# EN3160 Assignment 2 on Fitting and Alignment

**Index No: 200285E** **Name : Nuwan Kannangara**

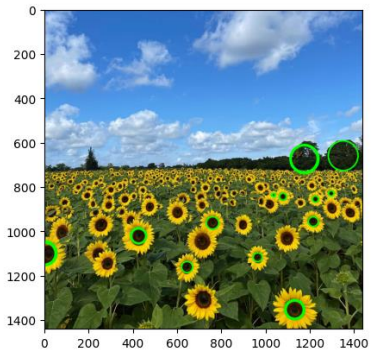
GitHub link: https://github.com/kannangaranv/EN3160_Assignment_2

## Question 1

### Code:

```
start = np.arange(4,50,2)
for k in start:
    scale_space = np.empty((image.shape[0] ,image.shape[1] , 20), dtype=np.float64)
    sigmas = np.arange(k,k+2, 0.1)
    for i, 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 = 1/(2*np.pi*sigma**2)*(X**2/(sigma**2) + Y**2/(sigma**2) - 2)*np.exp(-(X**2 +
Y**2)/(2*sigma**2))
        f_log = cv.filter2D(image_g, cv.CV_64F, log)
        scale_space[:, :, i] = f_log
    indices = np.unravel_index(np.argmax(scale_space, axis=None), scale_space.shape)
    r = sigmas[indices[2]]*np.sqrt(2)
    col = (np.random.random(), np.random.random(), np.random.random())
    cv.circle(image, (int(indices[1]), int(indices[0])), int(r), (0,255,0), 6)
```

### Output:




```
sigma range :  4 - 6
radius of largest circle :  7.353910524340089
sigma range :  6 - 8
radius of largest circle :  8.485281374238571
sigma range :  8 - 10
radius of largest circle :  12.445079348883233
sigma range :  10 - 12
radius of largest circle :  14.142135623730951
sigma range :  12 - 14
radius of largest circle :  16.970562748477143
sigma range :  14 - 16
radius of largest circle :  20.647518010647186
sigma range :  16 - 18
radius of largest circle :  22.627416997969522
sigma range :  18 - 20
radius of largest circle :  25.455844122715714
sigma range :  20 - 22
radius of largest circle :  28.284271247461902
sigma range :  22 - 24
radius of largest circle :  32.52691193458121
sigma range :  24 - 26
radius of largest circle :  33.941125496954285
sigma range :  26 - 28
radius of largest circle :  36.76955262170048
sigma range :  28 - 30
radius of largest circle :  39.59797974644666
sigma range :  30 - 32
radius of largest circle :  42.42640687119285
```

```
sigma range :  32 - 34
radius of largest circle :  45.254833995939045
sigma range :  34 - 36
radius of largest circle :  48.083261120685236
sigma range :  36 - 38
radius of largest circle :  50.91168824543143
sigma range :  38 - 40
radius of largest circle :  53.74011537017761
sigma range :  40 - 42
radius of largest circle :  56.568542494923804
sigma range :  42 - 44
radius of largest circle :  59.396969619669996
sigma range :  44 - 46
radius of largest circle :  62.22539674441619
sigma range :  46 - 48
radius of largest circle :  65.05382386916237
sigma range :  48 - 50
radius of largest circle :  67.88225099390857
```

### Interpretation & Discussion:

Here, created array of sigma values with a range size of two and a step of 0.1. And applied LoG filter kernel to each sigma value and stored result in the scale space array which is a 3D array of size 20. Then by using this array, identified the blob with largest radius in the current sigma value range. Finally calculated the radius of the largest blob.

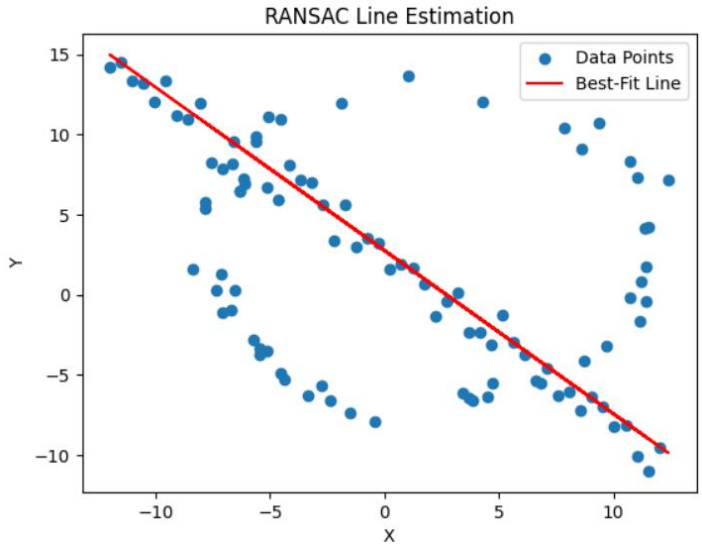
## Question 2

RANSAC algorithm to line estimation

Here, first I defined functions to estimate parameters of a line and to calculate errors. Then applied RANSAC algorithm to estimate the line. Here I took 1 as the threshold for inliers and 30 as the minimum number of inliers required to fit the line. Code of the algorithm as follows:

```
# RANSAC algorithm for line estimation
best_line = None
best_inliers_line = 0
for _ in range(max_iterations):
    # Randomly select two points
    random_indices = np.random.choice(len(X), 2,
replace=False)
    random_points = X[random_indices]
    # Estimate the line parameters [a, b, d]
    a, b, d = estimate_line(random_points)
    # Calculate the error (normal distance to the
line)
    errors = line_error([a, b, d], X)
    # Count inliers (points that are within the
threshold)
    inliers = np.sum(errors < inlier_threshold)
    if inliers >= min_inliers and inliers >
best_inliers_line:
        best_line = [a, b, d]
        best_inliers_line = inliers
```

Result is as follows:

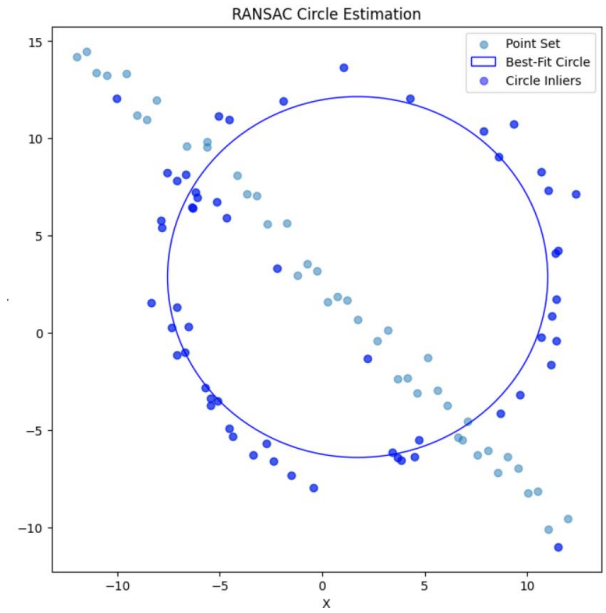



RANSAC algorithm to circle estimation

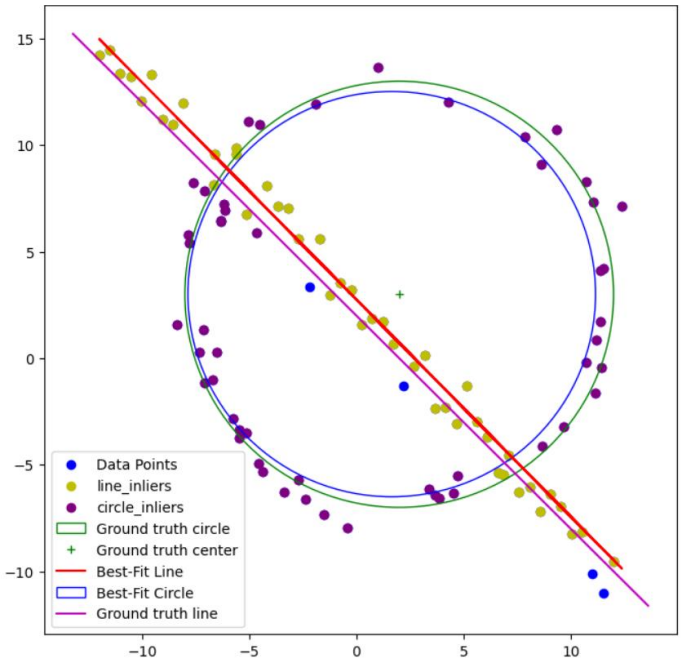
Then subtract the consensus of the best line. Next, I defined functions to estimate parameters of a circle and to calculate errors. Then applied RANSAC algorithm to estimate the circle. Here I took 3 as the threshold for inliers and 20 as the minimum number of inliers required to fit the circle. Code of the algorithm as follows:

```
# RANSAC algorithm for circle estimation
best_circle = None
best_inliers_circle = 0
for _ in range(max_iterations_circle):
    # Randomly select three points
    random_indices =
np.random.choice(len(X_remnant), 3, replace=False)
    random_points = X_remnant[random_indices]
    # Estimate the circle parameters [x, y, r]
    x, y, r = estimate_circle(random_points)
    # Calculate the radial error
    errors = circle_error([x, y, r], X_remnant)
    # Count inliers (points that are within the
threshold)
    inliers = np.sum(errors <
inlier_threshold_circle)
    if inliers >= min_inliers_circle and inliers >
best_inliers_circle:
        best_circle = [x, y, r]
        best_inliers_circle = inliers
```

Result is as follows:



Then I plotted 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, the sample of three point selected for the circle estimation, the best fit line, line inliers, the best fit circle and circle inliers in the same plot.




### 1.4 Discussion

What will happen if we fit the circle first?

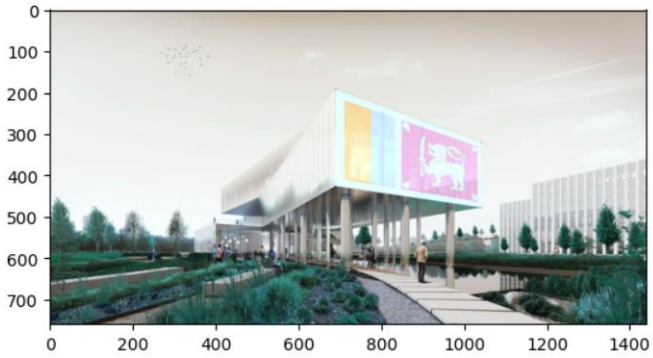
We know that circle is locally act as a line. So, if we fit the circle first, the data points corresponding to the line can fit to a circle with large radius. So, this method will not be applicable to best line and circle estimation.

## Question 3

First created a function to get the four points on a planar surface in the architectural image. Then compute the homography that maps the flag image to the plane and wrapped the flag. Then blended it to the architectural image. The code is as follows:

```
architectural_image =
cv.imread('./architectural.jpg')
flag_image = cv.imread('./flag.webp')
# Define the four points on the planar surface in
the architectural image
architectural_points =
get_points(architectural_image)
# Define the corresponding points on the flag
image
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)
# Compute the homography matrix
homography_matrix, _ =
cv.findHomography(flag_points,
architectural_points)
# Warp the flag image onto the architectural image
warped_flag = cv.warpPerspective(flag_image,
homography_matrix, (architectural_image.shape[1],
architectural_image.shape[0]))
# Blend the warped flag image with the
architectural image
result = cv.addWeighted(architectural_image, 1,
warped_flag, 0.5, 0)
```

Result is as follows:



**Discussion:**

Explanation for the choice

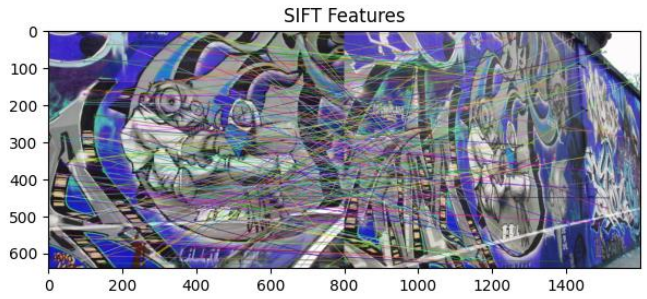
Here I choice an architectural image with a building with a planar surface. The planar surface looked like a rectangular surface. And flag also a rectangular shaped image. So, it is easy to superimpose the flag on to the building.

## Question 4

Here, first I computed and matched SIFT features between two images. The following code shows how it happens.

```
sift = cv.SIFT_create()
# Detect and compute SIFT keypoints and
descriptors
keypoints1, descriptors1 =
sift.detectAndCompute(image1, None)
keypoints5, descriptors5 =
sift.detectAndCompute(image5, None)
# Create a Brute-Force Matcher
bf_matcher = cv.BFMatcher(cv.NORM_L1,
crossCheck=True)
# Match descriptors
matches = bf_matcher.match(descriptors1,
descriptors5)
# Sort matches by distance
matches = sorted(matches, key=lambda x:
x.distance)
# Draw the first 250 matches
matched_image = cv.drawMatches(image1, keypoints1,
image5, keypoints5, matches[:250], None,
                               flags=cv.DrawMatche
sFlags_NOT_DRAW_SINGLE_POINTS)
```

Result is as follows:




Then defined a function to random sampling and homography estimation. Code for homography estimation as follows:

```
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(np.dot(matrix_A.T,
matrix_A))
    temph = v[:, np.argmin(W)]
    H = temph.reshape((3, 3))
    return H
```

Then estimate the homography matrix between each pair of consecutive images using SIFT feature matching technique and RANSAC. Then using homography matrix, wrapped the image 1 to the coordinate system of image 5. Then two images are blended together. The computed homography and provided homagraphies as follows:

```
Computed Homography =  [[ 6.30362951e-01  4.11591883e-02  2.21970116e+02]
 [ 2.24957592e-01  1.13577665e+00 -2.02511908e+01]
 [ 5.09006488e-04 -8.41212298e-05  1.00000000e+00]]
Provided 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
```

Result as follows: