DEPARTMENT OF ELECTRONIC AND TELECOMMUNICATION
UNIVERSITY OF MORATUWA

EN3160 : Image processing and Machine Vision : Fitting and Alignment

200686J : Vishagar A.

October 4, 2023

1 Question 01

In this question we are supposed to detect blobs in a given image. Blob detection is used to identify areas or objects that share characteristics like color, intensity, or texture. They distinguish out from their surroundings due to their homogenous characteristics.

The Laplacian of Gaussians (LoG), which uses a two-step procedure to improve features and find edges. The image is first smoothed using a Gaussian filter to reduces nois and highlights important structures. The Laplacian operator will draw attention to the areas where intensity changes have place. By changing the valueis of σ and threshold values we can detect blobs accurately. And generally blobs are the superpositioned ripple gained from LoG.

And the Laplacian of Gaussian (LoG) is defined as follows.

$$\nabla^2 G(x,y) = \frac{1}{2\pi\sigma^2}\left(\frac{x^2+y^2-2\sigma^2}{\sigma^4}\right)e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (1)$$

And following is the code for the implementation of LoG.

```
def create_gaussian_kernel(size, sigma, verbose=False):
    kernel_1D = np.linspace(-(size // 2), size // 2, size)
    kernel_1D = np.exp(-(kernel_1D**2) / (2 * sigma**2)) / (sigma * np.sqrt(2 * np.pi))
    kernel_2D = np.outer(kernel_1D.T, kernel_1D.T)
    kernel_2D *= 1.0 / kernel_2D.max()
    return kernel_2D
sigma = 0.6375
gaussian_kernel = create_gaussian_kernel(5, sigma, verbose=False)
laplace_of_gaussian_kernel = cv.Laplacian(gaussian_kernel, cv.CV_64F) * sigma**2
laplace_of_gaussian_img = cv.filter2D(img, -1, laplace_of_gaussian_kernel)
threshold_value = 81 # Adjust as needed
_, binary_img = cv.threshold(np.abs(laplace_of_gaussian_img), threshold_value, 255, cv.THRESH_BINARY)
contours, _ = cv.findContours(binary_img.astype(np.uint8), cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
original_img = cv.cvtColor(img, cv.COLOR_GRAY2BGR)
for contour in contours:
    ((x, y), radius) = cv.minEnclosingCircle(contour)
    center = (int(x), int(y))
    radius = int(radius)
    cv.circle(original_img, center, radius, (0, 255, 0), 2)  # (0, 255, 0) is color (green), 2 is thickness
```

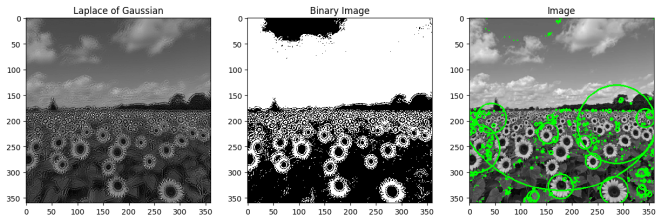

Figure 1: LoG, thresholded LoG and detected blobs

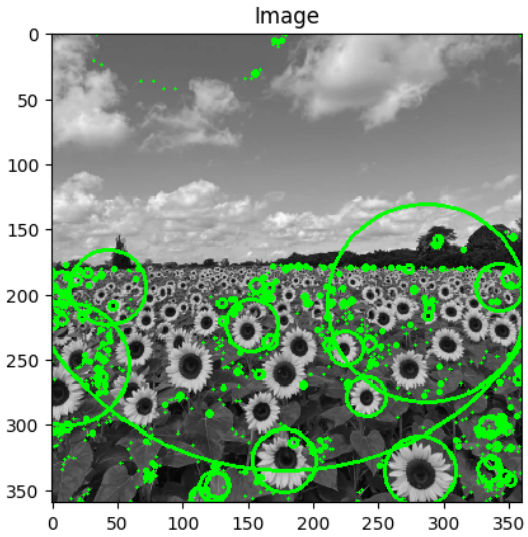

Figure 2: Detected blobs

- Maximum radius got : 76
- Range of σ values : 0.5, 0.6, 0.75, 1.0, 2.0, 3.0
- Selected σ value : 0.6375

2 Question 02

In this question we are supposed to estimate a line and a circle for a noisy dataset using RANSAC algorithm. RANSAC is an iterative method to estimate parameters of a mathematical model from a set of observed data that contains outliers.

Following will be the code for the implementation of RANSAC algorithm for predicting a line for a noisy dataset.

```
# create dataset
N = X.shape[0]
data = X
def calculate_distance(x1, y1, x2, y2):
    distance_mag =  math.sqrt((x2-x1)**2 + (y2-y1)**2)
    a = (x2-x1)/distance_mag
    b = (y2-y1)/distance_mag
    d = a*x1 + b*y1
    return a , b, d
def calc_tls (x,indices):
    a,b,d = x[0],x[1],x[2]
    return np.sum(np.square(a*data[indices,0] + b*data[indices,1] - d))
def g(x):
    return x[0]**2 + x[1]**2 - 1
constraints = ({'type': 'eq', 'fun': g})
def best_fit_line(X,x,t):
    a,b,d = x[0],x[1],x[2]
    e = np.absolute(a*X[:,0] + b*X[:,1] - d)
    return e < t
while iters < max_iters:
    indices = np.random.randint(0, N, 2)
    x0 = np.array([1,1,0])
    res = minimize(fun = calc_tls, args = indices, x0 = x0, tol= 1e-6, constraints=constraints, options={'disp': True})
    line_inliers = best_fit_line(data,res.x,threshold_value)

    if line_inliers.sum() > d :
        x0 = res.x
        res = minimize(fun = calc_tls, args = line_inliers, x0 = x0, tol= 1e-6, constraints=constraints, options={'disp': True})
        if res.fun < best_error:
            best_error = res.fun
            best_line_model = res.x
            best_sample_line = data[indices,:]
            best_inlier_line = line_inliers
            res_only_with_sample = x0
    iters += 1
```

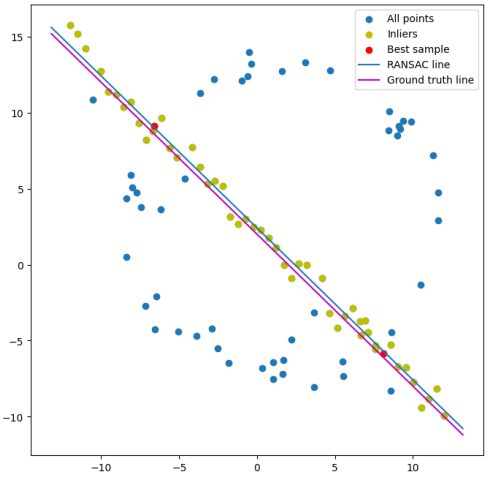

Figure 3: Best fit line for the noisy dataset

Code for estimating the circle

```
def fit_circle_ransac(points, num_iterations, error_threshold, min_consensus):
    best_circle = None
    best_inliers = []
    for _ in range(num_iterations):
        sample_indices = np.random.choice(len(points), 3, replace=False)
        sample_points = points[sample_indices]
        x, y, r = calculate_circle_parameters(sample_points)
        radial_errors = np.abs(np.sqrt((points[:, 0] - x)**2 + (points[:, 1] - y)**2) - r)
        inliers = np.where(radial_errors < error_threshold)[0]
        if len(inliers) >= min_consensus and len(inliers) > len(best_inliers):
            best_inliers = inliers
            best_circle = (x, y, r)
    return best_circle, best_inliers
```

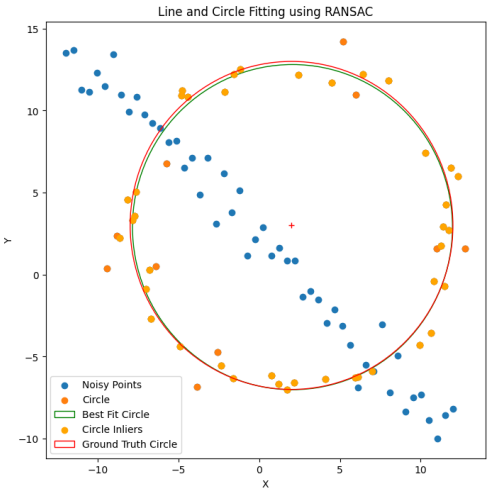

Figure 4: Best fit circle for the noisy dataset

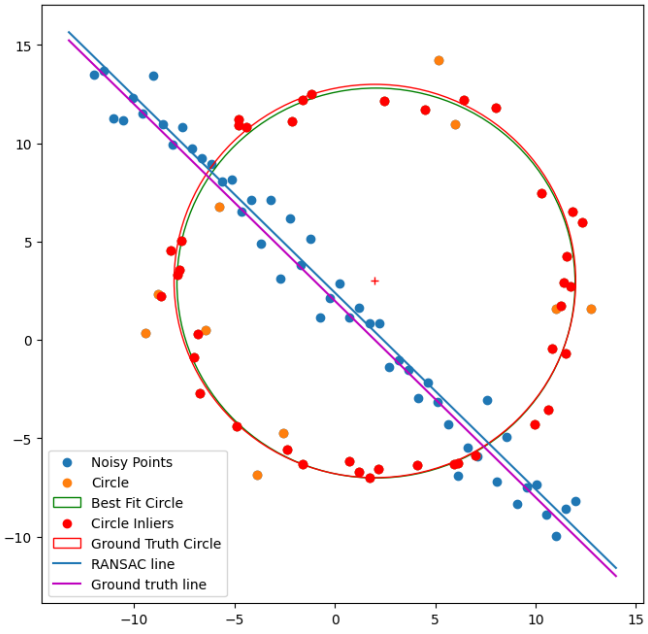

Figure 5: Best fit and Grown truth line and circle.

2.1 Answer for part (d)

In this case we are doing RANSAC fit for the line and subtract it from the noisy points and then we are proceeding with the fit for circle. Because of this RANSAC the circle inliers and creates a better fit for the circle by choosing random points.

But if we perform the ransac fit for the circle first, it will select some rondom points from the line inliers also. Because of this RANSAC can predict erroneous fits.

3 Question 03

This question is about wrapping one figure and superimposing it to another image.

```
def draw_circle(event,x,y,flags,param):
    global n
    architectural_points = param[0]
    if event == cv.EVENT_LBUTTONDOWN:
        cv.circle(param[1],(x,y)
            ,5,(255,0,0),-1)
        architectural_points[n] = (x,y)
        n += 1
cv.namedWindow('Image', cv.WINDOW_AUTOSIZE)
param = [architectural_points,
    architectural_image]
cv.setMouseCallback('Image',draw_circle,
    param)
while(1):
    cv.imshow('Image', architectural_image)
    if n == number_of_points:
        break
    if cv.waitKey(20) & 0xFF == 27:
        break
flag_points = np.array([[0, 0], [flag_image
    .shape[1], 0], [flag_image.shape[1],
    flag_image.shape[0]], [0, flag_image.
    shape[0]]], dtype=np.float32)
homography_matrix, _ = cv.findHomography(
    flag_points, architectural_points)
flag_warped = cv.warpPerspective(flag_image
    , homography_matrix, (
    architectural_image.shape[1],
    architectural_image.shape[0]))
blended_image = cv.addWeighted(
    architectural_image, 1, flag_warped,
    0.7, 0)
```

Figure 6: Superposition Example 01

Figure 7: Superposition Example 02

4 Question 04

In this question we are supposed to stich to images together. For this we need to create keypoints and descriptors for both images and then match them using a matcher. Then we need to find the homography matrix and warp the image to the other image. We can SIFT for creating keypoints and descriptors. And we can use RANSAC for finding the best homography matrix along with BFmatcher for matching the descriptors.

Following will be the implementation for creating keypoints and descriptors.

```
sift = cv2.SIFT_create()
keypoints1, descriptors1 = sift.
    detectAndCompute(img1, None)
keypoints5, descriptors5 = sift.
    detectAndCompute(img5, None)
img1_with_keypoints = cv2.drawKeypoints(
    img1, keypoints1, img1)
img5_with_keypoints = cv2.drawKeypoints(
    img5, keypoints5, img5)
```

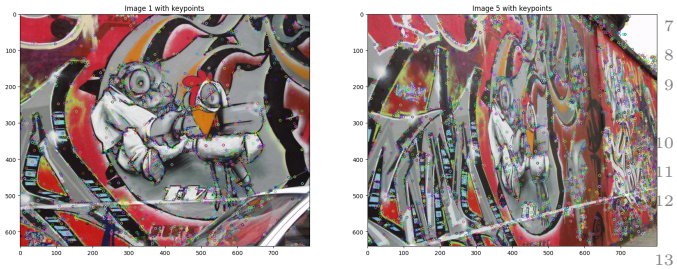

Figure 8: EDetected Keypoints

And the keypoints were mathched using BFmatcher and the homography matrix was found using RANSAC. And the following is the code for that.

```
bf = cv2.BFMatcher()
matches = bf.knnMatch(descriptors1, descriptors5, k=2)
good_matches = []
for m, n in matches:
    if m.distance < 0.85 * n.distance:
        good_matches.append(m)
src_pts = np.float32([keypoints1[m.queryIdx].pt for m in good_matches]).reshape(-1, 1, 2)
dst_pts = np.float32([keypoints5[m.trainIdx].pt for m in good_matches]).reshape(-1, 1, 2)
homography_matrix, _ = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 0.75)
```

Figure 9: Matched Keypoints

And I got the following matrix as my homography matrix. for the above code.

$$\begin{bmatrix} -3.99590 \times 10^{-1} & 1.98442 \times 10^{-1} & 1.73728 \times 10^{2} \\ -9.24469 \times 10^{-1} & 4.81236 \times 10^{-1} & 3.95571 \times 10^{2} \\ -2.32774 \times 10^{-3} & 1.19405 \times 10^{-3} & 1.00000 \end{bmatrix}$$

```
def calculate_error(homography, src_points, dst_points):
    transformed_points = cv2.perspectiveTransform(src_points, homography)
    errors = np.sqrt(np.sum((transformed_points - dst_points)**2, axis=2))
    return np.mean(errors)
def ransac_homography(src_points, dst_points, num_iterations=100, error_threshold=1.0):
    best_homography = None
    best_error = float('inf')
    for _ in range(num_iterations):
        indices = np.random.choice(len(src_points), 4, replace=False)
        sampled_src = src_points[indices]
        sampled_dst = dst_points[indices]
        homography, _ = cv2.findHomography(sampled_src, sampled_dst)
        current_error = calculate_error(homography, src_points, dst_points)
        if current_error < best_error:
            best_error = current_error
            best_homography = homography
        if current_error < error_threshold:
            break

    return best_homography
```

Figure 10: Matched Keypoints using RANSAC

And I tried to stich the matching keypoints but I could not able to accomplish it.

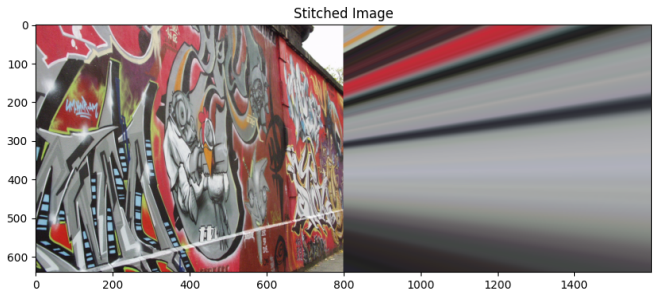

Figure 11: Stiched Image

5 References

- Numpy Documentation
- OpenCV Documentation

6 Github Repository

Following is the link to my Github repository for this assignment.

Github/EN160_Assignment_02