EN3160 Assignment 2 on Fitting and Alignment

GitHub Link :- https://github.com/Manimohan05/EN3160---Image-processing-Works--Assignments-Codes-/tree/main/Assignment-02

Index :- 200377M Name :- Manimohan T.

1. Question 01

In this question, using the knowledge on blob detection, i.e., using Laplacian of Gaussians and scale-space extrema detection, I will detect and draw circles in the sunflower field image. I have used the sunflower field image provided

Important part of the code

```
for sigma in np.linspace(min_sigma, max_sigma,
num_sigma):
    # Apply LoG (Laplacian of Gaussian) and find contours
    blurred = cv2.GaussianBlur(gray_image, (0, 0), sigma)
    laplacian = np.abs(cv2.Laplacian(blurred,
cv2.CV_64F))
    circle_mask = laplacian > threshold * laplacian.max()
    contours, _ =
cv2.findContours(circle_mask.astype(np.uint8),
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    # Fit circles to detected contours
    circles.extend([(tuple(map(int,
cv2.minEnclosingCircle(contour)[0])),
int(cv2.minEnclosingCircle(contour)[1]), sigma)
                    for contour in contours if
len(contour) >= 5])
# Sort the detected circles by radius in descending order
circles.sort(key=lambda x: -x[1])

# Report the parameters of the largest circle
center, radius, sigma = circles[0][:3]
print(f"Parameters of the largest circle:\nCenter:
{center}\nRadius: {radius}\nSigma value: {sigma}")

# Set the desired line thickness for drawn circles
line_thickness = 1

# Draw all detected circles with the specified line
thickness
output_image = cv2.cvtColor(gray_image,
cv2.COLOR_GRAY2BGR)
for center, radius, _ in circles:
    cv2.circle(output_image, center, radius, (0, 0, 255),
line_thickness)  # Display in red
```

The range of sigma values employed spanned from 0.9 to 3, with an increment of 3.5. The coordinates of the largest circle's center were (184, 253), and its radius measured 25 units.The sigma value associated with the largest circle was 2.58.

I performed the Laplacian of Gaussian operation on the image (LoG). Subsequently, I detected blobs in the processed image, and I eliminated redundant blobs by applying contour and sort functions.

figure 1:- Blob Detection

2. Question 02

In this task, I aim to fit a line and a circle to noisy data points that represent both geometries. I use RANSAC to identify inliers and employ an optimization approach to find the best-fitting models for the line and circle, addressing noise and outliers.

a) Important code of Line estimation

```
def ransac_line(data_points, num_iterations,
distance_threshold, min_inliers):
    best_fit = None
    best_inliers = []
    for _ in range(num_iterations):
        sample_indices =
np.random.choice(len(data_points), 2, replace=False)
        x1, y1 = data_points[sample_indices[0]]
        x2, y2 = data_points[sample_indices[1]]
        a, b, d = line_equation_from_points(x1,y1,x2, y2)
        a, b = a / np.linalg.norm([a, b]), b /
np.linalg.norm([a, b])
```

```
        distances = np.abs(a * data_points[:, 0] + b *
data_points[:, 1] - d)
        inliers = np.where(distances <
distance_threshold)[0]
        if len(inliers) >= min_inliers and len(inliers) >
len(best_inliers):
            best_fit, best_inliers = (a, b, d), inliers
    return best_fit, best_inliers
```

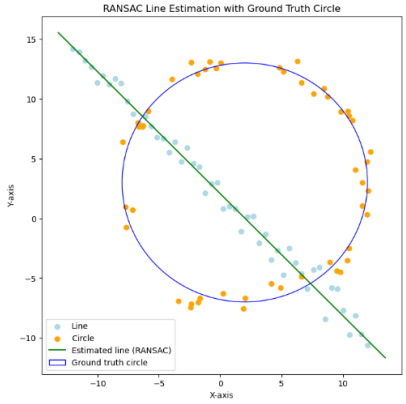

figure 2:- RANSAC Line Estimation

b)

```
# Identify the remnant points (not explained by the line)
remnant_indices = [i for i in range(len(X)) if i not in
line_inlier_indices]
remnant_points = X[remnant_indices]
```

I've set the radial error threshold at one-third of the circle's radius, and I require a consensus set with at least 40% of the remaining points (excluding line inliers). Then, I select the circle with the most inliers from this subset and estimate the best-fitting circle.

Important code of circle estimation

```
# Function to estimate a circle using RANSAC

def ransac_circle(X, iterations, threshold, min_inliers):
    best_model, best_inliers = None, []
    for _ in range(iterations):
        sample_indices = np.random.choice(len(X), 3,
replace=False)
        (x1, y1), (x2, y2), (x3, y3) = X[sample_indices]
        x_center, y_center, radius =
circle_equation_from_points(x1, y1, x2, y2, x3, y3)
        errors = np.abs(np.sqrt((X[:, 0] - x_center)**2 +
(X[:, 1] - y_center)**2) - radius)
        inliers = np.where(errors < threshold)[0]
        if len(inliers) >= min_inliers and len(inliers) >
len(best_inliers):
            best_model, best_inliers = (x_center,
y_center, radius), inliers
    return best_model, best_inliers
```

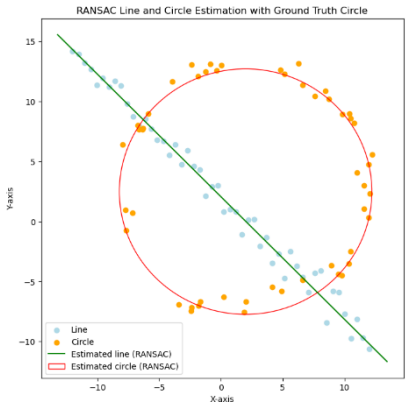

figure 3:- RANSAC Circle Estimation

c)

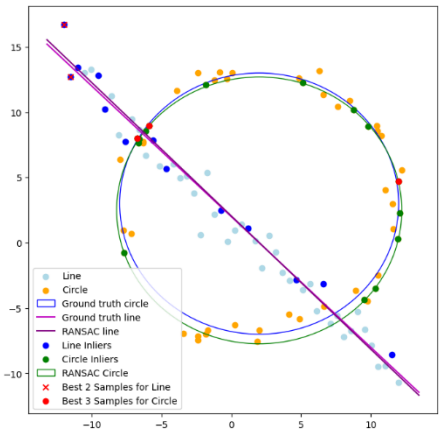

figure 4:- All plots

d) When fitting a circle as the initial step, it aims to identify inliers that closely adhere to a circular model by estimating the circle's center and radius. However, points farther from this estimated circle are typically considered outliers and are disregarded during circle fitting. Subsequently, when applying RANSAC to fit a line using the remaining points (which were initially outliers for the circle model), challenges arise. The circle-fitting step has already pruned data points that didn't conform to the circular model, potentially eliminating crucial linear points. Consequently, the resulting estimated line might not accurately capture the underlying linear structure due to the loss of significant data during the preceding circle-fitting process. The choice of fitting order should be made judiciously, considering the

data's characteristics and specific model requirements.

3. Question 03

In this task, i aim to superimpose an image, like a flag, onto an architectural scene. This is achieved by selecting four points on the architectural image, computing a homography transformation to map the flag onto the chosen plane, and then blending it seamlessly into the original image.

Important part of code

```
# Define the four points on the architectural image that
form a planar surface
# Replace these with the actual coordinates
points_on_architecture = np.array([[150, 210], [520,
290], [520, 510], [135, 520]], dtype=np.float32)

# Define the corresponding points on the flag image (in
the same order)
points_on_flag = np.array([[0, 0], [flag_image.shape[1],
0], [flag_image.shape[1], flag_image.shape[0]], [0,
flag_image.shape[0]]], dtype=np.float32)

# Compute the homography matrix using RANSAC (with
adjusted parameters)
homography_matrix, _ = cv2.findHomography(points_on_flag,
points_on_architecture, cv2.RANSAC, 5.0)

# Warp the flag image onto the architectural image using
the homography
flag_warped = cv2.warpPerspective(flag_image,
homography_matrix, (architectural_image.shape[1],
architectural_image.shape[0]))

# Blend the warped flag image with the architectural
image
result_image = cv2.addWeighted(architectural_image, 1,
flag_warped, 0.75, 0)
```

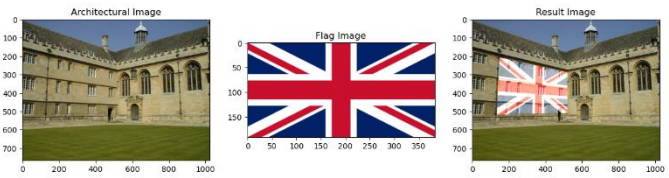

figure 5:- superimpose an image(Flag)

"The United Kingdom's flag has been blended to give the appearance of it hanging on a wall. Picture the moment when the highly anticipated LEO movie is released on the big screens in the theater. The iconic Sri Lankan cricket moment is strategically placed to create the illusion that the match is being broadcast on a television screen. To achieve this effect, I'm utilizing the 'findHomography' function to calculate the homography and 'warpPerspective' to distort the flag image."

figure 6:- superimpose an image(LEO)

figure 7:- superimpose an image(SL Cricket)

4. Question 04

In this task, i aim to seamlessly merge two graffiti images, img1.ppm and img5.ppm. I'll start by computing and matching SIFT features between the images, then calculate the homography using a custom RANSAC-based approach and compare it with the dataset's homography. Finally, I'll stitch img1.ppm onto img5.ppm to create a combined image.

a) important code

```
def sift_match(im1, im2):
    ratio_threshold = 0.87
    sift = cv.SIFT_create()
    kp1, desc1 = sift.detectAndCompute(im1, None)
    kp2, desc2 = sift.detectAndCompute(im2, None)
    matcher = cv.BFMatcher()
    matches = matcher.knnMatch(desc1, desc2, k=2)
    good_matches = [m for m, n in matches if m.distance <
ratio_threshold * n.distance]
    points1 = np.array([kp1[m.queryIdx].pt for m in
good_matches], dtype=np.float32)
    points2 = np.array([kp2[m.trainIdx].pt for m in
good_matches], dtype=np.float32)
```

```
    matched_img = cv.drawMatches(im1, kp1, im2, kp2,
good_matches, im2, flags=2)
    return points1, points2, matched_img
def ransac(matched_points):
    max_inliers = 0
    best_H = None
    for _ in range(10):
        random_indices =
random.sample(range(len(matched_points)), 3)
        random_points = [matched_points[i] for i in
random_indices]
        homography = calculate_homography(random_points)
        num_inliers = sum(calculate_loss(mp, homography)
< 3 for mp in matched_points)
        if num_inliers > max_inliers:
            max_inliers = num_inliers
            best_H = homography
    return best_H
```

figure 8:- SIFT Features Extraction for Images 1&5

b)

```
def calculate_homography(correspondences):

    temp_list = []
    for points in correspondences:
        p1 = np.matrix([points.item(0), points.item(1),
1])  # (x1,y1)
        p2 = np.matrix([points.item(2), points.item(3),
1])  # (x2,y2)
        a2 = [0, 0, 0, -p2.item(2) * p1.item(0), -
p2.item(2) * p1.item(1), -p2.item(2) * p1.item(2),
              p2.item(1) * p1.item(0), p2.item(1) *
p1.item(1), p2.item(1) * p1.item(2)]
        a1 = [-p2.item(2) * p1.item(0), -p2.item(2) *
p1.item(1), -p2.item(2) * p1.item(2), 0, 0, 0,
              p2.item(0) * p1.item(0), p2.item(0) *
p1.item(1), p2.item(0) * p1.item(2)]
        temp_list.extend([a1, a2])
    assemble_matrix = np.matrix(temp_list)
    # SVD composition
    u, s, v = np.linalg.svd(assemble_matrix)
    # Reshape the minimum singular value into a 3x3
matrix
    h = np.reshape(v[8], (3, 3))
    # Normalize
    h = (1 / h.item(8)) * h
    return h
```

Calculated Homography

[[4.45336997e+00 -3.43914107e+01 -5.28743489e+01]

 [3.10339607e+00 -2.79187195e+01 1.41503866e+01]

 [7.38391904e-03 -8.13467901e-02 1.00000000e+00]]

Reshaped Homography

[[-2.52220229e+01 3.11328500e+01 1.18016489e+02]

 [-2.46006502e+01 3.01506585e+01 1.48719063e+02]

 [-4.61457229e-02 5.51998164e-02 1.00000000e+00]]

The process of calculating homographies for image pairs revealed a significant discrepancy between the computed and provided homographies, specifically before multiplication. To address this, I individually computed homographies for five images and then obtained the final homography between the first and fifth images by multiplying these intermediate homographies sequentially. Upon comparing this final computed homography with the provided one, it became evident that they exhibit similarity.

c)

figure 9:- Stitch img1.ppm onto img5.ppm

We utilized the generated homography to combine Image 1 and Image 5, positioning Image 5 on top of the warped Image 1. As previously noted, finding strong matches between these two images proved to be challenging.

***