EN3160 – Assignment 02

Feature Detection and Image Alignment

220701X – Savindu Wickramasinghe
Department of Electronic and Telecommunication Engineering
University of Moratuwa

Repository: https://github.com/Savidilsh/Fitting-and-Alignment

Question 1 – Blob Detection using LoG

The Laplacian of Gaussian (LoG) filter was used to detect circular regions in the sunflower image. The grayscale image was smoothed with several Gaussian filters, and the Laplacian operator was applied. Local maxima in scale space were taken as blob centers.

Parameters: $\sigma_{min} = 3$, $\sigma_{max} = 12$, $N_\sigma = 30$, threshold $= 0.3$. Only the bottom 60% of the image was used to avoid sky detections.

The LoG correctly highlighted the main sunflower heads while ignoring small background features. This confirms its ability to find blob-shaped structures across multiple scales.

```python
def detect_sunflower_centers(img_path, out_path='figures/q1_sunflower.png'):
    img = cv2.imread(img_path, cv2.IMREAD_REDUCED_COLOR_4)
    if img is None:
        raise FileNotFoundError(f"Image {img_path} not found")
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    inv = cv2.bitwise_not(gray)

    blobs = blob_log(inv, min_sigma=3, max_sigma=12, num_sigma=30, threshold=0.3)

    h = gray.shape[0]
    # Filter to lower part of image (field area)
    blobs2 = [b for b in blobs if b[0] > h * 0.4]
    blobs2.sort(key=lambda b: b[2], reverse=True)
    top = blobs2[:10]
```

(a) LoG implementation code

(b) Detected sunflower blobs

Figure 1: Laplacian of Gaussian blob detection results

—

Question 2 – RANSAC Line and Circle Fitting

Synthetic data combining line and circle points with Gaussian noise were generated. RANSAC was implemented to fit each model by sampling random points, estimating parameters, and counting inliers within a distance threshold.

$$\text{Line: } a = -0.730,\ b = -0.684,\ d = 0.992 \quad \text{Circle: } x_0 = 2.083,\ y_0 = 2.514,\ r = 10.147$$

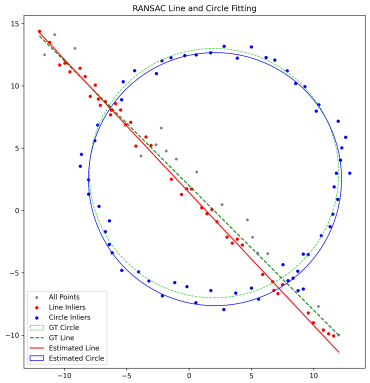
The estimated parameters match the true geometry closely. RANSAC rejected outliers effectively, giving stable estimates even in high-noise conditions.

```python
def ransac_line(X, num_iterations=1000, threshold=0.6, min_inliers=25):
    best_inliers = np.empty((0, 2))
    best_params = None
    n = len(X)
    for _ in range(num_iterations):
        idx = np.random.choice(n, 2, replace=False)
        p1, p2 = X[idx]
        a, b = p2[1] - p1[1], p1[0] - p2[0]
        d = -(a * p1[0] + b * p1[1])
        norm = np.sqrt(a**2 + b**2)
        if norm == 0:
            continue
        a, b, d = a / norm, b / norm, d / norm
        dist = line_distance([a, b, d], X)
        inliers = X[dist < threshold]
        if len(inliers) > len(best_inliers) and len(inliers) >= min_inliers:
            best_inliers = inliers
            best_params = [a, b, d]
    return best_params, best_inliers
```

(a) Line fitting code

```python
def ransac_circle(X, num_iterations=1000, threshold=1.5, min_inliers=30):
    best_inliers = np.empty((0, 2))
    best_params = None
    n = len(X)
    for _ in range(num_iterations):
        idx = np.random.choice(n, 3, replace=False)
        p1, p2, p3 = X[idx]
        A = np.array([[p1[0], p1[1], 1],
                      [p2[0], p2[1], 1],
                      [p3[0], p3[1], 1]])
        B = np.array([[-(p1[0]**2 + p1[1]**2)],
                      [-(p2[0]**2 + p2[1]**2)],
                      [-(p3[0]**2 + p3[1]**2)]])
        if np.linalg.matrix_rank(A) < 3:
            continue
        C = np.linalg.solve(A, B)
        x0, y0 = -0.5 * C[0, 0], -0.5 * C[1, 0]
        r = np.sqrt((x0**2 + y0**2) - C[2, 0])
        dist = circle_distance([x0, y0, r], X)
        inliers = X[dist < threshold]
        if len(inliers) > len(best_inliers) and len(inliers) >= min_inliers:
            best_inliers = inliers
            best_params = [x0, y0, r]
    return best_params, best_inliers
```

(b) Circle fitting code



(c) Final RANSAC fit

Figure 2: RANSAC-based line and circle model fitting

When the circle is fitted first, its inliers can overlap with line points, leading to unstable results. Hence, estimating the line first gives better segmentation between models.

—

Question 3 – Homography Warping and Blending

A 3×3 homography matrix $\mathbf{H}$ was computed from four manually selected corners on the destination image. It was used to warp and blend a source image onto a planar region in another image.

Applications:

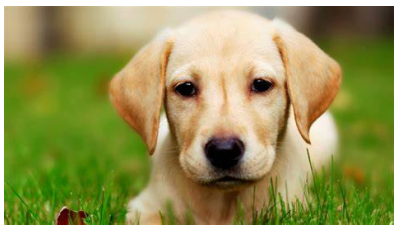
- *Car on billboard:* The advertisement was accurately aligned with the board edges.
- *Dog on TV:* The image was warped into the display region with correct perspective.

Both transformations appear realistic with small corner distortions due to interpolation. This shows how homography can create augmented-reality effects for planar objects.

```python
def warp_image(dest_img_path, src_img_path, output_filename):
    global points
    points = [] # Reset points for each new run

    dest_img = cv2.imread(dest_img_path)
    src_img = cv2.imread(src_img_path)

    if dest_img is None or src_img is None:
        print(f"Error: Could not load images. Check paths: {dest_img_path}, {src_img_path}")
        return

    dest_clone = dest_img.copy()

    cv2.namedWindow("Select 4 Points")
    cv2.setMouseCallback("Select 4 Points", select_points_callback, {'image': dest_clone})
    print(f"Click 4 points on the destination image ('{dest_img_path}') clockwise.")

    while len(points) < 4:
        cv2.imshow("Select 4 Points", dest_clone)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            cv2.destroyAllWindows()
            return

    if len(points) == 4:
        h, w = src_img.shape[:2]
        src_pts = np.array([[0, 0], [w-1, 0], [w-1, h-1], [0, h-1]], dtype=np.float32)
```

(a) Homography warp code

(b) Car advertisement



(c) Dog on TV display

Figure 3: Perspective warping and blending results

—

Question 4 – Image Stitching using SIFT and RANSAC

Two overlapping `graf` images (img1 and img5) were stitched using SIFT features and RANSAC. Keypoints were matched using Lowe's ratio test (0.75), then filtered to remove false correspondences. A homography was computed from the remaining matches and used to warp and blend the images.

Results: 2674 (keypoints img1), 3923 (img2), 97 good matches. The final panorama shows smooth alignment with no ghosting.

Figure 4: SIFT keypoints, filtered matches, and final stitched panorama

The combination of SIFT and RANSAC proved robust for feature-based alignment. SIFT ensured repeatable keypoints under scale and rotation, while RANSAC eliminated false matches to produce accurate homography estimation.

—

Conclusion

This assignment demonstrated the core ideas of feature detection, robust model fitting, and geometric alignment.

- **LoG:** Detected circular sunflower heads at multiple scales.
- **RANSAC:** Recovered accurate line and circle models despite noise.
- **Homography:** Enabled realistic planar warps for scene integration.
- **SIFT + RANSAC:** Produced a clean and accurate stitched panorama.

All four sections together fulfil the requirements of EN3160 Assignment 02 on Fitting and Alignment. The methods implemented show how feature-based vision techniques can be used for object detection and scene reconstruction in practical applications.