Department of Electronic and Telecommunication Engineering
University of Moratuwa

EN3160 - Image Processing and Machine Vision

Fitting and Alignment

210204R Hansindu K.A.A.

1 Blob Detection in Sunflower Field Using Laplacian of Gaussians

I used the Laplacian of Gaussians to detect blobs in the sunflower image by applying scale-space extrema detection.

```
# Apply Gaussian blur to reduce noise
blurred = cv.GaussianBlur(im, (9, 9), 0.1)
```

Listing 1: Gaussian blur is applied to reduce noise and enhance blob detection.

```
# Define parameters for blob detection
min_sigma = 3
max_sigma = 30
threshold = .1

# Detect blobs using Laplacian of Gaussians
blobs = blob_log(gray, min_sigma=min_sigma, max_sigma=max_sigma, threshold=threshold)
blobs[:, 2] = blobs[:, 2] * sqrt(2)  # Convert radii from sqrt(2) scaling
```

Listing 2: The LoG function computes the second derivative of the image, which is highly effective in detecting blobs as it finds areas in the image where pixel intensity changes rapidly. Here, we detect blobs over a range of scales by adjusting the parameter. The parameter controls the scale at which blobs are detected, with larger values of detecting larger structures.

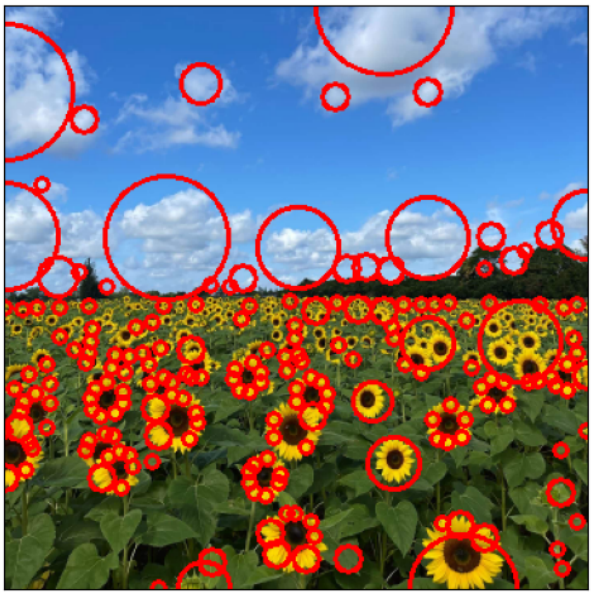


Figure 1: Detected circles in the sunflower field.

Parameters of the largest circles.

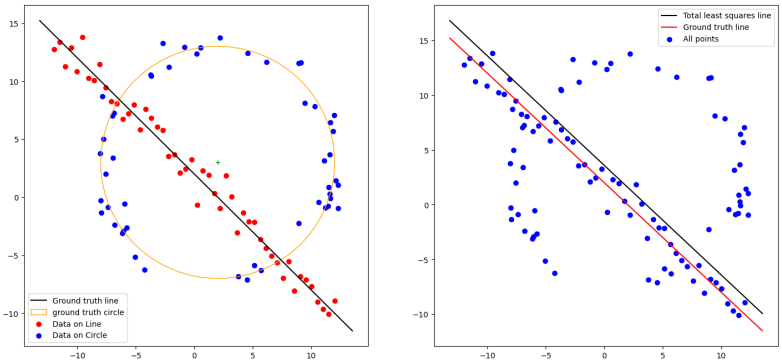
- Largest Circle Radius: 42.42640687119285
- Largest Circle Center Coordinates: (x: 234.0, y: 0.0)

2 Line Fitting to Noisy Data Using RANSAC

```
U = X_ - np.mean(X_, axis=0)

# Eigenvector of U^TU associated with the smallest eigenvalue
_, _, Vt = np.linalg.svd(np.transpose(U)@U)
a, b = Vt[-1, 0], Vt[-1, 1]
d = a*np.mean(X_[:,0]) + b*np.mean(X_[:,1])
best_model_line = np.array([a, b, d])
```

Listing 3: This code computes the normal vector to the line by solving the singular value decomposition (SVD) of the data matrix, ensuring that the vector corresponds to the line's smallest error direction.

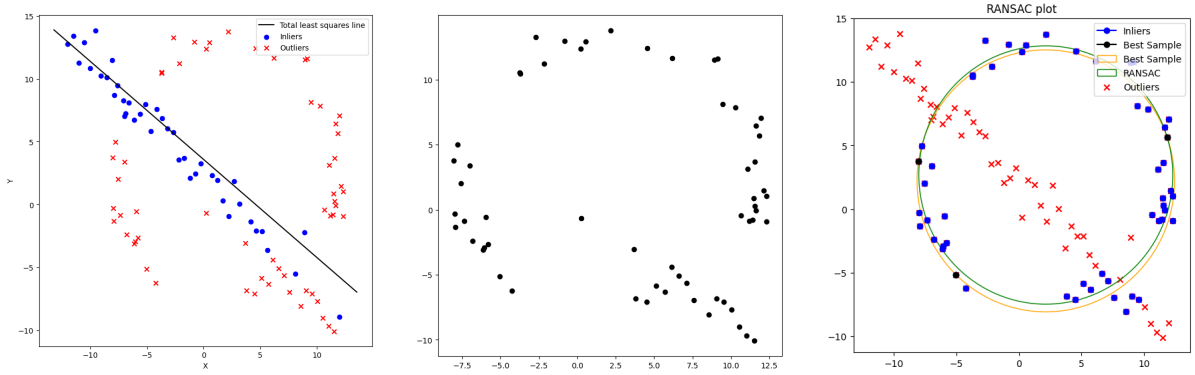


(a) Noisy points generated for both the line and the circle.

(b) Comparison of the noisy dataset generatedand the line fitted using RANSAC.

Figure 2: Comparison of the noisy dataset generated and the line fitted using RANSAC.

Subtracted the consensus inliers from the previously estimated best-fit line using RANSAC and proceed to fit a circle to the remaining outliers using a new RANSAC approach.



(a) Outliers to the RANSAC-fitted line, identified from the dataset.

(b) Separated points classified as outliers after subtracting the consensus inliers from the RANSAC line.

(c) Fitted circle using RANSAC, showing both the best sample circle and the estimated circle.

Figure 3: Circle fitting after subtracting line inliers using RANSAC.

```python
# Define the line parameters (a, b, d) and distance threshold
a, b, d = best_model_line
distance_threshold = 2.5

# Initialize arrays to store inliers and outliers
inliers = []
outliers = []

# Calculate and classify points as inliers or outliers
for x, y in X:
    distance = abs(a * x + b * y - d)
    if distance <= distance_threshold:
        inliers.append([x, y])
    else:
        outliers.append([x, y])

# Convert inliers and outliers to NumPy arrays
inliers = np.array(inliers)
outliers = np.array(outliers)
```

(a) Code for classifying points as inliers or outliers based on their distance to the RANSAC-fitted line.

(b) Functions for fitting a circle through three points and finding inliers based on radial distance.

```python
def RANSAC(X):
    S = 3
    d = 1.96 #gaussian threshold for selecting a
    #correct outlier with p1 = 95% probability.
    e = 0.50
    p = 0.99
    N = int(np.ceil(np.log(1-p)/np.log(1-(1-e)**S)))

    best_fit_circle = None
    best_fit_x = None
    best_inlier_count = 0

    for _ in range(N):
        x = []

        for i in range(5):
            hold = X[np.random.randint(0, len(X)), :]

            if len(x) == 0:
                x.append(hold)
            elif np.array_equal(hold, x[-1]):
                while np.array_equal(hold, x[-1]):
                    hold = X[np.random.randint(0, len(X)), :]

                x.append(hold)
            else:
                x.append(hold)

        a, b, r = circ_through_points(x[0], x[1], x[2])

        if a == None:
            continue

        count, inliers = get_inlier(a, b, r, X, d)

        if count > best_inlier_count:
            best_fit_circle = plt.Circle((a, b), r, color='orange', fill=False, label="Best Sample")
            best_fit_x = x
            best_fit_inliers = inliers
            best_inlier_count = count

    if best_inlier_count < e:
        print("The RANSAC algorithm did not find a suitable model")
        return None, None, None, None

    xc,yc,r,_ = cf.least_squares_circle(best_fit_inliers)

    ransac_circle = plt.Circle((xc, yc), r, color='green', fill=False, label="RANSAC")

    return ransac_circle, best_fit_circle, best_fit_x, best_fit_inliers
```

(c) RANSAC algorithm to find the best-fitting circle by maximizing inliers from the outlier points.

Figure 4: Important Code Snippets for Circle Fitting Using RANSAC.

Impact of Fitting the Circle First: If the circle is fitted first, there's a chance that the randomly selected points could all be on the line, causing the calculated circle to be large and resemble a line. Additionally, fitting a

circle is more complex as it involves more parameters. This could lead to an incorrect fit with a very large radius, making the circle appear similar to a line. However, since RANSAC runs multiple iterations, it might still find the correct fit despite the line points.

3 Homography and Image Warping

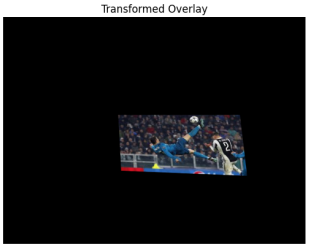
I applied homography to superimpose an overlay image onto a planar surface in two different architectural images.

```
# Create the matrix for homography
matrix_A = np.concatenate((
    np.concatenate((zero_matrix.T, f1, -y1 * f1), axis=1),
    np.concatenate((f1, zero_matrix.T, -x1 * f1), axis=1),
    np.concatenate((zero_matrix.T, f2, -y2 * f2), axis=1),
    np.concatenate((f2, zero_matrix.T, -x2 * f2), axis=1),
    np.concatenate((zero_matrix.T, f3, -y3 * f3), axis=1),
    np.concatenate((f3, zero_matrix.T, -x3 * f3), axis=1),
    np.concatenate((zero_matrix.T, f4, -y4 * f4), axis=1),
    np.concatenate((f4, zero_matrix.T, -x4 * f4), axis=1)
), axis=0, dtype=np.float64)
# Compute homography matrix and warp the overlay image
transformed_flag = cv.warpPerspective(flag_im, H, (w, h))
# Blend the flag with the original image
final = cv.addWeighted(im, 1, transformed_flag, 1, 0)
```

Listing 4: Important Code Snippet for Computing Homography and Blending.

(a) The original architectural image of the Galle Fort.

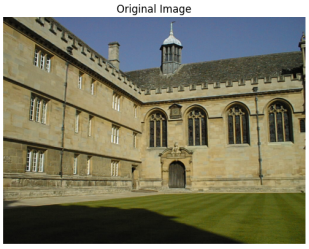
(b) The image of Cristiano Ronaldo performing a bicycle kick used as the overlay.



(c) The overlay image after being warped using the computed homography matrix to fit the plane in the Galle Fort image.

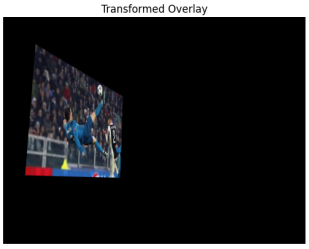
(d) The final result of blending the warped overlay with the Galle Fort image.

Figure 5: Homography and warping of an overlay image onto the Galle Fort.



(a) The original architectural image of Wadham College.

(b) The image of Cristiano Ronaldo performing a bicycle kick used as the overlay.



(c) The overlay image after being warped using the homography matrix to fit the plane in the Wadham College image.

(d) The final result of blending the warped overlay with the Wadham College image.

Figure 6: Homography and warping of an overlay image onto the Galle Fort.

I chose these images because the flat surfaces of the Galle Fort and Wadham College make them ideal for superimposing the overlay. The contrast between these historical landmarks and my favorite goal ever, Cristiano Ronaldo's bicycle kick, creates a visually interesting result.

4 Stitching Graffiti Images Using SIFT, Homography, and RANSAC

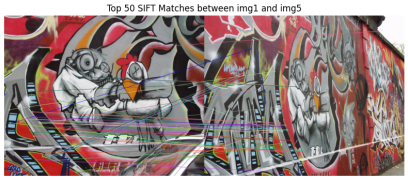


Figure 7: Top 50 SIFT matches between img1.ppm and img5.ppm.

I estimated the homography between consecutive images (img1 to img2, img2 to img3, etc.) by matching keypoints using SIFT. Since there are not enough matching features between img1 and img5, had to chain the homographies across consecutive image pairs to obtain the final homography matrix from img1 to img5.

```python
# RANSAC Homography computation
def RANSAC_homography(points1, points2, iterations=1000, threshold=5):
    max_inliers = 0
    best_H = None
    points = np.hstack((points1, points2))

    for _ in range(iterations):
        np.random.shuffle(points)
        pts1_sample = points[:4, :2]
        pts2_sample = points[:4, 2:]
        H = homography(pts1_sample, pts2_sample)

        # Find inliers
        inliers = [(points1[i], points2[i]) for i in range(len(points1))
     if dist(points1[i], points2[i], H) < threshold]
        if len(inliers) > max_inliers:
            max_inliers = len(inliers)
            best_H = H

    return best_H
```

Listing 5: Important Code Snippet for Homography Estimation Using RANSAC

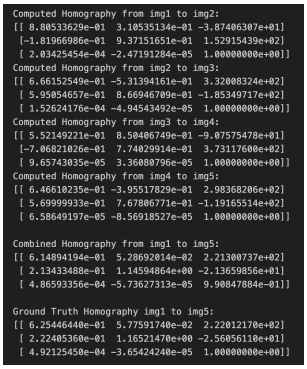
(a) Original image (img1.ppm).

(b) img1 warped.

(a) Original img1 used for stitching.

(b) Original img5 as the base for stitching.

(c) The final stitched image with differences highlighted using a red circle.



(d) Computed homographies between consecutive images, combined homography from img1 to img5, and the ground truth homography matrix for comparison.

Figure 9

A GitHub Link

github.com/AchiraHansindu/EN3160-Image-Processing-and-Machine-Vision/Assignment 2