EN3160 Assignment 2 on Fitting and Alignment

Index Number: 200087A

Y.U.K.K. Chandrasiri

[Image-Processing/Fitting and Allignment at main · kavindukalinga/Image-Processing (github.com)](https://github.com/kavindukalinga/Image-Processing)

....

1. Blob Detection

In this section, using the knowledge on blob detection, i.e., using Laplacian of Gaussians and scale-space extrema detection, we will detect and draw circles in the sunflower field image provided.

Range of σ values used for the maximum blob detection: range (5,10.5,0.5)

The parameters of the largest circle:
Sigma=9.5, Radius=13.435

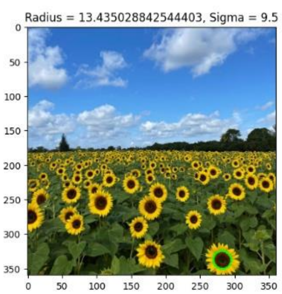

Figure: Maximum blob detection

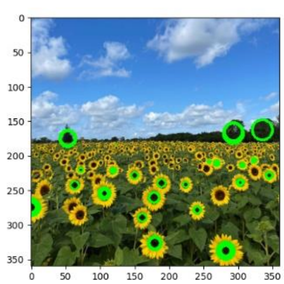

Figure: All the blobs detection

Range of σ values used for all the blobs detection range (0.5,11.5,0.25)

```python
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

input_image = cv.cvtColor(im, cv.COLOR_RGB2GRAY)
sigma_values = np.arange(5, 10.5, 0.5)

for sigma in sigma_values:
    image_copy = im.copy()
    scale_space = np.empty((im.shape[0], im.shape[1], 500), dtype=np.float64)
    sigmas = np.arange(sigma, sigma + 0.5, 0.01)

    for i, current_sigma in enumerate(sigmas):
        log_hw = 3 * np.ceil(np.max(sigmas))
        X, Y = np.meshgrid(np.arange(-log_hw, log_hw + 1, 1), np.arange(-log_hw, log_hw + 1, 1))
        log_filter = 1 / (2 * np.pi * current_sigma ** 2) * (X ** 2 / (current_sigma ** 2) +\
            Y ** 2 / (current_sigma ** 2) - 2) * np.exp(-(X ** 2 + Y ** 2) / (2 * current_sigma ** 2))
        filtered_log = cv.filter2D(input_image, cv.CV_64F, log_filter)
        scale_space[:, :, i] = filtered_log

    max_indices = np.unravel_index(np.argmax(scale_space, axis=None), scale_space.shape)
    radius = sigmas[max_indices[2]] * np.sqrt(2)
    cv.circle(image_copy, (int(max_indices[1]), int(max_indices[0])), int(radius), (0, 255, 0), 2)

    fig, ax = plt.subplots()
    plt.title('Radius = ' + str(radius) + ', Sigma = ' + str(sigmas[max_indices[2]]))
    plt.imshow(image_copy)
```

2.Estimate using the RANSAC Algorithm

In this section, we will fit a line and, subsequently, a circle to a set of noisy points that conform to a line and a circle.

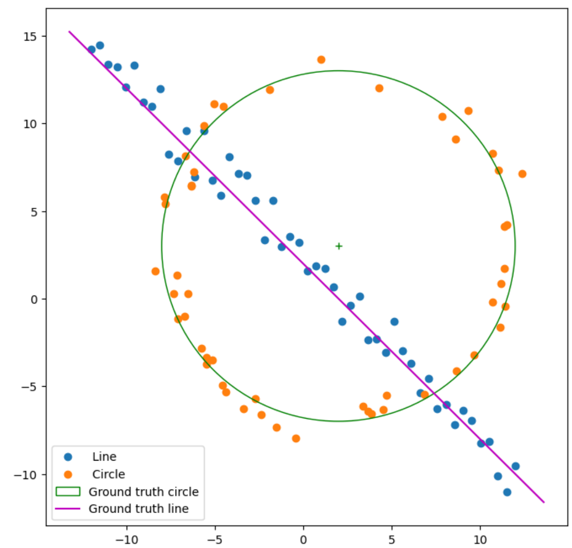

Figure: The generated noisy point set X amounting to a circle and a line

Estimate the line using the RANSAC Algorithm

```python
points = all_points

for _ in range(num_iterations):
    selected_indices = np.random.choice(len(points), size=min_sample_count, replace=False)
    selected_points = points[selected_indices]
    a, b, d = create_line(selected_points[:, 0], selected_points[:, 1])
    norm = np.sqrt(a**2 + b**2)
    a /= norm
    b /= norm
    d /= norm

    consensus_set = []

    for i, point in enumerate(points):
        if calculate_distance_to_line(point, [a, b, d]) < max_distance_threshold:
            consensus_set.append(i)

    if len(consensus_set) > 42:
        break

xc_consensus = points[consensus_set, 0]
yc_consensus = points[consensus_set, 1]
x_coords = points[:, 0]
y_coords = points[:, 1]
```

Consensus threshold = 2, Number of points = 42

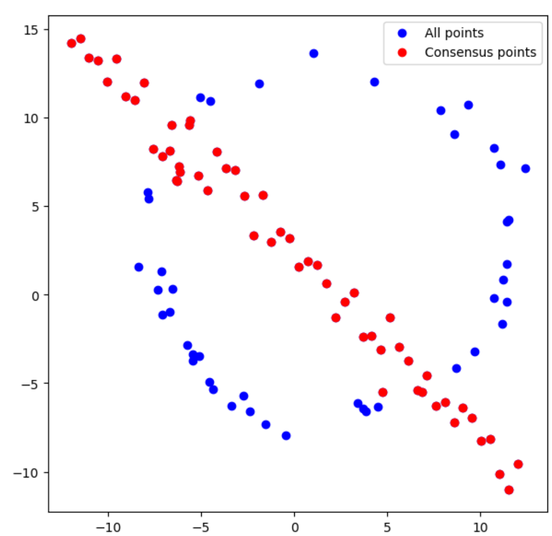

Figure: The Consensus points

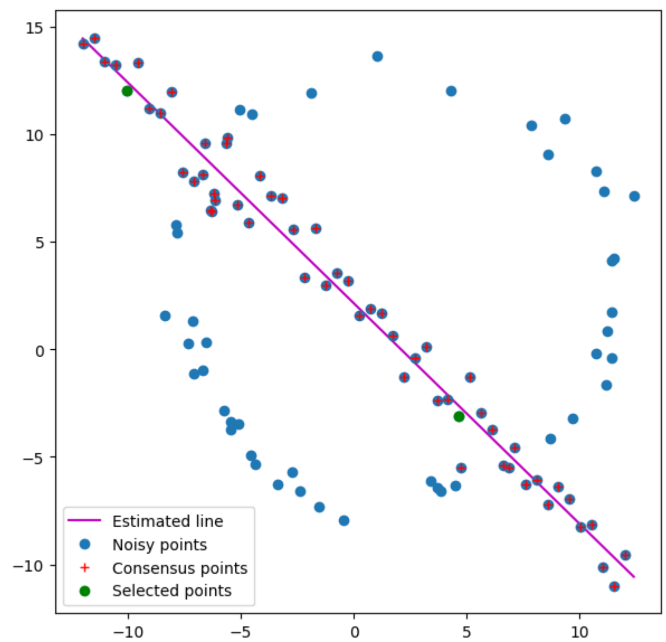

Figure: The Estimated line

The estimated line:

$$y = -1.02610304x + 2.13986715$$

Ground Truth: $y = -1x + 2$

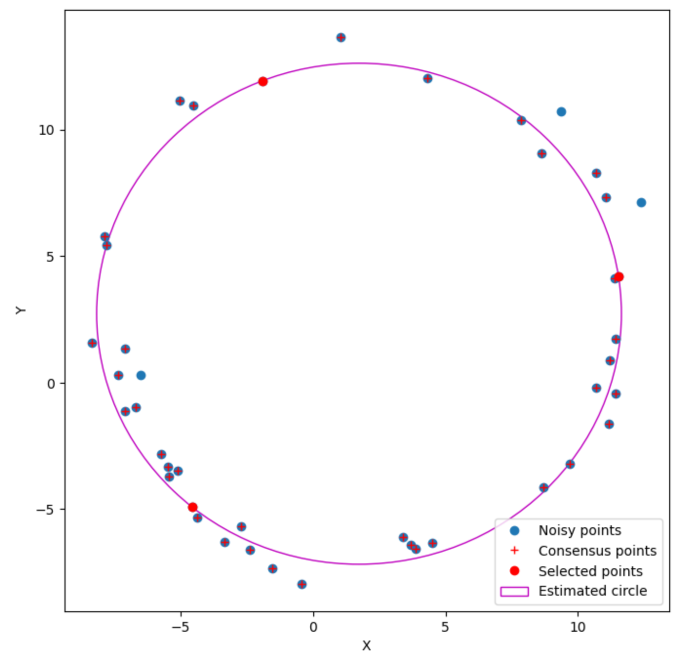

Figure: The Estimated circle

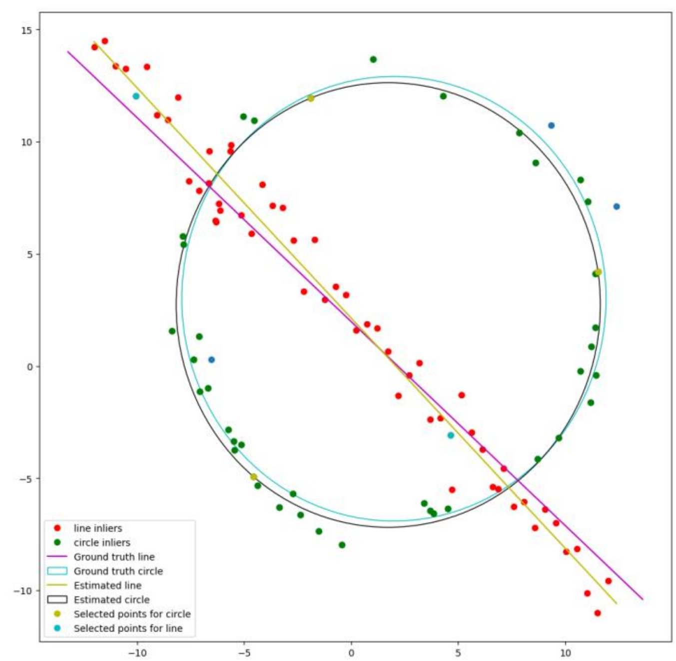

Figure: In the same plot, the point set, the line estimated from the sample leading to the best estimate, the circle estimated from the sample leading to the best estimate, this sample of three points, the best fit line, line inliers, the the best-fit circle and circle inliers.

Estimate the circle that fits the remnant

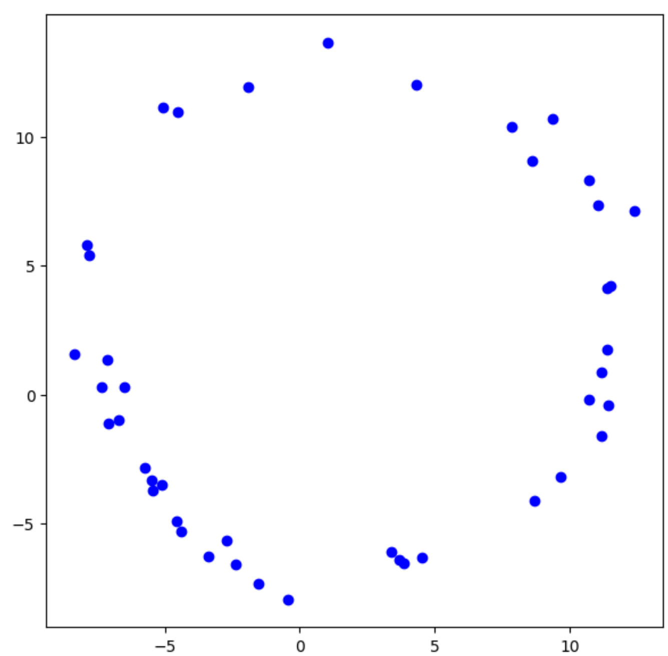

Figure: After subtracting the consensus of the best line (remnant) and estimate the circle that fits the remnant using RANSAC

Number of points = 35

What happen if the circle is fit first

Then some points in the intersection that used to fit the line, now used to fit the circle. Therefore, the estimated line and circle differ from the previous estimations.

3. Superimpose Images

In the below figure shows an architectural image with a flag superimposed. This is done by clicking four points on a planar surface in the architectural image, computing a homography that maps the flag image to this plane, and warping the flag, and blending on to the architectural image.

```python
def measure_images(image_1, image_2):
    coordinates = []
    def click_event(event, x, y, flags, param):
        nonlocal coordinates
        if event == cv.EVENT_LBUTTONDOWN:
            coordinates.append((x, y))
            cv.circle(image_1, (x, y), 5, (0, 0, 255), -1)
            cv.imshow('Image', image_1)
            if len(coordinates) == 4:
                cv.destroyAllWindows()
    cv.imshow('Image', image_1)
    cv.setMouseCallback('Image', click_event)
    while len(coordinates) < 4:
        cv.waitKey(1)
    image_1_points = np.array(coordinates, dtype=np.float32)
    image_2_width = image_2.shape[1]
    image_2_height = image_2.shape[0]
    image_2_points = np.array([[0, 0], [image_2_width, 0], [image_2_width, image_2_height], [0, image_2_height]], dtype=np.float32)

    return image_1_points, image_2_points

def superimpose_images(image_1, image_2, image_1_points, image_2_points, alpha , beta):
    # Compute the homography matrix
    homography_matrix= cv.findHomography(image_2_points, image_1_points)[0]

    # Warp the flag image to fit the architectural image
    image_2_warped = cv.warpPerspective(image_2, homography_matrix, (image_1.shape[1], image_1.shape[0]))

    #Blend the images
    blended_image = cv.addWeighted(image_1, alpha, image_2_warped, beta, 0)

    return blended_image
```

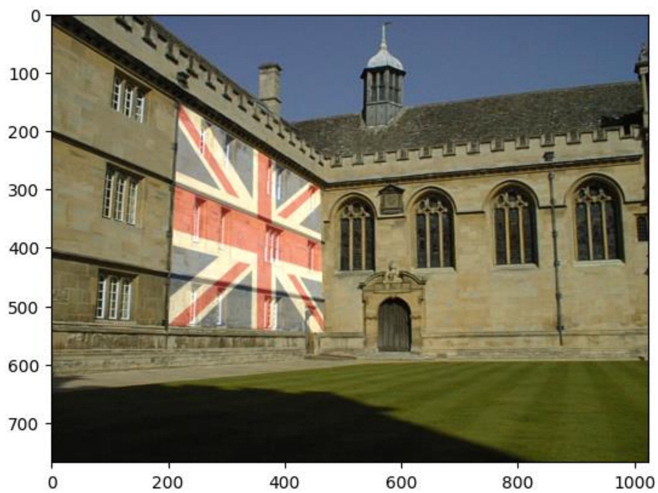

Figure: Flag of UK in an old building in UK

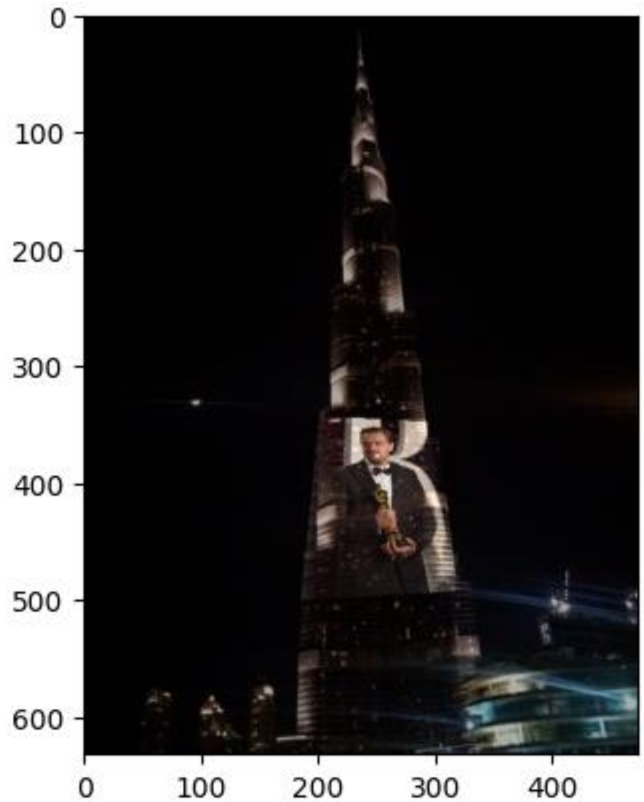

Figure: Putting one of my favorite moments in Academy Award history, Leonardo DiCaprio winning an academy award for the best actor, on the Burj Khalifa tower in UAE. Only the most remarkable events in the world presents in the Burj Khalifa tower and it was astonishing that this particular achievement wasn't among them, therefore I made that into reality using "Superimpose images" technique.

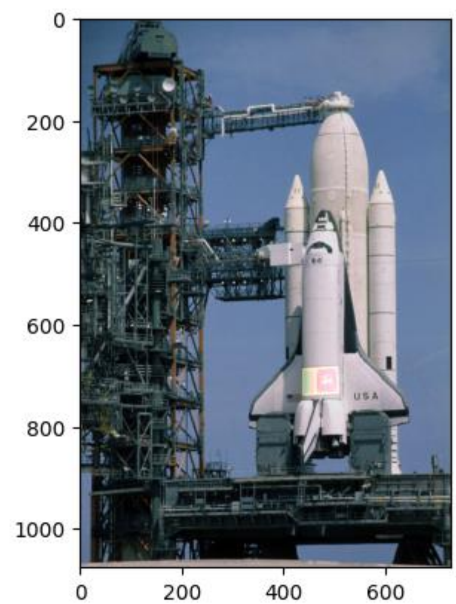

Figure: First Sri Lankan rocket to go out of atmosphere:

The result of a mission between Sri Lanka and USA

4. Stitch two Graffiti Images

SIFT features between the two images are computed and matched in this task.

```python
sift = cv.SIFT_create()
key_points_1, descriptors_1 = sift.detectAndCompute(im1,None)
key_points_2, descriptors_2 = sift.detectAndCompute(im5,None)
bf_match = cv.BFMatcher(cv.NORM_L1, crossCheck=True)
matches = sorted(bf_match.match(descriptors_1, descriptors_2), key = lambda x:x.distance)

im = cv.drawMatches(im1, key_points_1, im5, key_points_2, matches[:250], im5, flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
fig, ax = plt.subplots(figsize=(12,12))
im = cv.cvtColor(im, cv.COLOR_BGR2RGB)
```

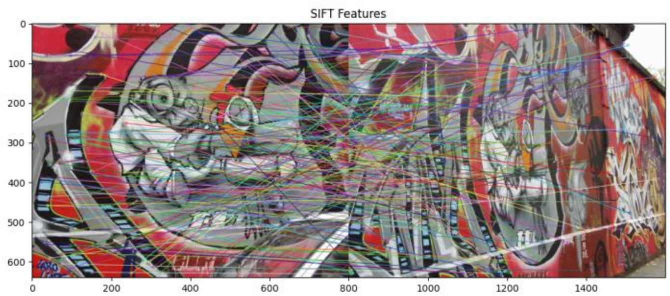

Figure: SIFT Features between Img1.ppm and Img5.ppm

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:

```
✓ 6.3s

Computed Homography
 [[ 6.15836123e-01  4.70870416e-02  2.23134428e+02]
 [ 2.20887924e-01  1.13194518e+00 -2.15537405e+01]
 [ 4.89461445e-04 -9.09548051e-05  1.00000000e+00]]
```

Stitch img1.ppm onto img5.ppm

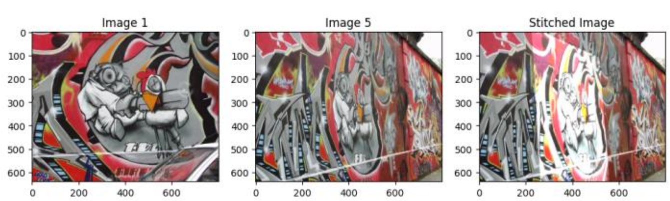

Figure: img1.ppm, img5.ppm and the stitched image (img1.ppm stitched onto img5.ppm)

Appendix

```python
def Homography(p1, p2):
    x1, y1, x2, y2, x3, y3, x4, y4 = p2[0], p2[1], p2[2], p2[3], p2[4], p2[5], p2[6], p2[7]
    x1T, x2T, x3T, x4T = p1[0], p1[1], p1[2], p1[3]
    zero_matrix = np.array([[0], [0], [0]])

    matrix_A = np.concatenate((np.concatenate((zero_matrix.T,x1T, -y1*x1T), axis = 1), np.concatenate((x1T, zero_matrix.T, -x1*x1T), axis = 1),
                               np.concatenate((zero_matrix.T,x2T, -y2*x2T), axis = 1), np.concatenate((x2T, zero_matrix.T, -x2*x2T), axis = 1),
                               np.concatenate((zero_matrix.T,x3T, -y3*x3T), axis = 1), np.concatenate((x3T, zero_matrix.T, -x3*x3T), axis = 1),
                               np.concatenate((zero_matrix.T,x4T, -y4*x4T), axis = 1), np.concatenate((x4T, zero_matrix.T, -x4*x4T), axis = 1)), axis = 0, dtype=np.float64)
    W, v = np.linalg.eig((matrix_A.T)@matrix_A))
    temph= v[:,np.argmin(W)]
    H = temph.reshape((3,3))
    return H
```

```python
N = int(np.ceil(np.log(1-p)/np.log(1-((1-e)**s))))
Hs = []
for i in range(4):
    sift = cv.SIFT_create()
    key_points_1, descriptors_1 = sift.detectAndCompute(ims[i],None) #sifting
    key_points_2, descriptors_2 = sift.detectAndCompute(ims[i+1],None)
    bf_match = cv.BFMatcher(cv.NORM_L1, crossCheck=True)  #feature matching
    matches = sorted(bf_match.match(descriptors_1, descriptors_2), key = lambda x:x.distance)

    Source_Points = [key_points_1[k.queryIdx].pt for k in matches]
    Destination_Points = [key_points_2[k.trainIdx].pt for k in matches]
    threshold, best_inliers, best_H = 2, 0, 0

    for i in range(N):
        ran_points = random_number(len(Source_Points)-1, 4)
        f_points = []
        for j in range(4):
            f_points.append(np.array([[Source_Points[ran_points[j]][0], Source_Points[ran_points[j]][1], 1]]))
        t_points = []
        for j in range(4):
            t_points.append(Destination_Points[ran_points[j]][0])
            t_points.append(Destination_Points[ran_points[j]][1])
        H = Homography(f_points,t_points)
        inliers = 0
        for k in range(len(Source_Points)):
            X = [Source_Points[k][0], Source_Points[k][1], 1]
            HX = H @ X
            HX /= HX[-1]
            err = np.sqrt(np.power(HX[0]-Destination_Points[k][0], 2) + np.power(HX[1]-Destination_Points[k][1], 2))
            if err < threshold:
                inliers +=1
        if inliers > best_inliers:
            best_inliers = inliers
            best_H = H
    Hs.append(best_H)
H1_H5 = Hs[3] @ Hs[2] @ Hs[1] @ Hs[0]
H1_H5 /= H1_H5[-1][-1]
```