EN3160 - Image Processing and Machine Vision

Assignment 02 - Fitting and Alignment

- Index No : 210095F
- Name : D.M.T.K.R Dassanayake

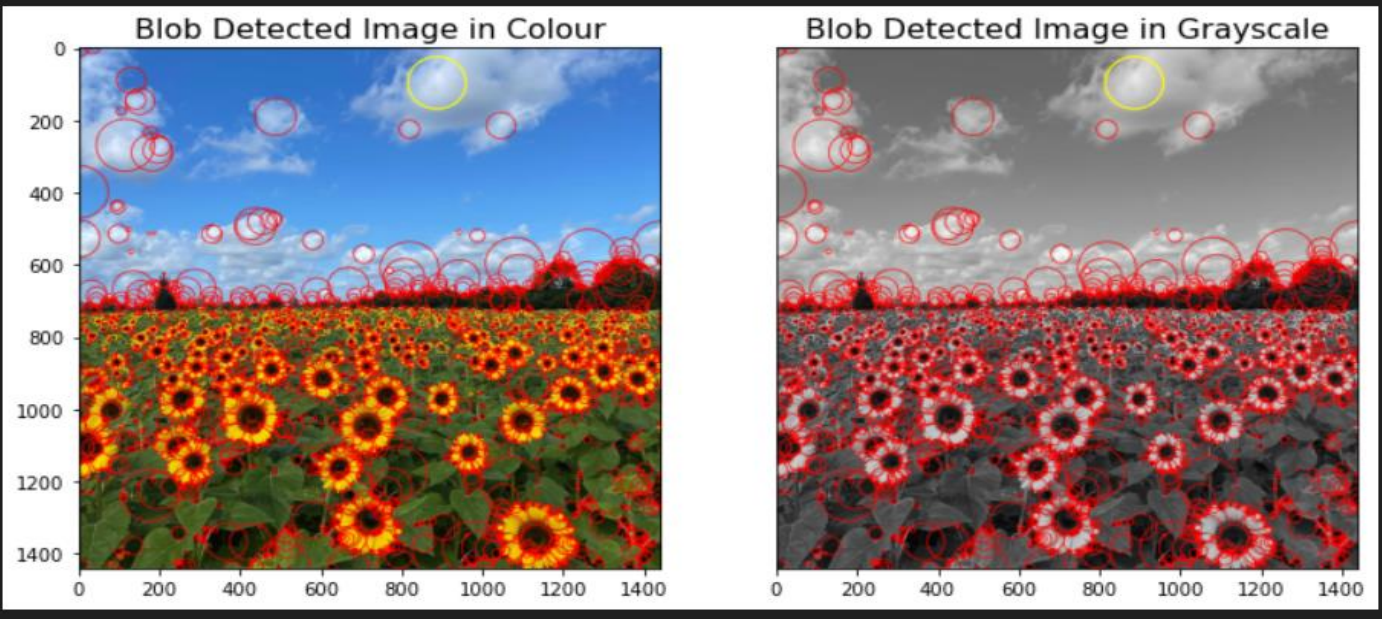
Github : https://github.com/Thathsara-Dassanayake/Computer-vision-and-Image-Processing---Fitting-and-Alignment

1.) Question 1

```
# Draw the detected blobs on the output images
for i in range(np.size(x_vec)):
        if i == max_circle_idx:
                # Draw the largest blob in yellow
                cv.circle(out_colour, (y_vec[i], x_vec[i]), int(radius_vec[i]), (0,255,255), 3)
                cv.circle(out_grayscale, (y_vec[i], x_vec[i]), int(radius_vec[i]), (0,255,255), 3)
        else:
                # Draw smaller blobs in red
                cv.circle(out_colour, (y_vec[i], x_vec[i]), int(radius_vec[i]), (0, 0, 255), 2)
                cv.circle(out_grayscale, (y_vec[i], x_vec[i]), int(radius_vec[i]), (0, 0, 255), 2)
```

```
# Define initial sigma, scale factor, and the number of scales
sigma0 = 0.4  # Starting sigma value (selected empirically)
k = np.sqrt(2)  # Scaling factor for sigma
num_scales = 15  # Number of scales to analyze
sigmas = sigma0 * np.power(k, np.arange(num_scales))  # Array of sigma values for each scale
```

```
Largest Circle Parameters:
Radius: 72.40773439350254
Center Coordinates (x, y): (98, 884)

Range of σ values used:
Minimum σ: 0.4
Maximum σ: 51.200000000000045
```



2.) Question 2

a.)

```python
def find_line_parameters_using_2_points(x1,y1,x2,y2):

    m = (y2-y1)/(x2-x1)
    c = y1 - m*x1

    a = m
    b = -1
    d = -c

    a_normalized = a/math.sqrt(a**2+b**2)
    b_normalized = b/math.sqrt(a**2+b**2)
    d_normalized = d/math.sqrt(a**2+b**2)

    return a_normalized,b_normalized,d_normalized
```

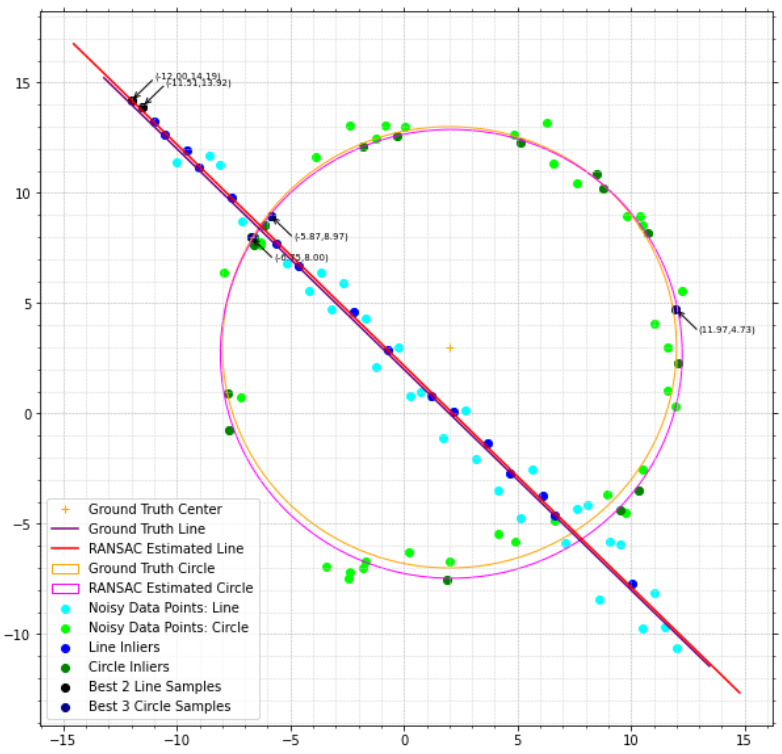
```python
def RANSAC_line_fitting(X, iterations, threshold, min_inliers):

    best_model = None
    best_inliers = []

    for i in range(iterations):

        sample_indices = np.random.choice(len(X),2,replace=False)
        x1,y1 = X[sample_indices[0]]
        x2,y2 = X[sample_indices[1]]

        a,b,d = find_line_parameters_using_2_points(x1,y1,x2,y2)

        magnitude = np.sqrt(a**2 + b**2)
        a = a/magnitude
        b = b/magnitude

        distances = np.abs(a*X[:,0]+b*X[:,1]-d)

        inliers  = np.where(distances<threshold)[0]

        if len(inliers) >= min_inliers:
            if len(inliers) > len(best_inliers):
                best_model  = (a,b,d)
                best_inliers = inliers

    return best_model,best_inliers
```

b.)

```python
def RANSAC_circle_fitting(X,iterations,threshold,min_inliers):

    best_model  = None
    best_inliers = []

    for i in range(iterations):
        sample_indices = np.random.choice(len(X),3,replace=False)
        x1,y1 = X[sample_indices[0]]
        x2,y2 = X[sample_indices[1]]
        x3,y3 = X[sample_indices[2]]

        x_center,y_center,radius = find_circle_parameters_using_3_points(x1,y1,x2,y2,x3,y3)

        errors  = np.abs(np.sqrt((X[:,0]-x_center)**2+(X[:,1]-y_center)**2)-radius)

        inliers = np.where(errors<threshold)[0]

        if len(inliers) >= min_inliers:
            if len(inliers) > len(best_inliers):
                best_model  = (x_center,y_center,radius)
                best_inliers = inliers

    return best_model,best_inliers
```

```python
def find_circle_parameters_using_3_points(x1,y1,x2,y2,x3,y3):

    mx1,my1 = (x1+x2)/2,(y1+y2)/2
    mx2,my2 = (x2+x3)/2,(y2+y3)/2

    if y2-y1 == 0:
        slope1 = 0
    else:
        slope1 = (x2-x1)/(y2-y1)

    if y3-y2 == 0:
        slope2 = 0
    else:
        slope2 = (x3-x2)/(y3-y2)

    x_center = (slope1*mx1-slope2*mx2+my2-my1)/(slope1-slope2)
    y_center = -slope1*(x_center-mx1)+my1

    radius = np.sqrt((x1-x_center)**2+(y1-y_center)**2)

    return x_center,y_center,radius
```

c.)



d.) **If we attempt to fit the circle first, the points that actually belong to the line will likely produce high radial errors when evaluated against the circle model. This is because line points are far from conforming to a circular shape, and their distances from the estimated circle center will be inconsistent. As a result, the RANSAC algorithm will struggle to identify a good circle model, leading to poor circle fitting.**

3.) Question 3

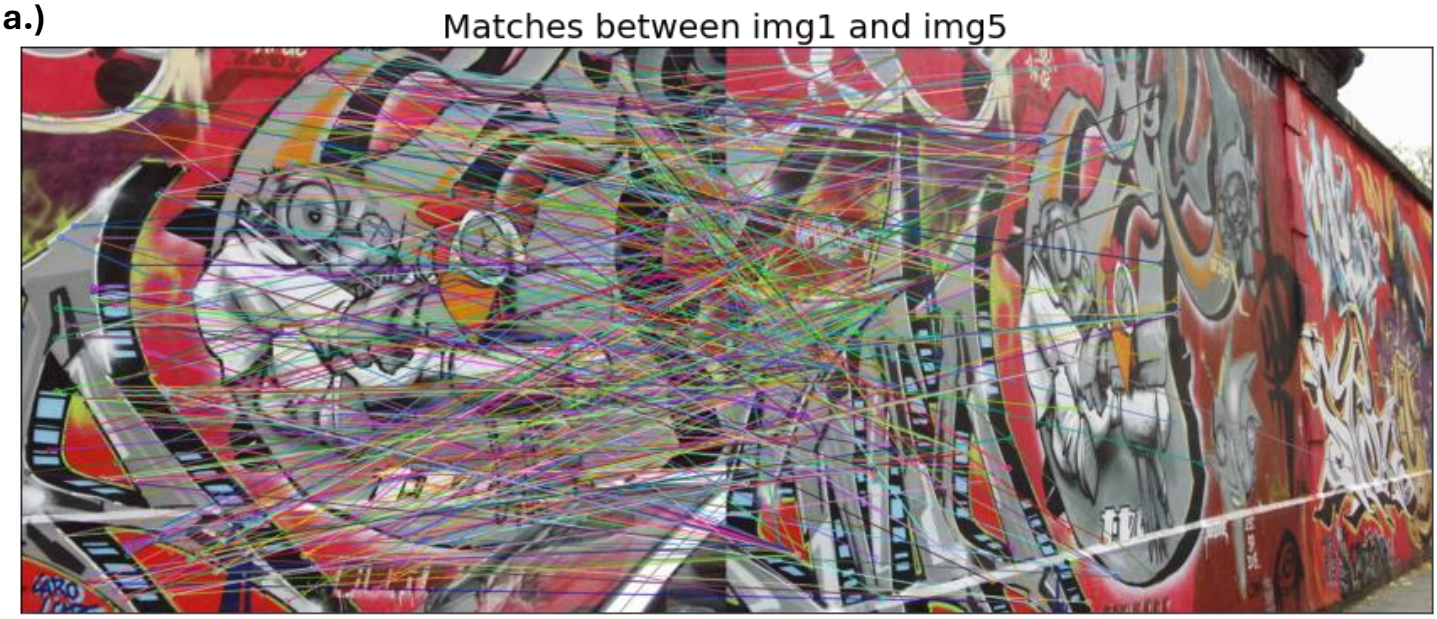
- Compute homography, warp flag and blend

```
H = cv2.getPerspectiveTransform(src_points, dst_points)  # Homography matrix
print('Homography matrix\n', H)

flag_warped = cv2.warpPerspective(flag, H, (architectural_image.shape[1], architectural_image.shape[0]))
superimposed_image = cv.addWeighted(architectural_image, 1, flag_warped, 0.5, 0)
```

```
Homography matrix
 [[ 1.66111998e-01 -2.88983207e-01  5.01000000e+02]
 [-1.16903934e-01  2.44539802e-01  1.68000000e+02]
 [-2.66599873e-04 -5.12530020e-04  1.00000000e+00]]
```

4.) Question 4

a.)



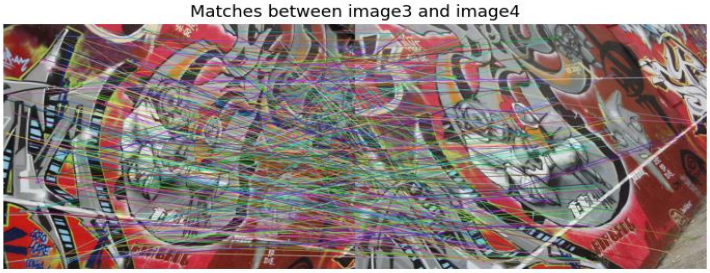
b.)

- The matches between img1 and img5 are subpar. Calculating the homography matrix based on these matches will yield poor results. Afterward, combine all these homography matrices to derive the final homography matrix.

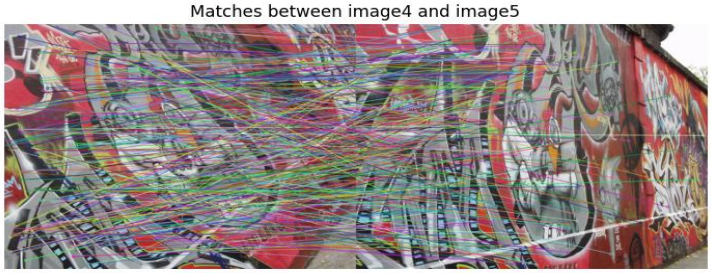
Matches between image1 and image2

Matches between image2 and image3

Matches between image3 and image4



Matches between image4 and image5



```
Calculated Homography
 [[ 7.50767642e-01  1.44871582e-01  1.89851849e+02]
 [ 3.06867140e-01  1.32819838e+00 -7.02388117e+01]
 [ 6.90913005e-04  7.49002820e-05  1.00000000e+00]]
Number of inliers 255

Original Homography
    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
```

c.)

Using the calculated homography matrix

img1

img5

Perspective transformed image

Final blended image

Using the homography matrix given in the dataset

img1

img5

Perspective transformed image

Final blended image