EN3160 Image Processing and Machine Vision

Assignment 2 on Fitting and Alignment

Name:	Warren Jayakumar
Module:	EN3160
Date of Submission:	2024.10.23
GitHub Repository:	Warren-SJ/Image-Processing-Exercises

1. The Laplacian of Gaussian was calculated using the following code:

```
def LoG(sigma):
    hw = round(3*sigma)
    X = np.arange(-hw, hw + 1, 1)
    Y = np.arange(-hw, hw + 1, 1)
    X, Y = np.meshgrid(X, Y)
    log = ((X**2 + Y**2)/(2*sigma**2) - 1) *
    np.exp(-(X**2 + Y**2)/(2*sigma**2)) / (np.pi * sigma**4)
    return log
```

Detecting the maximum was done as:

```
def detect_max(img_log, threshold):
    coordinates = []
    (h, w) = img_log.shape
    k = 1
    for i in range(k, h-k):
        for j in range(k, w-k):
            slice_img = img_log[i-k:i+k+1, j-k:j+k+1]
            result = np.max(slice_img)
            if result >= threshold:
                x, y = np.unravel_index(slice_img.argmax(), slice_img.shape)
                coordinates.append((i+x-k, j+y-k))
    return set(coordinates)
```

The overall result was plotted. This is seen in figure 1. By trial and error, a threshold of

Figure 1: Detected Blobs

0.09 was used as it seemed to detect most of the flowers without false positives.

2. (a) a The error was selected to be 0.4 for line fitting. This value was selected based on trial and error.2 points are required to estimate a line and the number of points in the consensus was 40 which is 80% of the total data. The total least square error and function to estimate how well a line fits were calculated as follows:

```
def line_tls(x, indices,X):
    a, b, d = x[0], x[1], x[2]
    return np.sum(np.square(a*X[indices,0] + b*X[indices,1] - d))
def consensus_line(X, x, t):
    a, b, d = x[0], x[1], x[2]
    error = np.absolute(a*X[:,0] + b*X[:,1] - d)
    return error < t

while iteration < max_iterations:
    indices = np.random.randint(0, N, s)
```

```
        x0 = np.array([1, 1, 0])
        res = minimize(fun = line_tls, args = (indices,X_),
        x0 = x0, tol= 1e-6, constraints=cons, options={'disp': True})
        inliers_line=consensus_line(X_,res.x,t)
        if inliers_line.sum() > d:
            x0 = res.x
            res = minimize(fun = line_tls, args = (inliers_line, X_),
            x0 = x0, tol= 1e-6, constraints=cons, options={'disp': True})
            print(res.x, res.fun)
            if res.fun < best_error:
                best_model_line = res.x
                best_eror = res.fun
                best_sample_line = X_[indices,:]
                res_only_with_sample = x0
                best_inliers_line = inliers_line
        iteration += 1
```

It initially considers a random set of points to estimate the line and then iteratively checks if the line is the best line possible. If a better line is found, it is updated. 100 iterations were used.

(b) In order to fit the circle some functions had to be changed. Additionally, only the remaining points were considered.

```
def dist_circle(params, X, indices):
    xc, yc, r = params
    return np.sum(np.square(np.sqrt((X[indices, 0] - xc)**2 +
    (X[indices, 1] - yc)**2) - r))
def consensus_circle(X, params, t):
    xc, yc, r = params
    error = np.abs(np.sqrt((X[:, 0] - xc)**2 + (X[:, 1] - yc)**2) - r)
    return error < t
```

The RANSAC loop remained the same.

(c) This is shown in figure 2

(d) If the circle is fitted first, there is a chance that some points of the line may be considered for the circle. This might lead to the line not being fitted properly. However, the RANSAC loop might avoid this issue by appropriately fitting the ciccle

3. Image bleeding was done as follows:

```
points = []
def select_points(event, x, y, flags, param):
    global points
    if event == cv2.EVENT_LBUTTONDOWN and len(points) < 4:
        points.append((x, y))
        cv2.circle(arch_image, (x, y), 5, (0, 255, 0), -1)
        cv2.imshow("Select 4 Points", arch_image)
arch_image_rgb = cv2.cvtColor(arch_image, cv2.COLOR_BGR2RGB)
points = plt.ginput(4)
height, width, _ = flag_image.shape
src_points = np.array([[0, 0], [width - 1, 0], [width - 1, height - 1],
[0, height - 1]], dtype=np.float32)
```

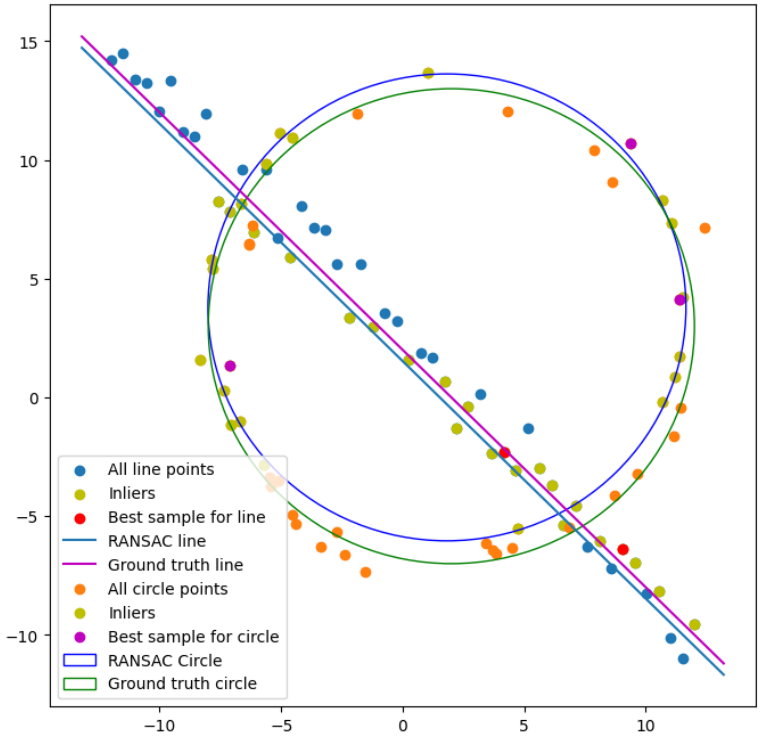

Figure 2: RANSAC line and circle

```
dst_points = np.array(points, dtype=np.float32)
homography_matrix, _ = cv2.findHomography(src_points, dst_points)
warped_flag = cv2.warpPerspective(flag_image, homography_matrix,
(arch_image.shape[1], arch_image.shape[0]))
gray_warped_flag = cv2.cvtColor(warped_flag, cv2.COLOR_BGR2GRAY)
_, mask = cv2.threshold(gray_warped_flag, 1, 255, cv2.THRESH_BINARY)
alpha = 0.5
warped_flag = cv2.addWeighted(warped_flag, alpha, arch_image, 1 - alpha, 0)
mask_inv = cv2.bitwise_not(mask)
arch_image_bg = cv2.bitwise_and(arch_image, arch_image, mask=mask_inv)
flag_fg = cv2.bitwise_and(warped_flag, warped_flag, mask=mask)
result_image = cv2.add(arch_image_bg, flag_fg)
```

A transparency of 0.5 was added to make it feel more natural. Result is shown in figure 3(a). Other figures were also tested as shown in 3(b) where the flag has been placed on the side of the train. It should be noted that square planar images give the best results for such image bleeding and that is why rectangular flags were chosen.

(a) Flag on wall (b) Shinkansen with flag

Figure 3: Examples for image bleeding

4. (a) The SIFT features were computed using the *SIFT_create()* and *detectAndCompute()* methods. Result is shown in figure 4.

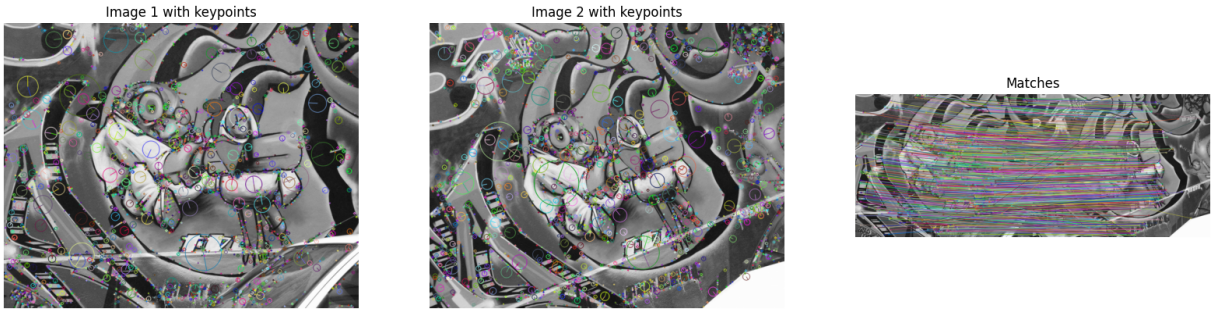

Figure 4: SIFT matching

(b) Using custom RANSAC code, the tomography matrix was found. Since there were insufficient matches from *img1.ppm* to *img5.ppm*, homography was computed by computing between *img1.ppm & img2.ppm*, *img2.ppm & img3.ppm*, *img3.ppm & img4.ppm* and *img4.ppm & img5.ppm* and multiplying them together. The obtained and actual homographies respectively were:

$$\begin{bmatrix} 6.120e-01 & 1.371e-01 & 1.537e+02 \\ 3.487e-01 & 8.803e-01 & -2.720e+01 \\ 3.445e-04 & -1.427e-04 & 1.000e+00 \end{bmatrix} \begin{bmatrix} 6.254e-01 & 5.775e-02 & 2.220e+02 \\ 2.224e-01 & 1.165e+00 & -2.560e+01 \\ 4.921e-04 & -3.654e-05 & 1.000e+00 \end{bmatrix}$$

(c) Homography to stitch the 2 images was obtained as explained above. The result is shown in figure 5.

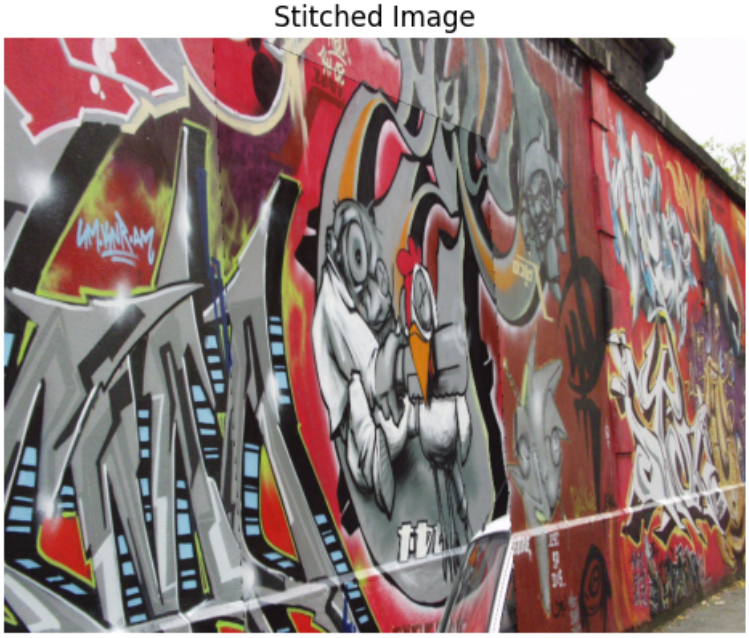

Figure 5: Stiched results