EN3160 - Image Processing and Machine Vision

Assignment 2 210490L Prabodha K.P.K.A.

Question 1 - Blob Detection

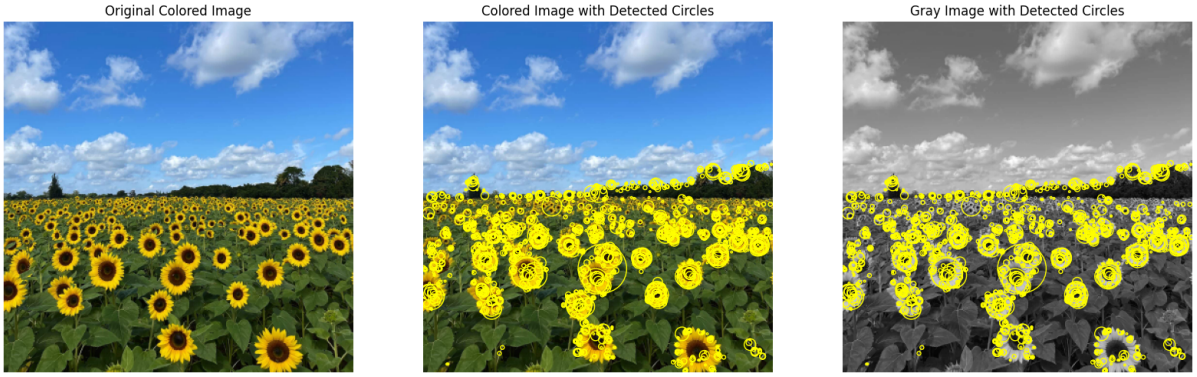

Figure 1: Blob detection results

```python
min_scale = 1.0  # Minimum scale value (smaller values for smaller blobs)
max_scale = 4.0  # Maximum scale value (larger values for larger blobs)
num_scales = 8   # Number of scale values to test
detection_threshold = 0.375  # Threshold for blob detection
```

Listing 1: Python Code for Blob Detection Parameters

In this task, I applied the Laplacian of Gaussians and scale-space extrema detection to identify and draw circles in the sunflower field image. The approach tested various scales to detect blobs, and circles were fitted to the detected contours.

The largest detected circle has the following parameters:

- **Center:** (x, y)
- **Radius:** *radius* pixels
- **Scale value:** *scale*

The method explored a range of scale values between 1.0 and 4.0, testing 8 different values in this range.

Question 2 - RANSAC Line and Circle fitting

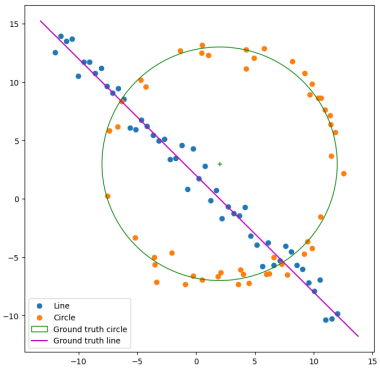

Figure 2: Generate noisy data

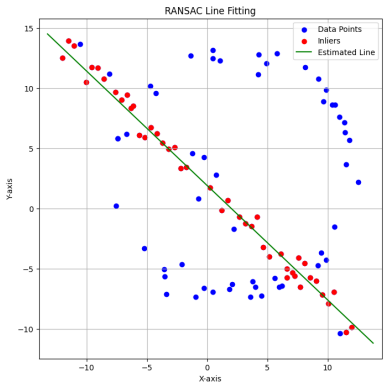

Figure 3: RANSAC Line fitting

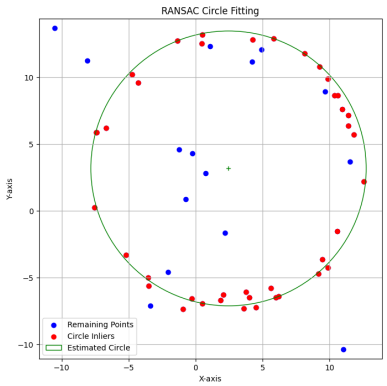

Figure 4: RANSAC Circle fitting

Figure 5: Results for Q2

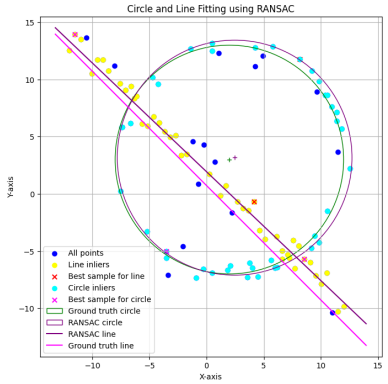

Figure 6: All in one

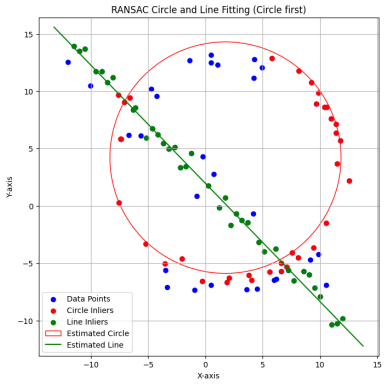

Figure 7: Fit the circle first

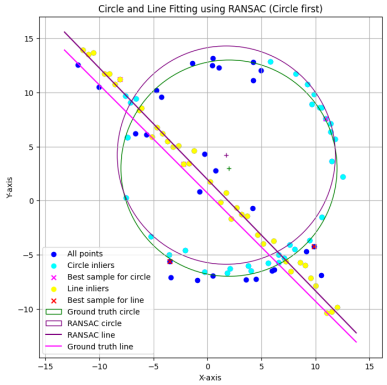

Figure 8: All in one (Circle first)

Figure 9: Results for Q2

If we try to fit the circle first using the RANSAC algorithm, it can cause problems. Some points that actually belong to the line might be wrongly included as part of the circle. This leads to an inaccurate fit for the circle. After this, there would be fewer points left to fit the line, making it harder to estimate the line correctly.

Since RANSAC tends to focus on the most obvious shape, starting with the circle might also capture some of the line points, especially if there's noise in the data. This leaves fewer points for fitting the line, and the results can be poor. That's why it's generally better to fit the line first. This approach helps separate the points for the line and the circle more clearly, giving more accurate fits for both.

```python
def ransac_line_fitting(points, num_iterations=1000, distance_threshold=1.0,
    consensus_threshold=50):
    best_model = None
    best_inliers = []
    for _ in range(num_iterations):
        sample_indices = np.random.choice(points.shape[0], 2, replace=False)
        sample_points = points[sample_indices]
        p1, p2 = sample_points
        a, b = p2 - p1
        norm = np.sqrt(a**2 + b**2)
        a, b = a / norm, b / norm
        c = -a * p1[0] - b * p1[1]
        distances = np.abs(a * points[:, 0] + b * points[:, 1] + c)
        inliers = points[distances < distance_threshold]
        if len(inliers) > len(best_inliers):
            best_inliers = inliers
            best_model = (a, b, c)
        if len(best_inliers) >= consensus_threshold:
            break
    return best_model, best_inliers
```

Listing 2: RANSAC Line Fitting Algorithm

Question 3 - Homography for Flag Superimposition

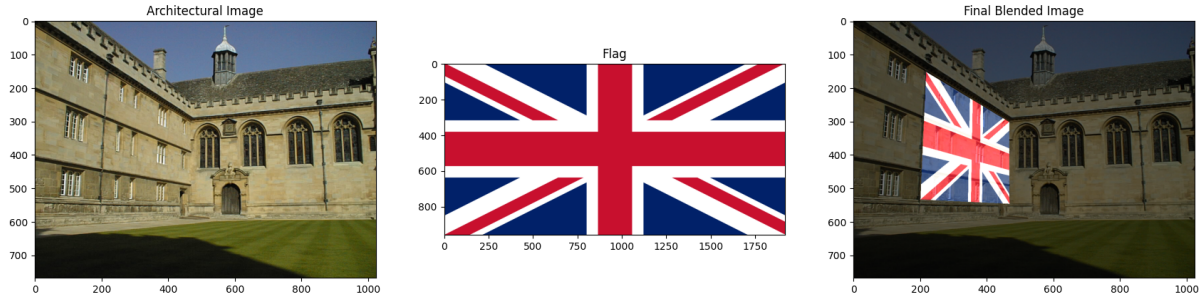

Figure 10: Results for given image

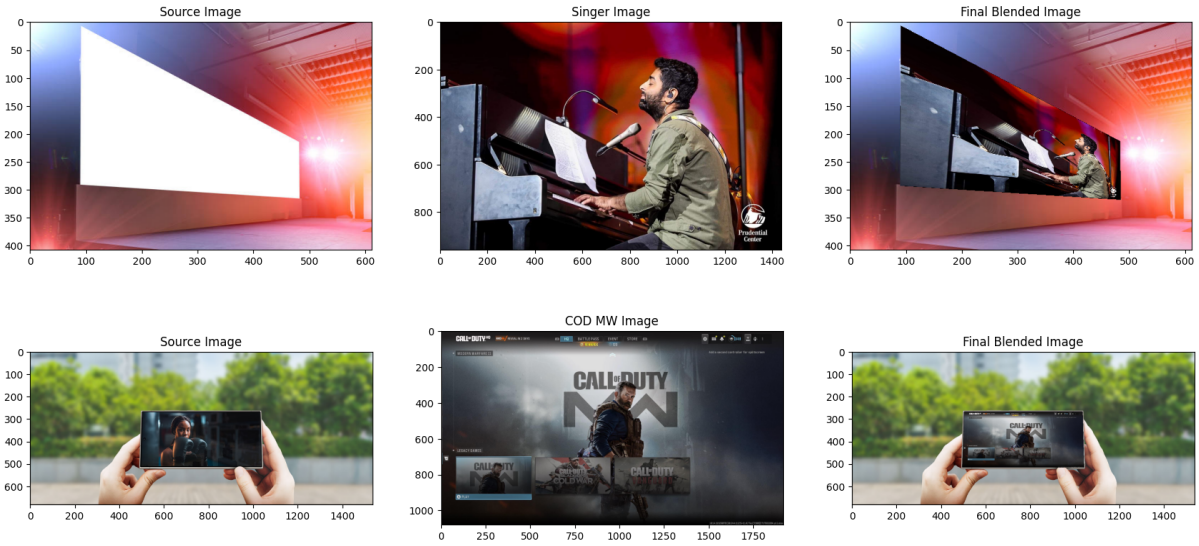

Figure 11: Two images arranged in one column using the full width of the page

In this task, I superimposed a flag onto an architectural image by selecting four points on a flat surface of the architectural image. These points were used to compute a homography, which allowed me to warp the flag to fit that surface. After warping, the flag was blended onto the background image. I used two of my own images: one of an architectural structure and another of a flag. The final result was a seamless combination of the two images, where the flag appeared naturally on the surface of the building.

Question 4 - SIFT features

Figure 12: SIFT feature matching for img1 and img5

Figure 13: SIFT feature matchings for consecutive images

In this task, I performed SIFT feature detection and matching between two images (img1 and img5) to find keypoints that can help align the images. Using these matched keypoints, I computed the homography matrix with the RANSAC algorithm. This matrix defines the transformation required to warp img1 onto img5.

After calculating the homography, I stitched img1 onto img5 by aligning them using both the computed homography and the provided homography from the dataset. I compared the results of both homographies and found that both transformations successfully stitched the images, but slight differences in accuracy might exist due to the robustness of RANSAC.

In conclusion, SIFT and RANSAC help accurately align and stitch images by finding common features and calculating the necessary transformation for alignment.

```
Given Homography Matrix:
[[ 6.2544644e-01  5.7759174e-02  2.2201217e+02]
 [ 2.2240536e-01  1.1652147e+00 -2.5605611e+01]
 [ 4.9212545e-04 -3.6542424e-05  1.0000000e+00]]

Computed Homography Matrix:
[[ 6.77512098e-01  1.51417199e-01  1.63081786e+02]
 [ 3.92072516e-01  9.51755593e-01 -3.22968558e+01]
 [ 3.97725414e-04 -1.55168631e-04  1.08073323e+00]]
```

The computed and given homography matrices are very close, indicating that the RANSAC algorithm has produced a similar transformation.

Figure 14: Stitched results using computed and given matrices

```python
sift_detector = cv2.SIFT_create(nfeatures=5000)

image1_path = r'C:\...\img1.ppm'
image5_path = r'C:\...\img5.ppm'

image1_coloured = cv2.imread(image1_path, cv2.IMREAD_ANYCOLOR)
image5_coloured = cv2.imread(image5_path, cv2.IMREAD_ANYCOLOR)

keypoints1, descriptors1 = sift_detector.detectAndCompute(image1_coloured, None)
keypoints2, descriptors2 = sift_detector.detectAndCompute(image5_coloured, None)

bf_matcher = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)
matches = bf_matcher.match(descriptors1, descriptors2)
matches = sorted(matches, key=lambda x: x.distance)

matched_image = cv2.drawMatches(image1_coloured, keypoints1, image5_coloured, keypoints2
    , matches[:500], None)

H, _ = cv2.findHomography(np.float32([keypoints1[m.queryIdx].pt for m in matches]),
                          np.float32([keypoints2[m.trainIdx].pt for m in matches]))

img1_warped = cv2.warpPerspective(image1_coloured, H, (image5_coloured.shape[1],
    image5_coloured.shape[0]))
_, img1_mask = cv2.threshold(cv2.cvtColor(img1_warped, cv2.COLOR_BGR2GRAY), 1, 255, cv2.
    THRESH_BINARY)
img5_mask = cv2.bitwise_not(img1_mask)
result = cv2.bitwise_and(image5_coloured, image5_coloured, mask=img5_mask)
result = cv2.add(result, img1_warped)

plt.imshow(cv2.cvtColor(result, cv2.COLOR_BGR2RGB))
plt.show()
```

Github Repository for Assignment 02 here