EN3160 Assignment 2 : 220400A

Question 1

Results and Discussion

(a) Input and Result

Figure shows the output of the blob detection process. The dark central regions of the sunflowers give the strongest blob responses, as they closely resemble the LoG kernel structure. By scanning through radius values from 1 to 10, corresponding to σ values of:

$$0.71,\ 1.41,\ 2.12,\ 2.83,\ 3.54,\ 4.24,\ 4.95,\ 5.66,\ 6.36,\ 7.07$$

we can observe how different Gaussian scales respond to flowers of varying sizes. The largest detected blob was reported at a radius of $r = 10$, indicating the largest sunflower head detected in the image. A blob of radius r approximately corresponds to:

$$r = \sqrt{2}\,\sigma$$

Python Implementation

Listing 1: Main blob detection loop using Laplacian of Gaussian.

```
coordinates = []  # Store blob coordinates
threshold = 0.1   # Threshold for maxima detection

for r in range(1, 11):
    sigma = r / 1.414           #    = r /  2
    hw = round(3 * sigma)       # Half width of the kernel
    X, Y = np.meshgrid(np.arange(-hw, hw + 1),
                       np.arange(-hw, hw + 1))
    # Laplacian of Gaussian (LoG) kernel
    LOG = ((X**2 + Y**2) / (2 * sigma**2) - 1) \
          * np.exp(-(X**2 + Y**2) / (2 * sigma**2)) \
          / (np.pi * sigma**4)
    LOG = sigma**2 * LOG        # Scale normalization
    # Apply LoG filter and square response
```

```
    img1_log = np.square(cv.filter2D(gray, -1, LOG))
    (H, W) = img1_log.shape
    k = 1  # Neighborhood half-size for local maxima
    # --- Local maxima detection ---
    for i in range(k, H - k):
        for j in range(k, W - k):
            region = img1_log[i - k:i + k + 1,
                              j - k:j + k + 1]
            if np.max(region) >= threshold:
                x, y = np.unravel_index(region.argmax(),
                                        region.shape)
                coordinates.append((i + x - k, j + y - k))
# Remove duplicates
coordinates = set(coordinates)
```

Question 2

The RANSAC algorithm was applied to the noisy point set consisting of a mixture of line and circle points. In the first stage, the best-fit line was estimated under the constraint $\|[a, b]^\top\| = 1$, using the perpendicular error measure as the distance metric. After removing its inliers, the remaining points were used to estimate the circle model based on radial error.

The final estimated parameters were:

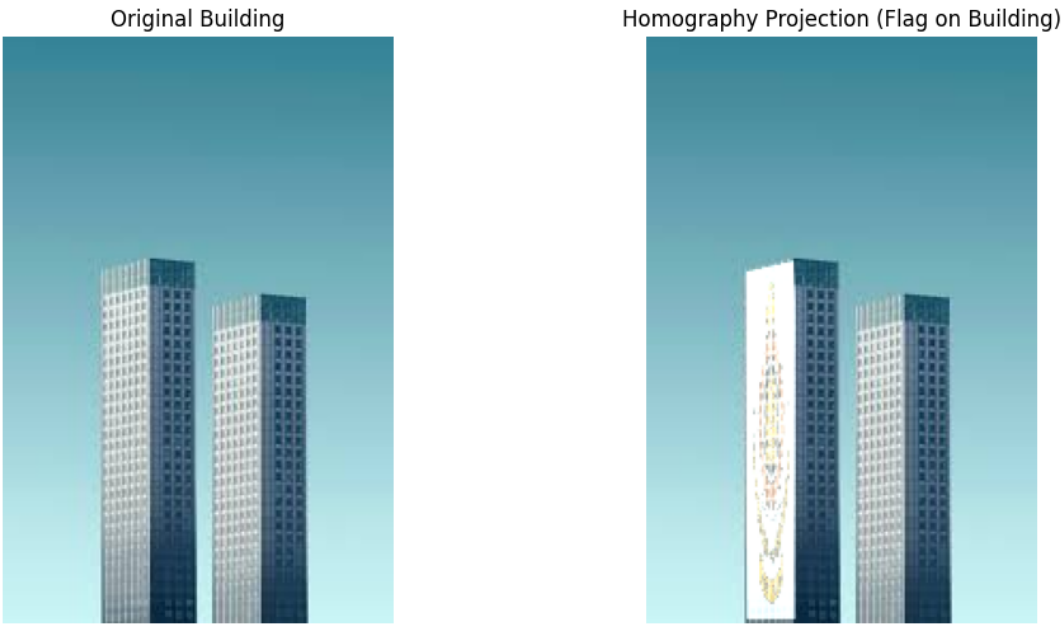
$$\text{Line: } a = 0.706,\ b = 0.708,\ d = -1.692$$

$$\text{Circle: } (x_0, y_0) = (1.86,\ 2.93), \quad r = 9.57$$

These values closely match the ground-truth line ($y = -x + 2$) and circle (center $(2, 3)$, radius 10). Fitting the line first ensures that the dominant linear structure is removed before fitting the circular model, leading to accurate parameter recovery for both shapes. If the circle were fitted first, the linear points would significantly bias the radius estimation, resulting in a poor fit.

Question 3

In this task, a homography was computed to project a UOM flag image onto a planar surface of building image. Four corner points on the planar region of the building image were selected manually using mouse clicks. The corresponding four points of the flag image were defined as the image corners. Using these point correspondences, the homography matrix H was estimated with the `cv2.findHomography()` function, and the flag was warped using `cv2.warpPerspective()` to align with the selected plane. The warped flag was then blended with the building image using a weighted sum to achieve a realistic overlay.



(a) Input and Result

Question 4

Feature detection and matching. Scale-Invariant Feature Transform (SIFT) features were extracted from both images and matched using the inbuilt `BFMatcher()`

Figure 3: Best SIFT matches found between `img1.ppm` and `img5.ppm` using the Brute Force matcher.

Image warping and stitching. The homography obtained from RANSAC was then used to warp `img1.ppm` onto the reference image `img5.ppm`. Finally, the two images were blended to form a seamless panorama.

Listing 2: RANSAC loop for homography estimation.

```
# --- RANSAC parameters ---
num_points = 4
thres = 4
iters = 200
best_homography = None
best_inlier_count = 0
best_inliers = None

# --- RANSAC loop ---
for i in range(iters):
    chosen_matches = np.random.choice(len(good_matches),
       num_points, replace=False)
    src_points = src_full[chosen_matches]
    dst_points = dst_full[chosen_matches]
    H = compute_homography(src_points, dst_points)

    inliers = get_inliers(src_full, dst_full, H, thres)
    if len(inliers) > best_inlier_count:
        best_inlier_count = len(inliers)
        best_homography = H
        best_inliers = inliers
```

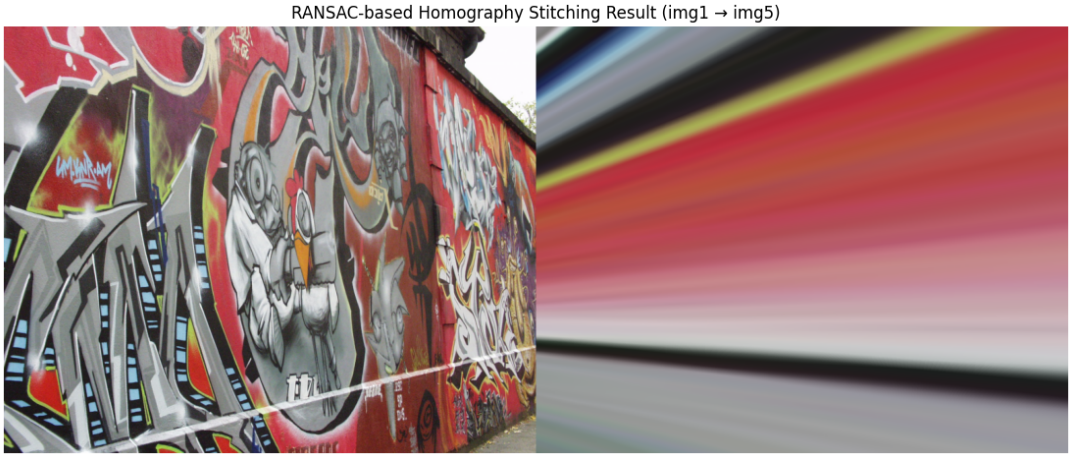


Figure 4: Homography transformation and image stitching results for the Graffiti dataset.

GitHub Repository: Vision Assignment 2