**1 practical**
**perform geometric transformation**

**1 download test.jpg AND STORE WHERE the python file will be stored**

then use this code

```
import cv2
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

# Load local image (replace 'test.jpg' with your image file)
image_path = "test.jpg"
image = cv2.imread(image_path)

if image is None:
    print("Image not found. Please check the path or file name.")
    exit()

# Convert BGR to RGB for matplotlib display
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# 1. Original Image
plt.imshow(image_rgb)
plt.title("Original Image")
plt.axis('off')
plt.show()

# 2. Grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
plt.imshow(gray, cmap='gray')
```

```
plt.title("Grayscale Image")
plt.axis('off')
plt.show()

# 3. Gaussian Blur + Canny Edges
blurred = cv2.GaussianBlur(gray, (5, 5), 1.4)
edges = cv2.Canny(blurred, 100, 200)
plt.imshow(edges, cmap='gray')
plt.title("Canny Edge Detection")
plt.axis('off')
plt.show()

# 4. Draw Rectangle
img_copy = image.copy()
cv2.rectangle(img_copy, (30, 30), (130, 130), (0, 255, 0), 3)
plt.imshow(cv2.cvtColor(img_copy, cv2.COLOR_BGR2RGB))
plt.title("Rectangle on Image")
plt.axis('off')
plt.show()

# 5. Resize
resized = cv2.resize(image, (300, 300))
plt.imshow(cv2.cvtColor(resized, cv2.COLOR_BGR2RGB))
plt.title("Resized Image")
plt.axis('off')
plt.show()

# 6. Thresholding
_, thresholded = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)
plt.imshow(thresholded, cmap='gray')
plt.title("Thresholded Image")
```

```
plt.axis('off')
plt.show()

# 7. Rotation using PIL
img_pil = Image.open(image_path)
rotated = img_pil.rotate(90)
plt.imshow(rotated)
plt.title("Rotated Image (90°)")
plt.axis('off')
plt.show()
```

2 practical
perform image stitching

- Make sure you have 3 overlapping images saved as:

**image1.jpg**, **image2.jpg**, and **image3.jpg** in the same folder as your .py file.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load 3 local images
img1 = cv2.imread("image1.jpg")
img2 = cv2.imread("image2.jpg")
img3 = cv2.imread("image3.jpg")

if img1 is None or img2 is None or img3 is None:
    print("One or more images not found. Please check the file names.")
    exit()

# Convert BGR to RGB for display
def show_image(img, title):
```

```
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.title(title)
    plt.axis('off')
    plt.show()

# Show individual images
show_image(img1, "Image 1")
show_image(img2, "Image 2")
show_image(img3, "Image 3")

# Convert to grayscale
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
gray3 = cv2.cvtColor(img3, cv2.COLOR_BGR2GRAY)

# SIFT feature detection
sift = cv2.SIFT_create()
kp1, des1 = sift.detectAndCompute(gray1, None)
kp2, des2 = sift.detectAndCompute(gray2, None)
kp3, des3 = sift.detectAndCompute(gray3, None)

# FLANN based matcher
index_params = dict(algorithm=1, trees=5)
search_params = dict(checks=50)
flann = cv2.FlannBasedMatcher(index_params, search_params)

# Match descriptors
matches1to2 = flann.knnMatch(des1, des2, k=2)
matches2to3 = flann.knnMatch(des2, des3, k=2)

# Filter good matches
```

```
good_matches1to2 = []
good_matches2to3 = []

for m, n in matches1to2:
    if m.distance < 0.7 * n.distance:
        good_matches1to2.append(m)

for m, n in matches2to3:
    if m.distance < 0.7 * n.distance:
        good_matches2to3.append(m)

# Draw matches
img_matches1to2 = cv2.drawMatches(img1, kp1, img2, kp2, good_matches1to2[:10], None,
flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
img_matches2to3 = cv2.drawMatches(img2, kp2, img3, kp3, good_matches2to3[:10], None,
flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

# Show matches
show_image(img_matches1to2, "Matches: Image 1 & 2")
show_image(img_matches2to3, "Matches: Image 2 & 3")

# Find Homographies
pts1 = np.float32([kp1[m.queryIdx].pt for m in good_matches1to2]).reshape(-1, 1, 2)
pts2 = np.float32([kp2[m.trainIdx].pt for m in good_matches1to2]).reshape(-1, 1, 2)

pts3 = np.float32([kp2[m.queryIdx].pt for m in good_matches2to3]).reshape(-1, 1, 2)
pts4 = np.float32([kp3[m.trainIdx].pt for m in good_matches2to3]).reshape(-1, 1, 2)

H1, _ = cv2.findHomography(pts2, pts1, cv2.RANSAC)
H2, _ = cv2.findHomography(pts3, pts4, cv2.RANSAC)

# Warp and Stitch
```

```
height, width = img1.shape[:2]

img2_warped = cv2.warpPerspective(img2, H1, (width * 2, height))
img2_warped[0:height, 0:width] = img1

img3_warped = cv2.warpPerspective(img3, H2, (width * 3, height))
img3_warped[0:height, 0:width * 2] = img2_warped[0:height, 0:width * 2]

# Final stitched image
final_stitched = img3_warped
show_image(final_stitched, "Final Stitched Panorama")
```

3rd practical
camera calibration

📷 **Requirement:**

You must have a **chessboard calibration image** (like the one in OpenCV docs).
Save it as: chessboard.jpg in the same folder as your .py file.

link for image
https://docs.opencv.org/4.x/calib_radial.jpg

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load a local chessboard image
image_path = "chessboard.jpg"
img = cv2.imread(image_path)

if img is None:
```

```
    print("Image not found. Please place 'chessboard.jpg' in the same folder.")
    exit()

# Resize image for faster processing
img = cv2.resize(img, (640, 480))

# Chessboard pattern size (number of internal corners)
chessboard_size = (9, 6)
square_size = 1.0  # Any unit (e.g., 1cm or 1inch)

# Prepare 3D object points in real-world space
objp = np.zeros((chessboard_size[0] * chessboard_size[1], 3), np.float32)
objp[:, :2] = np.indices(chessboard_size).T.reshape(-1, 2) * square_size

obj_points = []  # 3D real-world points
img_points = []  # 2D image points

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

# Detect chessboard corners
print("Starting chessboard detection...")
ret, corners = cv2.findChessboardCorners(gray, chessboard_size, None)
print(f"Chessboard detection result: {ret}")

# If found, proceed
if ret:
    obj_points.append(objp)
    img_points.append(corners)

    # Draw corners
```

```
    img = cv2.drawChessboardCorners(img, chessboard_size, corners, ret)
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.title("Detected Chessboard Corners")
    plt.axis('off')
    plt.show()
else:
    print("Chessboard corners not found.")
    exit()

# Camera Calibration
ret, camera_matrix, dist_coeffs, rvecs, tvecs = cv2.calibrateCamera(
    obj_points, img_points, gray.shape[::-1], None, None)

# Print camera parameters
print("\nCamera matrix:\n", camera_matrix)
print("\nDistortion coefficients:\n", dist_coeffs)

# Undistort image
h, w = img.shape[:2]
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(camera_matrix, dist_coeffs, (w, h), 1,
(w, h))
undistorted_img = cv2.undistort(img, camera_matrix, dist_coeffs, None, new_camera_matrix)

# Display original vs undistorted
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title("Original Image")
plt.axis('off')

plt.subplot(1, 2, 2)
```

```
plt.imshow(cv2.cvtColor(undistorted_img, cv2.COLOR_BGR2RGB))
plt.title("Undistorted Image")
plt.axis('off')

plt.show()
```

4th practical

Perform the following
face detection
object detection

Pedestrian detection

⚙️ **Prerequisites (Install only once)**

Open Command Prompt and run:

bash

CopyEdit

```
pip install opencv-python matplotlib
```

Make sure you also have:

- A test image (e.g., face.jpg, object.jpg, pedestrian.jpg) for each part

CANCEL 4

5TH PRACTICAL

IMPLEMENT OBJECT DETECTion and tracking from video

Cancel

✅ Practical 6: Perform Feature Extraction using RANSAC

⚙️ **Prerequisites (Install Once)**

Open Command Prompt and run:

bash

CopyEdit

pip install opencv-python numpy matplotlib pillow

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

# Load images (grayscale)
img1 = cv2.imread("image1.jpg", cv2.IMREAD_GRAYSCALE)
img2 = cv2.imread("image2.jpg", cv2.IMREAD_GRAYSCALE)

if img1 is None or img2 is None:
    print("Error: One or both images not found.")
    exit()

# Initialize ORB detector
orb = cv2.ORB_create(nfeatures=1000)

# Detect keypoints and descriptors
keypoints1, descriptors1 = orb.detectAndCompute(img1, None)
keypoints2, descriptors2 = orb.detectAndCompute(img2, None)

print("Number of keypoints in image 1:", len(keypoints1))
print("Number of keypoints in image 2:", len(keypoints2))
```

```
# Match using Brute-Force matcher with Hamming distance
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
matches = bf.match(descriptors1, descriptors2)

# Sort matches by distance
matches = sorted(matches, key=lambda x: x.distance)

print("Number of matches found:", len(matches))

# Extract location of good matches
src_pts = np.float32([keypoints1[m.queryIdx].pt for m in matches]).reshape(-1, 1, 2)
dst_pts = np.float32([keypoints2[m.trainIdx].pt for m in matches]).reshape(-1, 1, 2)

# Compute Homography using RANSAC
H, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)

print("Homography Matrix (H):\n", H)

# Draw top 10 matches
img_matches = cv2.drawMatches(img1, keypoints1, img2, keypoints2, matches[:10], None,
                              flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

# Display result
plt.figure(figsize=(15, 8))
plt.imshow(img_matches, cmap='gray')
plt.title("Top 10 Feature Matches using ORB + RANSAC")
plt.axis('off')
plt.show()
```

✅ **Practical 7: Perform Text Detection and Recognition**

📚 **M.Sc. IT Sem 2 – Computer Vision**
🎯 Goal: Extract text from an image and draw bounding boxes around each character using **Tesseract OCR**

---

⚙️ **Prerequisites**

Before running, install the following:

🔹 **1. Python Libraries:**

bash

pip install pytesseract opencv-python pillow matplotlib

🔹 **2. Install Tesseract OCR (Windows Executable)**

- 📥 Download from: https://github.com/tesseract-ocr/tesseract
- Direct Windows installer:
  👉 https://github.com/UB-Mannheim/tesseract/wiki (direct download)

During install, note the install path, e.g.:

C:\Program Files\Tesseract-OCR\tesseract.exe

my case: C:\Program Files\Tesseract-OCR

also download text image

```
import cv2
import pytesseract
import numpy as np
from matplotlib import pyplot as plt
from PIL import Image

# Set path to tesseract.exe
pytesseract.pytesseract.tesseract_cmd = r"C:\Program Files\Tesseract-OCR\tesseract.exe"
 # <-- change if needed
```

```
# Load image
img = cv2.imread("text_image.jpg")  # Replace with your image file

if img is None:
    print("Image not found!")
    exit()

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

# Thresholding for better OCR
_, thresh = cv2.threshold(gray, 150, 255, cv2.THRESH_BINARY)

# Display thresholded image
plt.imshow(thresh, cmap='gray')
plt.title("Thresholded Image")
plt.axis('off')
plt.show()

# Perform OCR
text = pytesseract.image_to_string(thresh)
print("Extracted Text:\n", text)

# Draw bounding boxes around characters
h, w, _ = img.shape
boxes = pytesseract.image_to_boxes(img)

for b in boxes.splitlines():
    b = b.split()
    x, y, x2, y2 = int(b[1]), int(b[2]), int(b[3]), int(b[4])
```

```
    cv2.rectangle(img, (x, h - y), (x2, h - y2), (0, 255, 0), 2)

# Show final image with boxes
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title("Detected Characters")
plt.axis('off')
plt.show()
```

✅ **Practical 8: Image Matting and Compositing**

📚 **M.Sc. IT Sem 2 – Computer Vision**

🎯 Objective:

- Separate a foreground object using thresholding (matting)
- Blend it with a new background (compositing)

---

⚙️ **Prerequisites**

Install these packages first:

bash

CopyEdit

```
pip install opencv-python pillow numpy matplotlib
```

---

📁 **Required Files**

You need two images:

- foreground.jpg – image with subject (person/object)
- background.jpg – new background

Put them in the **same folder** as your Python file.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
from PIL import Image

# Load foreground and background
foreground_image = cv2.imread("foreground.jpg")
background_image = cv2.imread("background.jpg")

if foreground_image is None or background_image is None:
    print("One or both images not found.")
    exit()

# Resize background to match foreground
background_image_resized = cv2.resize(background_image, (foreground_image.shape[1],
foreground_image.shape[0]))

# Convert foreground to grayscale and apply threshold
foreground_gray = cv2.cvtColor(foreground_image, cv2.COLOR_BGR2GRAY)
_, mask = cv2.threshold(foreground_gray, 120, 255, cv2.THRESH_BINARY)

# Add alpha channel to foreground
foreground_with_alpha = cv2.cvtColor(foreground_image, cv2.COLOR_BGR2BGRA)
foreground_with_alpha[:, :, 3] = mask

# Display alpha-matted foreground
plt.imshow(cv2.cvtColor(foreground_with_alpha, cv2.COLOR_BGRA2RGBA))
plt.title("Foreground with Alpha Matte")
plt.axis("off")
plt.show()

# Normalize alpha channel to [0, 1]
alpha_channel = foreground_with_alpha[:, :, 3] / 255.0
```

```
# Extract RGB channels of foreground
foreground_rgb = foreground_with_alpha[:, :, :3]

# Blend images using alpha matte
blended = (alpha_channel[..., None] * foreground_rgb +
        (1 - alpha_channel[..., None]) * background_image_resized).astype(np.uint8)

# Display final composited image
plt.imshow(cv2.cvtColor(blended, cv2.COLOR_BGR2RGB))
plt.title("Composited Image")
plt.axis("off")
plt.show()

# Save the result
cv2.imwrite("composited_output.jpg", blended)
print("✅ Output saved as composited_output.jpg")
```