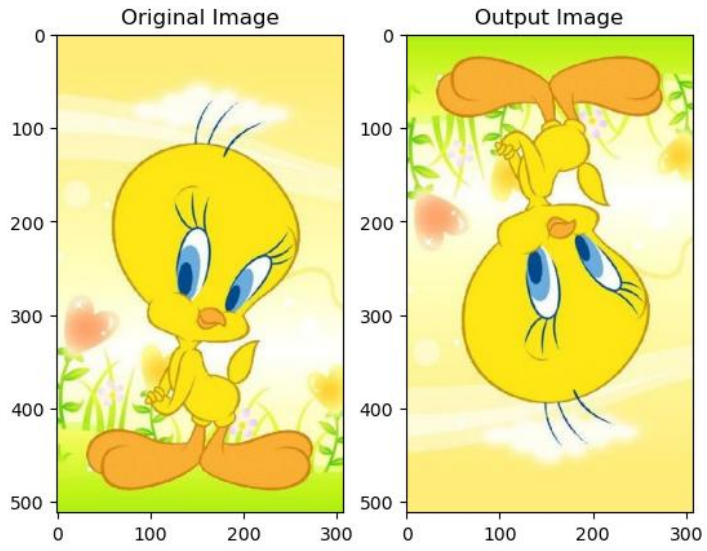
Practical – 1 (h)

Aim: Reflected Image.

CODE:

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
img = cv2.imread("C:/Users/Student/Downloads/tweety.jpg")
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
rows, cols, channels = img_rgb.shape
M = np.float32([[1, 0, 0], [0, -1, rows], [0, 0, 1]])
dst = cv2.warpPerspective(img_rgb, M, (cols, rows))
plt.subplot(121), plt.imshow(img_rgb), plt.title('Original Image')
plt.subplot(122), plt.imshow(dst), plt.title('Output Image')
plt.show()
```

OUTPUT:



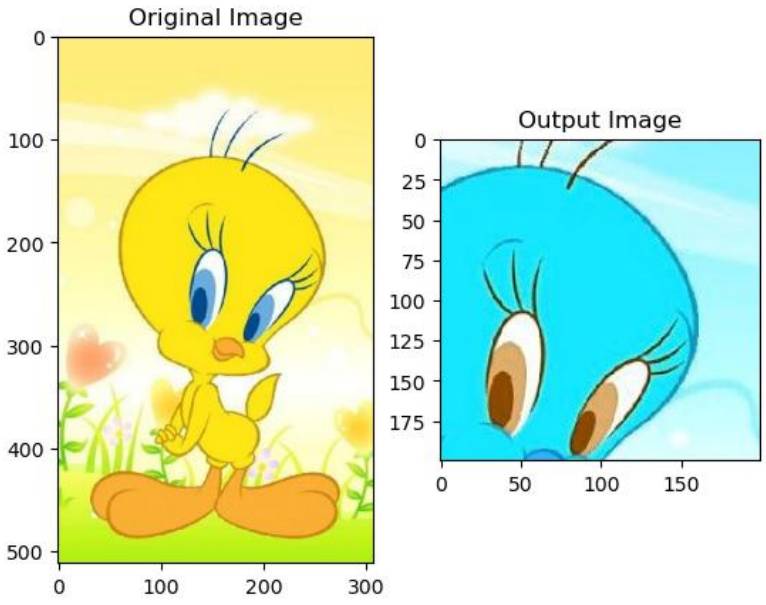
Practical – 1 (i)

Aim: Cropped Image.

CODE:

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
img = cv2.imread("C:/Users/Student/Downloads/tweety.jpg")
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
rows,  cols, channels = img_rgb.shape
dst = img[100:300, 100:300]
plt.subplot(121), plt.imshow(img_rgb), plt.title('Original Image')
plt.subplot(122), plt.imshow(dst), plt.title('Output Image')
plt.show()
```

OUTPUT:



Practical 2

Aim: Perform Image Stitching.

CODE:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Load images
img1 = cv2.imread("C:/Users/admin/Downloads/Right.jpeg")
img2 = cv2.imread("C:/Users/admin/Downloads/Left.jpeg")
# Convert to grayscale
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
# SIFT feature detector
sift = cv2.SIFT_create()
kp1, des1 = sift.detectAndCompute(gray1, None)
kp2, des2 = sift.detectAndCompute(gray2, None)
# BFMatcher with KNN
bf = cv2.BFMatcher()
matches = bf.knnMatch(des1, des2, k=2)
# Apply Lowe's ratio test
good_matches = []
for m, n in matches:
    if m.distance < 0.5 * n.distance:
        good_matches.append(m)
if len(good_matches) > 8:
    src_pts = np.float32([kp1[m.queryIdx].pt for m in good_matches]).reshape(-1, 1, 2)
    dst_pts = np.float32([kp2[m.trainIdx].pt for m in good_matches]).reshape(-1, 1, 2)
    # Compute homography
    H, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
    # Get dimensions for output
    height, width, _ = img2.shape
    panorama_width = width + img1.shape[1]
    # Warp first image
    result = cv2.warpPerspective(img1, H, (panorama_width, height))
    result[0:height, 0:width] = img2  # Overlay second image
    # Save and display
    cv2.imwrite("C:/Users/admin/Downloads/practical2/result.jpg", result)
    plt.imshow(cv2.cvtColor(result, cv2.COLOR_BGR2RGB))
    plt.title("Stitched Panorama")
    plt.axis("off")
    plt.show()
```

```
else:
    print("Not enough keypoints found for stitching.")
#Put right image path first as input and left second
```

OUTPUT:

Practical – 3

Aim: Perform Camera Calibration.

CODE:

```
import numpy as np
import cv2 as cv
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)
objp = np.zeros((6*7, 3), np.float32)
objp[:, :2] = np.mgrid[0:7, 0:6].T.reshape(-1, 2)
objpoints = []
imgpoints = []
image_paths = ["C:/Users/Student/Downloads/Chess.jpeg"
for fname in image_paths:
 img = cv.imread(fname)
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
 ret, corners = cv.findChessboardCorners(gray, (7,6), None)
 if ret:
 objpoints.append(objp)
 corners2 = cv.cornerSubPix(gray, corners, (11, 11), (-1, -1), criteria)
 imgpoints.append(corners2)
 cv.drawChessboardCorners(img, (7,6), corners, ret)
 cv.imshow('img', img)
cv.waitKey(500)
cv.destroyAllWindows()
# Camera calibration
ret, mtx, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints, gray.shape[::-1], None, None)
# Print calibration results
print("Camera matrix: ")
print(mtx)
print("Distortion coefficients: ")
print(dist)
print("Rotation Vectors: ")
print(rvecs)
print("Translation Vectors: ")
print(tvecs)
# Read an image for undistortion
undistort_img_path = "C:/Users/Student/Downloads/Chess.jpeg"
img = cv.imread(undistort_img_path)
h, w = img.shape[:2]
newcameramtx, roi = cv.getOptimalNewCameraMatrix(mtx, dist, (w, h), 1, (w, h))
dst = cv.undistort(img, mtx, dist, None, newcameramtx)
```

```
x, y, w, h = roi
dst = dst[y:y+h, x:x+w]
# Save the undistorted image
cv.imwrite('C:/Users/Student/Downloads/calibresult.png', dst)
print("Undistorted image saved as calibresult.png")
```

OUTPUT:

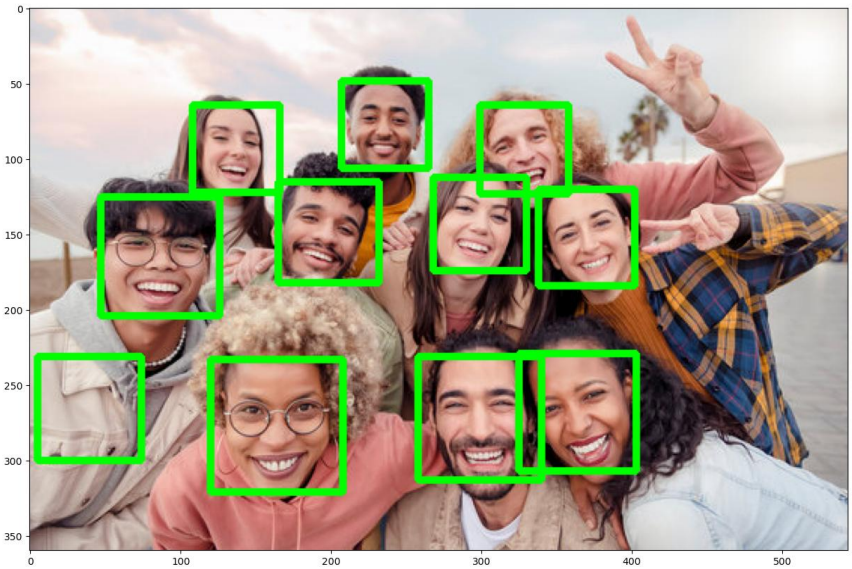
Practical – 4 (A)

Aim: Perform the following Face detection.

CODE:

```
import cv2
import matplotlib.pyplot as plt
imagePath = 'C:/Users/Student/Downloads/group.jpg'
img = cv2.imread(imagePath)
print(img.shape)
gray_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
face_classifier = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
face = face_classifier.detectMultiScale(gray_image, scaleFactor=1.1, minNeighbors=5, minSize=(40, 40))
for (x, y, w, h) in face:
 cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 4)
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.figure(figsize=(20,10))
plt.imshow(img_rgb)
plt.show()
```

OUTPUT:



Practical – 4 B

Aim: Object detection

CODE:

```
import cv2
from matplotlib import pyplot as plt
# Read the image
image = cv2.imread("C:/Users/Student/Downloads/stop.jpg")
# Convert to grayscale
image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Convert to RGB
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
# Load the pre-trained classifier
xml_data = cv2.CascadeClassifier('C:/Users/Student/Downloads/stop_data.xml')
# Detect objects (e.g., stop signals)
detecting = xml_data.detectMultiScale(image_gray, minSize=(30, 30))
# Count the number of detections
amount_detecting = len(detecting)
# If detection is found, draw rectangles around the detected objects
if amount_detecting != 0:
for (a, b, width, height) in detecting:
 cv2.rectangle(image_rgb, (a, b), (a + width, b + height), (0, 255, 0), 9)
# Show the image with detections
plt.imshow(image_rgb)
plt.show()
```

OUTPUT:

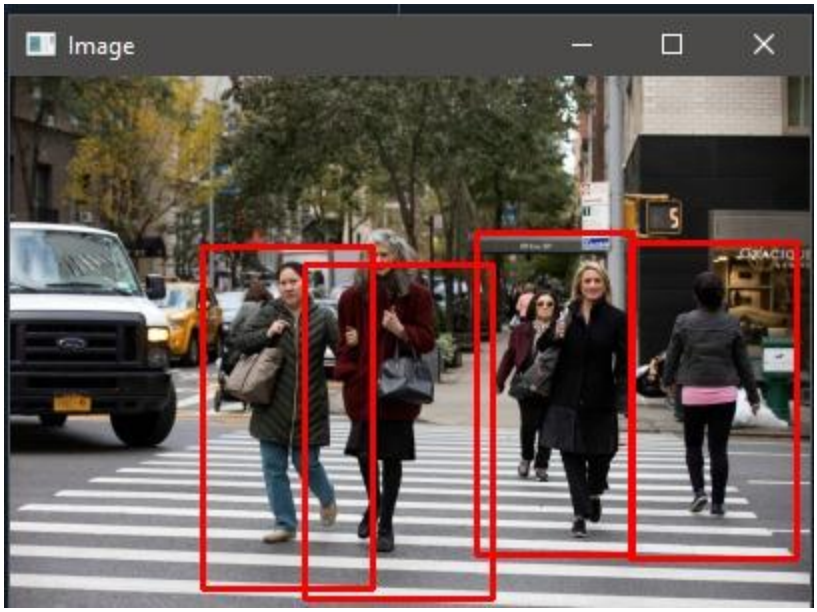
Practical – 4 C

Aim: Perform the following Pedestrian detection.

CODE:

```
import cv2
import imutils
hog = cv2.HOGDescriptor()
hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())
image = cv2.imread('C:/Users/Administrator/Downloads/pedestrain.jpeg')
image = imutils.resize(image, width=min(400, image.shape[1]))
(regions, _) = hog.detectMultiScale (image, winStride= (4, 4), padding= (4, 4), scale=1.05)
for (x, y, w, h) in regions:
    cv2.rectangle (image, (x, y), (x + y, y + h), (0, 0, 255), 2)
cv2.imshow("Image", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



Practical – 4 D

Aim: Perform the following Face Recognition.

CODE:

```
import cv2
import numpy as np
import face_recognition
# Load the first image (BGR format)
img_bgr = face_recognition.load_image_file('C:/Users/Student/Downloads/Practical 4d.jfif')
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
# Display the images
cv2.imshow('BGR', img_bgr)
cv2.imshow('RGB', img_rgb)
cv2.waitKey(0)
# Load and process the second image (for comparison)
img_modi = face_recognition.load_image_file('C:/Users/Student/Downloads/Practical 4d.jfif')
img_modi_rgb = cv2.cvtColor(img_modi, cv2.COLOR_BGR2RGB)
```

```
# Locate the face in the second image
face = face_recognition.face_locations(img_modi_rgb)[0]
# Create a copy of the image to annotate
copy = img_modi_rgb.copy()
cv2.rectangle(copy, (face[3], face[0]), (face[1], face[2]), (255, 0, 255), 2)
# Show the copy with the face rectangle
cv2.imshow('Copy', copy)
cv2.imshow('I am Steve Rogers', img_modi_rgb)
cv2.waitKey(0)
# Load the first image for encoding
img_modi = face_recognition.load_image_file('C:/Users/Student/Downloads/Practical 4d.jfif')
img_modi_rgb = cv2.cvtColor(img_modi, cv2.COLOR_BGR2RGB)
# Get the face location and encoding of the first image
face = face_recognition.face_locations(img_modi_rgb)[0]
train_encode = face_recognition.face_encodings(img_modi_rgb)[0]
# Load the test image for comparison
test = face_recognition.load_image_file('C:/Users/Student/Downloads/Practical 4d.jfif')
test_rgb = cv2.cvtColor(test, cv2.COLOR_BGR2RGB)
# Get the encoding of the test image
test_encode = face_recognition.face_encodings(test_rgb)[0]
# Compare the two face encodings
print(face_recognition.compare_faces([train_encode], test_encode))
# Draw a rectangle around the face in the image
cv2.rectangle(img_modi_rgb, (face[3], face[0]), (face[1], face[2]), (255, 0, 255), 1)
# Show the result
cv2.imshow('I am Steve Rogers', img_modi_rgb)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:

```
In [2]: runfile('C:/Users/Student/.spyder-py3/history.py', wdir='C:/Users/Student/.spyder-py3')
[True]
```

Practical – 5

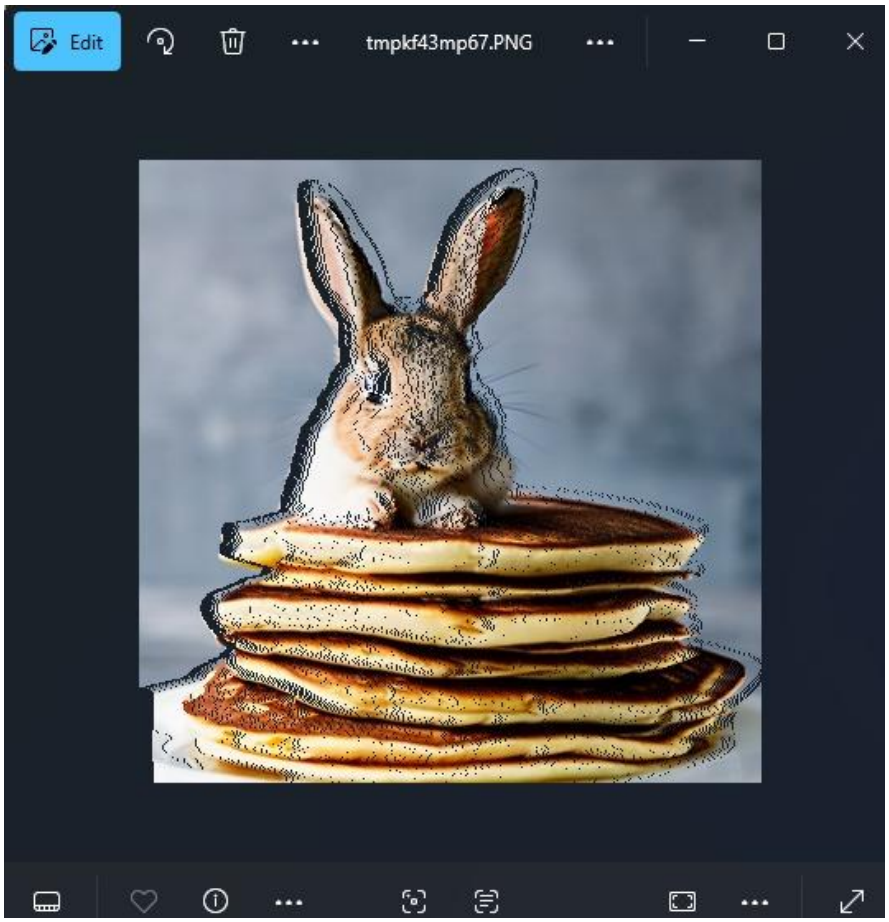
Aim: Construct 3D model from Images.

CODE:

```
from PIL import Image
import numpy as np
def shift_image(img, depth_img, shift_amount=10):
    img = img.convert("RGBA")  # Convert the image to RGBA format
    data = np.array(img)  # Get image data as an array
    depth_img = depth_img.convert("L")  # Convert depth image to grayscale (L mode)
    depth_data = np.array(depth_img)  # Get depth data as an array
    # Calculate the shift amounts based on depth data
    deltas = ((depth_data / 255.0) * float(shift_amount)).astype(int)
    # Initialize an empty array for the shifted image
    shifted_data = np.zeros_like(data)
```

```
    height, width, _ = data.shape  # Get image dimensions
    for y in range(height):
        for x in range(width):
            dx = deltas[y, x]  # Get the shift for the current pixel
            if 0 <= x + dx < width:  # Ensure the pixel is within bounds
                shifted_data[y, x + dx] = data[y, x]
    # Convert the shifted data back to an image
    shifted_image = Image.fromarray(shifted_data.astype(np.uint8))
    return shifted_image
# Load your images
img = Image.open("C:/Users/Admin/Downloads/bunny-cake.png")
depth_img = Image.open("C:/Users/Admin/Downloads/bunny-cake-alpha.png")
# Ensure both images are the same size
if img.size != depth_img.size:
    depth_img = depth_img.resize(img.size)
# Shift the image based on depth map
shifted_img = shift_image(img, depth_img, shift_amount=10)
# Show the result
shifted_img.show()
```

OUTPUT:



Practical – 6

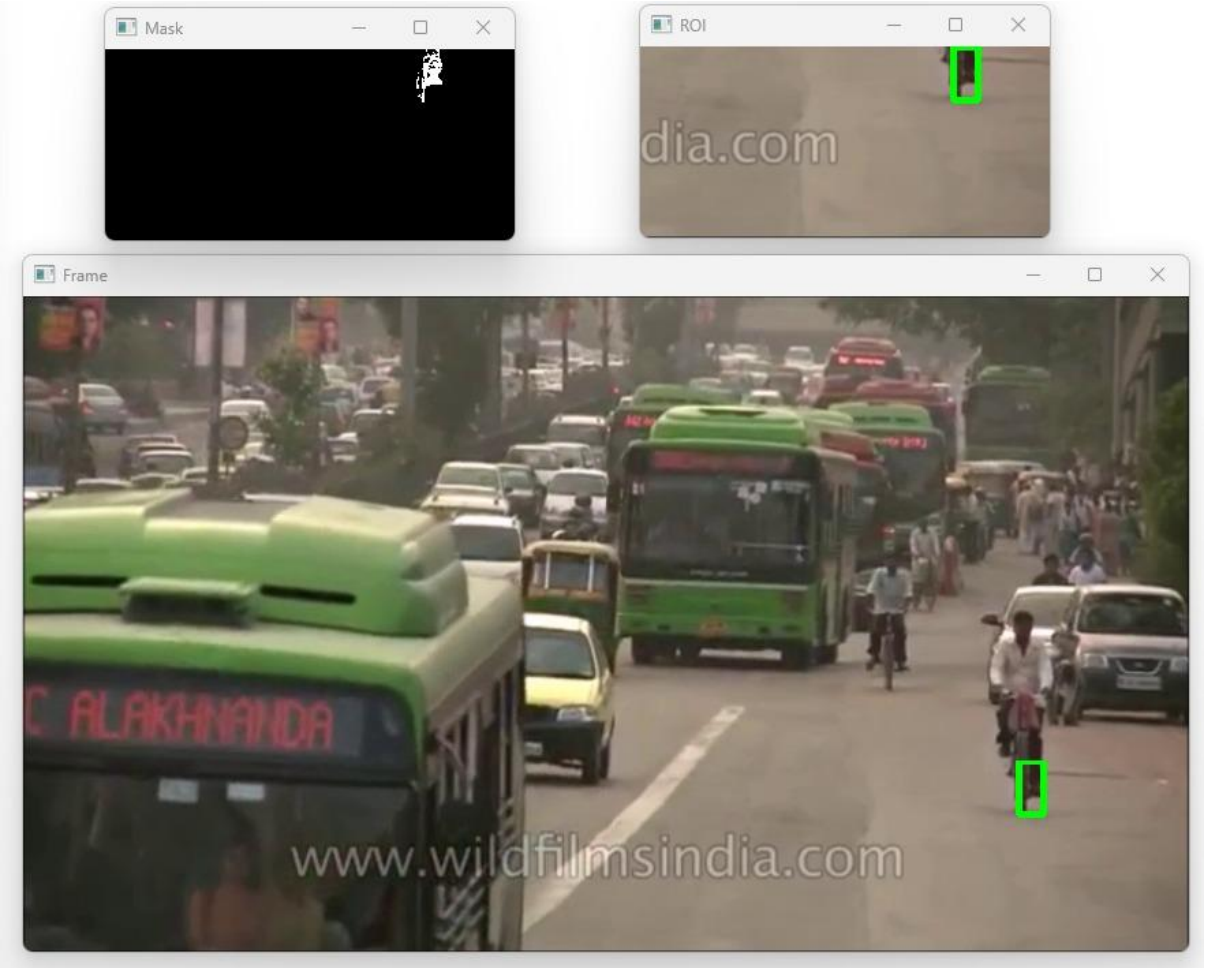
Mahesh Tanaji Chavan MSc-IT (Part-1) Roll. No:2024ITI2

Aim: Implement object detection and Tracking from video.
(Count number of Faces using Python)

CODE:

```
import sys
import cv2
sys.path.append('C:/Users/Admin/.spyder-py3')
from tracker import EuclideanDistTracker  # Make sure the tracker.py file is available
tracker = EuclideanDistTracker()
#tracker = cv2.TrackerCSRT_create()
# Video capture
cap = cv2.VideoCapture("C:/Users/Admin/Downloads/traffic-mini.mp4")
# Background subtractor
object_detector = cv2.createBackgroundSubtractorMOG2(history=100, varThreshold=40)
while True:
    ret, frame  = cap.read()
    if not ret:
        break  # Break the loop if the video ends
    height, width, _ = frame.shape
    print(height, width)
    # Region of Interest (ROI)
    roi = frame[340:720, 500:800]
    mask = object_detector.apply(roi)
    # Apply threshold to mask
    _, mask = cv2.threshold(mask, 254, 255, cv2.THRESH_BINARY)
    # Find contours
    contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    detections = []
    for cnt in contours:
        area = cv2.contourArea(cnt)
        if area > 100:
            x, y, w, h = cv2.boundingRect(cnt)
            cv2.rectangle(roi, (x, y), (x + w, y + h), (0, 255, 0), 2)
            detections.append([x, y, w, h])
    # Update tracker with current detections
    boxes_ids = tracker.update(detections)
    # Display the IDs on the tracked objects
    for box_id in boxes_ids:
        x, y, w, h, id = box_id
        cv2.putText(roi, str(id), (x, y - 15), cv2.FONT_HERSHEY_PLAIN,  1, (255, 0, 0), 2)
        cv2.rectangle(roi, (x, y), (x + w, y + h), (0, 255, 0), 3)
    # Display output
    cv2.imshow("ROI",  roi)
    cv2.imshow("Frame",  frame)
    cv2.imshow("Mask",  mask)
    key = cv2.waitKey(30)
    if key == 27:  # Press 'Esc' to exit
        break
cap.release()
cv2.destroyAllWindows()
```

OUTPUT:



Practical 7

Aim: Perform Feature extraction using RANSAC.

CODE:

```
import cv2
import numpy as np
# Read the images
img1_color = cv2.imread("C:/Users/Student/Downloads/download.png")
img2_color = cv2.imread("C:/Users/Student/Downloads/download.png")
# Convert images to grayscale
img1 = cv2.cvtColor(img1_color, cv2.COLOR_BGR2GRAY)
img2 = cv2.cvtColor(img2_color, cv2.COLOR_BGR2GRAY)
# Get image dimensions
height, width = img2.shape
# Initialize ORB detector
orb_detector = cv2.ORB_create(5000)
# Detect keypoints and descriptors
kp1, d1 = orb_detector.detectAndCompute(img1, None)
kp2, d2 = orb_detector.detectAndCompute(img2, None)
# Create BFMatcher object with Hamming distance and cross-checking
matcher = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
# Match descriptors
matches = matcher.match(d1, d2)
# Sort matches based on distance
matches = sorted(matches, key=lambda x: x.distance)
# Retain top 90% best matches
matches = matches[:int(len(matches) * 0.9)]
no_of_matches = len(matches)
# Prepare points for homography
p1 = np.zeros((no_of_matches, 2))
p2 = np.zeros((no_of_matches, 2))
for i in range(no_of_matches):
    p1[i, :] = kp1[matches[i].queryIdx].pt
    p2[i, :] = kp2[matches[i].trainIdx].pt
# Compute homography matrix
homography, mask = cv2.findHomography(p1, p2, cv2.RANSAC)
# Warp the image using the homography matrix
transformed_img = cv2.warpPerspective(img1_color, homography, (width, height))
# Save the result
cv2.imwrite("C:/Users/Student/Downloads/output.jpg", transformed_img)
```

OUTPUT:

Aling.jpg

Reflection.jpg

OUTPUT

Practical 8

Aim: Perform Colorization.

CODE:

```
import numpy as np
import cv2
from cv2 import dnn
# File paths
proto_file = 'C:/Users/Admin/Downloads/colorization_deploy_v2.prototxt'
model_file = 'C:/Users/Admin/Downloads/colorization_release_v2.caffemodel'
hull_pts = 'C:/Users/Admin/Downloads/pts_in_hull.npy'
img_path = 'C:/Users/Admin/Downloads/ansel_adams.jpg'
# Load the model
net = dnn.readNetFromCaffe(proto_file, model_file)
kernel = np.load(hull_pts)
# Read and process the image
img = cv2.imread(img_path)
scaled = img.astype("float32") / 255.0
lab_img = cv2.cvtColor(scaled, cv2.COLOR_BGR2LAB)
# Set up the model for colorization
class8 = net.getLayerId("class8_ab")
conv8 = net.getLayerId("conv8_313_rh")
pts = kernel.transpose().reshape(2, 313, 1, 1)
net.getLayer(class8).blobs = [pts.astype("float32")]
net.getLayer(conv8).blobs = [np.full([1, 313], 2.606, dtype="float32")]
# Prepare the LAB image for prediction
resized = cv2.resize(lab_img, (224, 224))
L = cv2.split(resized)[0]  # L channel (lightness)
# Predict the ab channels
net.setInput(cv2.dnn.blobFromImage(L))
ab_channel = net.forward()[0, :, :, :].transpose((1, 2, 0))
ab_channel = cv2.resize(ab_channel, (img.shape[1], img.shape[0]))
# Merge L channel with ab channels
L = cv2.split(lab_img)[0]
colorized = np.concatenate((L[:, :, np.newaxis], ab_channel), axis=2)
# Convert LAB to BGR
colorized = cv2.cvtColor(colorized, cv2.COLOR_LAB2BGR)
colorized = np.clip(colorized, 0, 1)
colorized = (255 * colorized).astype("uint8")
# Resize for comparison
img = cv2.resize(img, (250, 500))
colorized = cv2.resize(colorized, (250, 500))
# Display the result
result = cv2.hconcat([img, colorized])
cv2.imshow("Grayscale -> Color", result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:

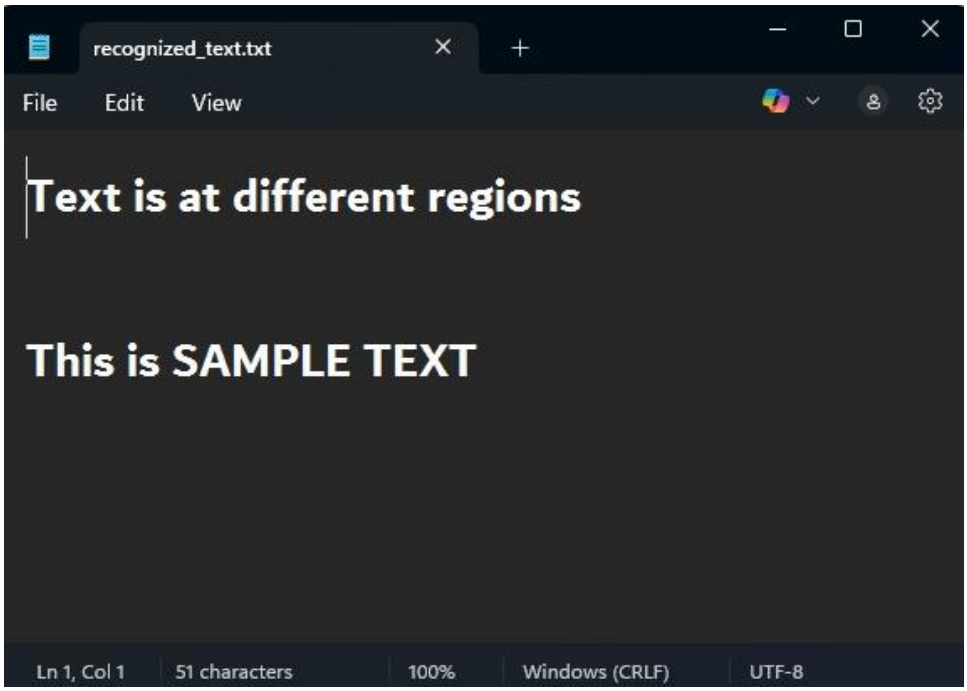
Practical 9

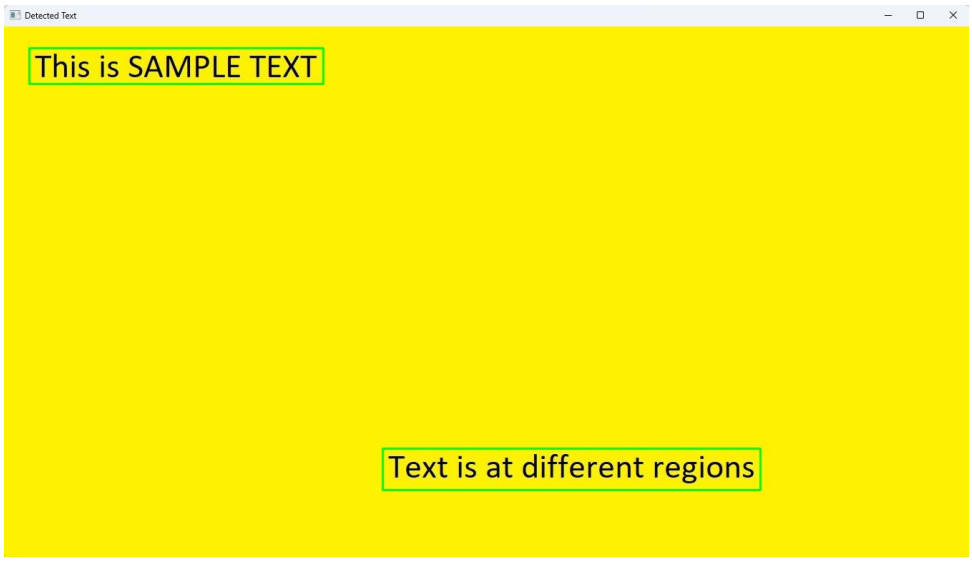
Aim: Perform Text Detection and Recognition

CODE:

```
import cv2
import pytesseract
# Set the path to Tesseract executable
pytesseract.pytesseract.tesseract_cmd = 'C:/Program Files/Tesseract-OCR/tesseract.exe'
# Read the image
img = cv2.imread('C:/Users/Admin/Downloads/sample.jpg')
# Convert to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Apply thresholding
ret, threshl = cv2.threshold(gray, 0, 255, cv2.THRESH_OTSU | cv2.THRESH_BINARY_INV)
# Define a rectangular kernel for dilation
rect_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (18, 18))
# Apply dilation to the thresholded image
dilation = cv2.dilate(threshl, rect_kernel, iterations=1)
# Find contours in the dilated image
contours, hierarchy = cv2.findContours(dilation, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
# Make a copy of the original image for drawing rectangles
im2 = img.copy()
# Open the output text file once
with open('C:/Users/Admin/Downloads/recognized_text.txt', 'w+') as file:
    for cnt in contours:
        # Get bounding box of each contour
        x, y, w, h = cv2.boundingRect(cnt)
        # Draw a rectangle around the detected text region
        cv2.rectangle(im2, (x, y), (x + w, y + h), (0, 255, 0), 2)
        # Crop the detected text region
        cropped = im2[y:y + h, x:x + w]
        # Use Tesseract to extract text from the cropped region
        text = pytesseract.image_to_string(cropped, config='--psm 6')  # --psm 6 assumes a single uniform block of
text
        # Write the recognized text to the output file
        if text.strip():  # Only write non-empty text
            file.write(text)
            file.write("\n")
# Optionally, display the result with bounding boxes drawn
cv2.imshow('Detected Text', im2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:


recognized_text.txt
File
Edit
View
Text is at different regions
This is SAMPLE TEXT
Ln 1, Col 1
51 characters
100%
Windows (CRLF)
UTF-8


Detected Text
This is SAMPLE TEXT
Text is at different regions

Practical 10

Aim: Perform Image matting and composition

CODE:

```
import cv2
import numpy as np
image_path = "C:/Users/Admin/Downloads/girl.jpg"
background_path = "C:/Users/Admin/Downloads/home.jpeg"
output_path = 'C:/Users/Admin/Downloads/result.jpeg'
def grabcut_matting(image_path, background_path, output_path):
    # Load the input image and background
    img = cv2.imread(image_path)
    bg = cv2.imread(background_path)
    # Check if images are loaded successfully
    if img is None:
        print(f'Error loading image: {image_path}")
        return
    if bg is None:
        print(f'Error loading background: {background_path}")
        return
    # Resize background to match the input image size
    bg = cv2.resize(bg, (img.shape[1], img.shape[0]))
    # Create initial mask
    mask = np.zeros(img.shape[:2], np.uint8)
    # Define a rectangle containing the foreground object (manually adjustable)
    rect = (50, 50, img.shape[1]-100, img.shape[0]-100)
    # Allocate memory for models (needed by GrabCut)
    bgdModel = np.zeros((1, 65), np.float64)
    fgdModel = np.zeros((1, 65), np.float64)
    # Apply GrabCut
    cv2.grabCut(img, mask, rect, bgdModel, fgdModel, 5, cv2.GC_INIT_WITH_RECT)
    # Prepare the mask for compositing
    mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')
    mask3 = cv2.merge([mask2, mask2, mask2])
    # Extract the foreground
    foreground = img * mask3
    cv2.imshow('Foreground', foreground)

    # Extract the background where the mask is 0
    background = bg * (1 - mask3)
    # Combine foreground and new background
    result = cv2.add(foreground, background)
    # Save the result to output path
```

```
    cv2.imwrite(output_path, result)
    cv2.imshow('Composited Image', result)
    cv2.waitKey(0)
# Call the function with paths
grabcut_matting(image_path, background_path, output_path)
```

OUTPUT: