

EN3160 Assignment 2 - Fitting and Alignment

Name: Kodikara U. S. S. | Index No.: 210293K | Date: 23rd October 2024

Github: <https://github.com/uvinduuu/EN3160-ImageProcessingMachineVision/tree/main/Assignment02>

1 Blob Detection

The algorithm iteratively applies Gaussian blur and the Laplacian operator to the grayscale image for different sigma values, detecting blobs by thresholding the Laplacian's absolute values. Contours are then extracted from the binary blob mask, and circles are fitted to the detected contours.

Range of σ Values: [0.71, 1.01, 1.31, 1.62, 1.92, 2.22, 2.53, 2.83]

The largest detected circle: Center: (184, 253), Radius: 25 pixels, Sigma value: 2.83

```
1 # Set up parameters for scale-space extrema detection
2 min_sigma = 1.0 # Minimum sigma value (smaller values for smaller blobs)
3 max_sigma = 4.0 # Maximum sigma value (larger values for larger blobs)
4 num_sigma = 8 # Number of sigma values to test
5 threshold = 0.375 # Threshold for blob detection
6 circles = [] # Create an empty list to store detected circles
7 for sigma in np.linspace(min_sigma, max_sigma, num_sigma):
8     sigma = sigma / 1.414
9     print(f"Current sigma value: {sigma}")
10    blurred = cv2.GaussianBlur(gray_image, (0, 0), sigma)
11    laplacian = cv2.Laplacian(blurred, cv2.CV_64F)
12    abs_laplacian = np.abs(laplacian)
13    blob_mask = abs_laplacian > threshold * abs_laplacian.max()
14    contours, _ = cv2.findContours(blob_mask.astype(np.uint8), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE
15    )
16    for contour in contours:
17        if len(contour) >= 5:
18            (x, y), radius = cv2.minEnclosingCircle(contour)
19            center = (int(x), int(y))
20            radius = int(radius)
21            circles.append((center, radius, sigma))
```

Listing 1: Blob Detection using LoG and Contour Fitting



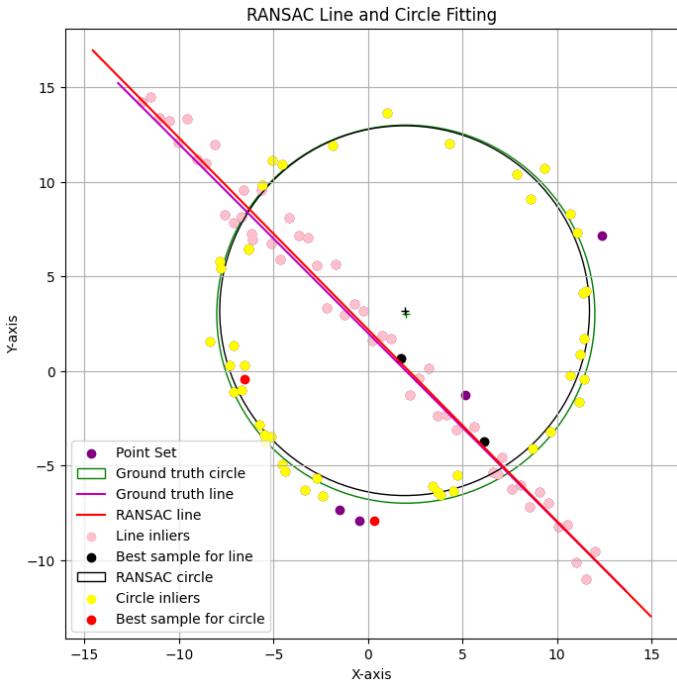
2 Line and circle fitting using RANSAC

- **Best line:** [0.7121, 0.7021, 1.5328]
No. of inliers = 52, Best error = 8.0987

- **Best circle:** [1.7751, 3.1418, 9.9013]
No. of inliers = 45, Best error = 8.0338

If the circle is fitted first,

There is a possibility that the three randomly chosen sample points might lie on the line, resulting in a large circle that resembles a line locally. However, because RANSAC runs for many iterations with varying sample points, it can still accurately fit the circle without needing to remove the line points.



RANSAC Parameters:

- Minimum number of points (s):
 - Line: 2
 - Circle: 3
- Error threshold (t):
 - Line: 1
 - Circle: 1.2
- Consensus size (d):
 - Line: 40
 - Circle: 40

Figuur 1: RANSAC Fitting Results and Parameters

```

1 # Squared error calculation for line and circle
2 def tls_error_line(params, *args):
3     a, b, d = params
4     indices, X = args
5     error = np.sum((a * X[indices, 0] + b * X[indices, 1] - d)**2)
6     return error
7 def tls_error_circle(params, *args):
8     cx, cy, r = params
9     indices, X = args
10    error = np.sum((dist((cx, cy), (X[indices, 0], X[indices, 1])) - r)**2)
11    return error
12
13 def consensus_line(params, thres, X):
14     a, b, d = params
15     errors = np.abs(a * X[:, 0] + b * X[:, 1] - d)
16     return np.where(errors < thres)[0] # Return the indices of inliers
17 def consensus_circle(params, thres, X): # Determine inliers
18     cx, cy, r = params
19     errors = np.abs(dist((cx, cy), (X[:, 0], X[:, 1])) - r) # Radial error
20     return np.where(errors < thres)[0] # Return indices of inliers
21 def constraint(params): # Constraint to normalize line parameters
22     a, b, d = params
23     return (a**2 + b**2)**0.5 - 1 # Should equal zero
24 def least_squares_line_fit(indices, initial, X): # Fit a line using least squares optimization
25     res = minimize(fun=tls_error_line, x0=initial, args=(indices, X), constraints=constraint_dict, tol=1e-6)
26     print(res.x, res.fun) # Print fitted parameters and error
27     return res
28 def least_squares_circ_fit(indices, initial, X): # Fit a circle using least squares optimization
29     res = minimize(fun=tls_error_circle, x0=initial, args=(indices, X), tol=1e-6)
30     print(res.x, res.fun) # Print fitted parameters and error
31     return res
32 # Fitting the line
33 iters = 100 # Number of iterations for fitting
34 min_points = 2 # Minimum points needed to define a line
35 N = X.shape[0] # Total number of points
36 np.random.seed(14) # Set random seed for reproducibility
37 thres = 1. # Error threshold for inlier selection
38 d = 0.4 * N # Minimum number of inliers for a valid fit
39 for i in range(iters):
40     # Randomly select two points to define the line
41     indices = np.random.choice(np.arange(0, N), size=min_points, replace=False)
42     params = line_eq(X[indices[0]], X[indices[1]]) # Calculate line parameters

```

```

43     inliers = consensus_line(params, thres, X) # Get indices of inliers
44     num_inliers = len(inliers) # Get number of inliers from the array
45     if num_inliers >= d: # Check if inlier count meets the threshold
46         res = least_squares_line_fit(inliers, params, X) # Optimize the line fit
47         if res.fun < best_error: # Update best fit if error is lower
48             best_error = res.fun
49             best_model_line = params
50             best_fitted_line = res.x
51             best_line_inliers = inliers
52             best_line_sample_points = indices
53
54 # Fitting the circle, Get remaining points
55 remaining_points = np.setdiff1d(np.arange(N), line_inliers) # More efficient
56 X_rem = X[remaining_points]
57 iters = 100
58 min_points = 3 # Minimum points for circle estimation
59 thres = 1.2 # Error threshold
60 d = 0.4 * N # Minimum inlier count
61 for i in range(iters):
62     indices = np.random.choice(len(X_rem), size=min_points, replace=False)
63     params = circle_eq(*X_rem[indices]) # Calculate circle parameters
64     inliers = consensus_circle(params, thres, X_rem) # Get inliers
65     # Ensure inliers is treated correctly
66     if isinstance(inliers, tuple) and len(inliers) > 0:
67         inlier_indices = inliers[0] # Get inlier indices
68     elif isinstance(inliers, np.ndarray):
69         inlier_indices = inliers
70     else:
71         inlier_indices = np.array([]) # Set to empty array if not valid
72     num_inliers = len(inlier_indices) # Count inliers
73     if num_inliers >= d: # Check inlier count
74         res = least_squares_circ_fit(inlier_indices, params, X_rem) # Optimize circle fit
75         if res is not None and res.fun < best_error: # Check validity of res
76             best_error = res.fun
77             best_model_circle = params
78             best_fitted_circle = res.x
79             best_circ_inliers = inlier_indices
80             best_circ_sample_points = indices

```

Listing 2: Line and Circle fitting using RANSAC

3 Superimposing an image on another

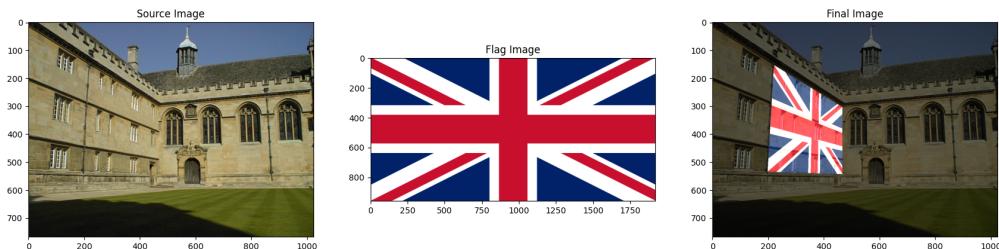
```

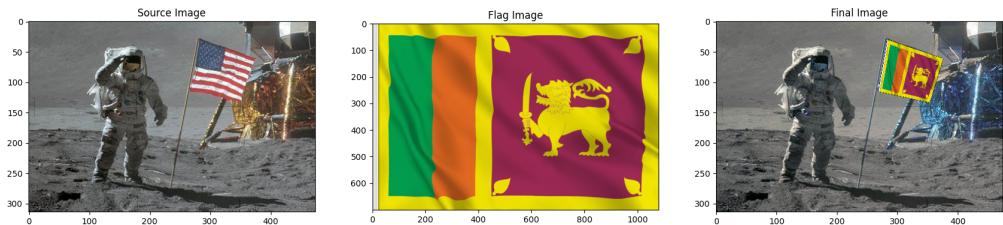
1 h, status = cv.findHomography(p, p_flag) # Calculating homography between image and flag
2 # Warping image of flag
3 warped_img = cv.warpPerspective(image_superimposed, np.linalg.inv(h), (image_background.shape[1],
4 image_background.shape[0]))
blended = cv.addWeighted(image_background, 0.5, warped_img, 0.9, 0.0)

```

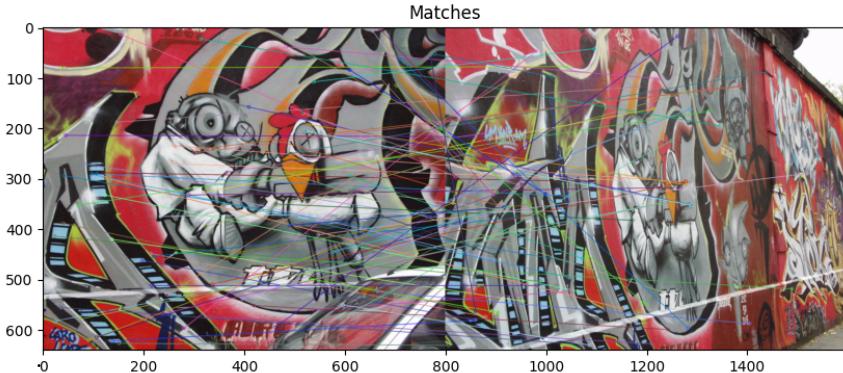
A mouse callback function is used to collect points from mouse clicks on background image. The coordinates of the clicked points are stored in the *p* array, while the corner locations of the flag image are stored in the *pflag* array. Then computes the homography matrix to map the flag's corners to the selected points. The flag image is then warped using this matrix and blended with the background image to create the final output.

For my images I created a mask on the background image based on the selected pixel range, which sets the corresponding area to black. The flag image is then warped using this mask, allowing for accurate placement on the background without blending the two images.





4 Image stitching



Then homography between images was computed using the `calculateHomography` function. Initially, the calculated homography showed significant differences from the provided homography. To improve accuracy, individual homographies for five images were calculated and then multiplied sequentially to obtain the final homography between images one and five. Comparing this final calculated homography with the provided one indicates that they are quite similar.

```

1 def compute_homography(correspondences):
2     matrix_list = []
3     for point_set in correspondences:
4         point1 = np.matrix([point_set.item(0),
5             point_set.item(1), 1])
6         point2 = np.matrix([point_set.item(2),
7             point_set.item(3), 1])
8         row1 = [-point2.item(2) * point1.item(0),
9             -point2.item(2) * point1.item(1),
10            point2.item(2) * point1.item(2), 0, 0,
11            0,
12            point2.item(0) * point1.item(0),
13            point2.item(0) * point1.item(1),
14            point2.item(0) * point1.item(2)]
15         row2 = [0, 0, 0, -point2.item(2) * point1.
16             item(0), -point2.item(2) * point1.item
17             (1), -point2.item(2) * point1.item(2),
18             point2.item(1) * point1.item(0),
19             point2.item(1) * point1.item(1),
20             point2.item(1) * point1.item(2)]
21         matrix_list.append(row1)
22         matrix_list.append(row2)
23     matrix_to_solve = np.matrix(matrix_list)
24     u, s, v = np.linalg.svd(matrix_to_solve)
25     homography_matrix = np.reshape(v[8], (3, 3))
26     homography_matrix = (1 / homography_matrix.
27         item(8)) * homography_matrix
28     return homography_matrix

```

$$\begin{bmatrix} 6.25446 \times 10^{-1} & 5.77591 \times 10^{-2} & 2.22012 \times 10^2 \\ 2.22405 \times 10^{-1} & 1.16521 \times 10^0 & -2.56056 \times 10^1 \\ 4.92125 \times 10^{-4} & -3.65424 \times 10^{-5} & 1.00000 \times 10^0 \end{bmatrix}$$

$$\begin{bmatrix} 7.25696 \times 10^{-1} & -1.21646 \times 10^0 & 8.31404 \times 10^1 \\ 1.35638 \times 10^{-1} & -9.94981 \times 10^0 & 4.31717 \times 10^1 \\ 8.21085 \times 10^{-4} & -4.46336 \times 10^{-3} & 1.00000 \times 10^0 \end{bmatrix}$$



Figuur 2: Homography calculation, matrices comparison and Final stitched output