EN3160 Image Processing and Machine Vision

Assignment 2: Fitting and Alignment

Jayaweera M. V. L. M. (220285X)

Link to GitHub repository – Click Here.

1 Question 1: Blob Detection using Laplacian of Gaussian (LoG)

1.1 Objective

The goal of this task is to detect blobs (circular regions) of varying sizes in the given sunflower field image. The detection was performed using the **Laplacian of Gaussian (LoG)** operator and **scale-space extrema detection**.

1.2 Methodology

To detect blobs of different sizes, the image was convolved with a set of LoG kernels corresponding to multiple scales σ. The filter response was analyzed across both spatial and scale dimensions to identify points of local maxima.

- **Scale-space construction:** For each σ in the range $[1, 35]$, the image was filtered with a scale-normalized LoG kernel ($\sigma^2 \nabla^2 G$), and the responses were stacked to form a 3D scale-space $L(x, y, \sigma)$.
- **Kernel design:**

```
for sigma in sigma_values:
    ksize = int(2 * np.ceil(3 * sigma) + 1) # kernel size
    X, Y = np.meshgrid(np.arange(-ksize//2 + 1, ksize//2 + 1), np.arange(-ksize//2 + 1,
    ksize//2 + 1))
    LOG = ((X**2 + Y**2) / (2 * sigma**2) - 1) * np.exp(-(X**2 + Y**2) / (2 * sigma**2))
    / (np.pi * sigma**4)  # Normalized LoG
    LOG = LOG * (sigma**2) # Scale normalization
    resp = np.square(cv.filter2D(gray_image, -1, LOG))
    responses.append(resp)
```

- **3D Non-Maximum Suppression (NMS):** The local maxima were detected across space and scale using a 3D NMS procedure. A pixel (x, y, σ) was considered a blob center if it was a local maximum in a 3×3×3 neighborhood.
- **Thresholding:** Only strong responses were retained using the 98^{th} percentile of the scale-space values.

```
candidates = is_local_max_2d & is_scale_peak
thr = np.percentile(scale_space[candidates], 98) if np.any(candidates) else np.percentile
    (scale_space, 99)
strong = candidates & (scale_space >= thr)
```

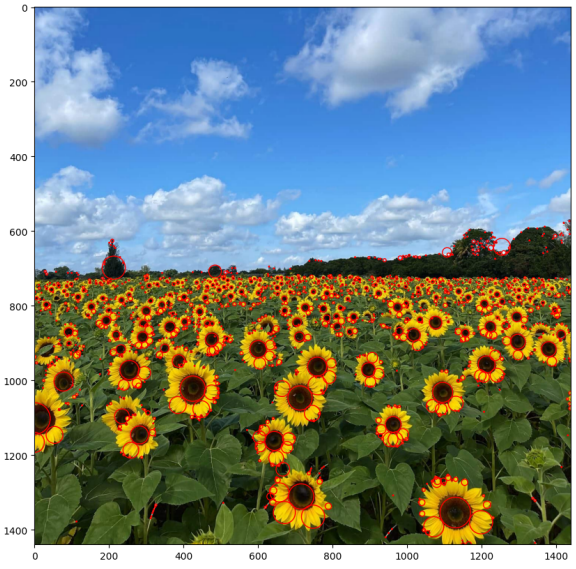


Figure 1: Blob detection

Detected blobs correspond to circular regions of varying sizes in the image (e.g., sunflower heads). Each blob's radius was determined by $r = \sqrt{2}\sigma$, which relates the scale of maximum response to the blob size. The largest detected blobs and their parameters were reported as required.

Important

- Scales tested: $\sigma \in \{1, 2, \ldots, 35\}$ (integers).
- Largest blob: center (0.0, 1102.0), radius 42.4 px, $\sigma = 30.0$.

2 Question 2: Line and Circle Fitting using RANSAC

2.1 Methodology

The process was performed in two stages:

- **Line fitting:** Random pairs of points were sampled to define hypotheses of a line in the form $ax + by = d$, where $\sqrt{a^2 + b^2} = 1$. For each hypothesis, the perpendicular distances $|ax_i + by_i - d|$ were computed. Points with distances below a threshold were counted as inliers. The model with the highest inlier count was selected.
- **Circle fitting:** After removing the line inliers, RANSAC was applied again on the remaining points. Triplets of points were used to estimate the circle parameters by solving the linear system:

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \end{bmatrix} = - \begin{bmatrix} x_1^2 + y_1^2 \\ x_2^2 + y_2^2 \\ x_3^2 + y_3^2 \end{bmatrix},$$

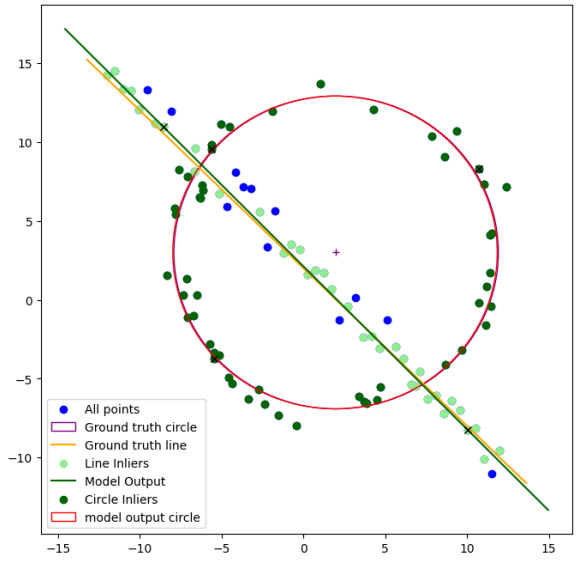
from which the circle center and radius were obtained as $x_0 = -A/2$, $y_0 = -B/2$, and $r = \sqrt{x_0^2 + y_0^2 - C}$.

```python
def ransac_line(X, n_iters=1000, threshold=0.6, min_inliers=35):
    best_a, best_b, best_d = 0, 0, 0
    best_inliers = []
    n_points = X.shape[0]
    for _ in range(n_iters):
        idx = np.random.choice(n_points, 2, replace=False)
        p1, p2 = X[idx]
        a = p2[1] - p1[1]  #parameters
        b = p1[0] - p2[0]
        d = p1[0]*p2[1] - p2[0]*p1[1]
        norm = np.sqrt(a**2 + b**2)   #normalizing
        a, b, d = a/norm, b/norm, d/norm
        distances = calc_tls_line(a, b, d, X)
        inliers = np.where(distances < threshold)[0]
        if len(inliers) > len(best_inliers) and len(inliers) >= min_inliers:
            best_inliers = inliers
            best_a, best_b, best_d = a, b, d
            choosen_points_line = [p1, p2]
    return best_a, best_b, best_d, best_inliers, choosen_points_line
```

Listing 1: Function for RANSAC line fitting

```python
def ransac_circle(X, n_iters=1000, threshold=1.5, min_inliers=45):
    best_params = [0, 0, 0]
    best_inliers = []
    n_points = X.shape[0]
    for _ in range(n_iters):
        idx = np.random.choice(n_points, 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)]])
        try :
            C = linalg.solve(A, B)
        except linalg.LinAlgError:
            continue
        C = C.flatten()
        x0, y0 = -C[0]/2, -C[1]/2
        r = np.sqrt((C[0]**2 + C[1]**2)/4 - C[2])
        distances = distance_error_circle((x0, y0, r), X)
        inliers = np.where(distances < threshold)[0]
        if len(inliers) > len(best_inliers) and len(inliers) >= min_inliers:
            best_inliers = inliers
            best_params = [x0, y0, r]
            choosen_points_circle = [p1, p2, p3]
    return best_params, best_inliers, choosen_points_circle
```

Listing 2: Function for RANSAC Circle fitting



Results The algorithm successfully identified the two geometric structures in the mixed dataset. The estimated parameters are summarized below:

- **Line:** $-0.718x - 0.696y = -1.475$ (equivalently $y = -1.033x + 2.121$) **Inliers:** 36 points
- **Circle:** center $= (1.933, 2.999)$, radius $= 9.913$ **Inliers:** 52 points

(d) If the circle is fitted first, RANSAC is likely to fail or give inaccurate results. Many random 3-point samples will come from the line portion of the data, which are nearly collinear and lead to unstable or extremely large-radius circles that incorrectly cover both the line and circle points. This causes RANSAC to select a false circle model with a large consensus, removing many true circle points as outliers. As a result, the remaining points will not form a clean line set, and the later line fitting step will perform poorly. Therefore, it is better to fit the line first and then the circle, since the line's smaller minimal sample (2 points) and well-defined perpendicular error metric make it more stable and easier to separate before estimating the circle.

3 Question 3: Image Overlay using Homography

3.1 Methodology

Four corner points were selected on a planar region of the background image, and the corresponding corners of the overlay image were defined. The OpenCV function `cv2.findHomography()` was used to compute the homography matrix H that maps the flag's coordinates to the wall region. The flag image was then warped with `cv2.warpPerspective()` and blended onto the background using a transparency factor α. Following is the code I have used.

```
# Compute Homography and warp the overlay image
H, _ = cv2.findHomography(src_pts, dst_pts)
warped_flag = cv2.warpPerspective(overlay, H, (background.shape[1], background.shape[0]))
# Binary mask of the quad
mask = np.zeros(background.shape[:2], dtype=np.uint8)
cv2.fillConvexPoly(mask, dst_pts.astype(int), 255)
# Convert to float for proper blending
bg_f   = background.astype(np.float32)
flag_f = warped_flag.astype(np.float32)
alp    = 0.5  # 0=transparent, 1=opaque
# Build per-pixel alpha in [0,1] but only inside the quad
a = (mask.astype(np.float32) / 255.0) * alp
# Broadcast alpha to 3 channels
a3 = cv2.merge([a, a, a])
# Blend INSIDE the mask with original background pixels
result_f = bg_f * (1.0 - a3) + flag_f * a3
# Keep background outside the mask exactly unchanged
result = result_f.astype(np.uint8)
```

Figure 2: Overlaying the flag image onto the building image using Homography

Figure 3: Overlaying the logo onto the car image using Homography

4 Question 4: Image Stitching using SIFT and RANSAC Homography

I have used following codes for the relevant tasks.

```
sift = cv2.SIFT_create(nOctaveLayers = 3,contrastThreshold = 0.09, edgeThreshold = 25,
    sigma = 1)
kpsA, desA = sift.detectAndCompute(imgA, None)
kpsB, desB = sift.detectAndCompute(imgB, None)
bf = cv2.BFMatcher()
knn = bf.knnMatch(desA, desB, k=2)
ratio = 0.75   # Lowe's ratio =0.75
good = []
for m, n in knn:
    if m.distance < ratio * n.distance:
        good.append(m)
```

Listing 3: SIFT feature matching.

```
def find_best_homography(good_matches, keypoints1, keypoints5):
    .....
    for i in range(iters):
        chosen_matches = np.random.choice(good_matches, num_points, replace = False)
        src_points = []
        dst_points = []
        for match in chosen_matches:
            src_points.append(np.array(keypoints1[match.queryIdx].pt))
            dst_points.append(np.array(keypoints5[match.trainIdx].pt))
        src_points = np.array(src_points)
        dst_points = np.array(dst_points)
        tform = transform.estimate_transform('projective', src_points, dst_points)
        inliers = get_inliers(src_full, dst_full, tform, thres)
        #print(f'Iteration {i}: No. of inliers = {len(inliers)}')
        if len(inliers) > best_inlier_count:
            best_inlier_count = len(inliers)
            best_homography = tform
            best_inliers = inliers
    .........
```

Listing 4: RANSAC-based homography estimation and stitching

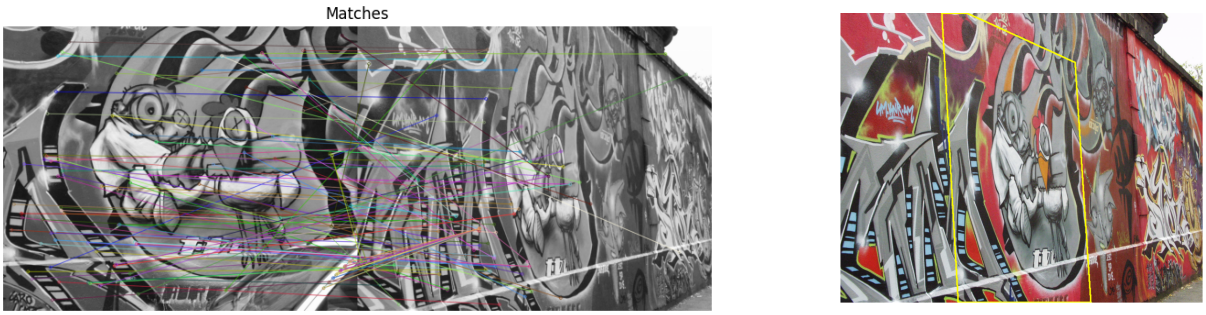


Figure 4: Homography Transformation and stitched image