EN3510 Assignment 02 - Index no: 200740V (Github : [oshanyalegama/EN3160-Assignment-01 (github.com)](github.com))

1) This question involves detecting blobs in a sunflower field image. This was done by convolving the image with a Laplassian of Gaussian filter as follows:

```
def LoG(sigma):
    '''Laplacian of Gaussian Function'''
    #window size
    n = np.ceil(sigma*6)
    y,x = np.ogrid[-n//2:n//2+1,-n//2:n//2+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))
    return log
✓ 0.0s
```

```
def LoG_convolve(img):
    '''Laplacian of Gaussian Convolution'''
    scale_space = [] #scale space
    for i in range(0,9):
        y = np.power(k,i)
        sigma_1 = sigma*y #computing value of sigma
        print(sigma_1)
        filter_log = LoG(sigma_1) #filter generation
        image = cv2.filter2D(img,-1,filter_log) # convolving image
        image = np.pad(image,((1,1),(1,1)),'constant') #padding the image
        image = np.square(image) # squaring the response
        scale_space.append(image)
    scale_space_np = np.array([i for i in scale_space]) # converting the scale space to an array
    return scale_space_np
scale_space_np = LoG_convolve(img)
✓ 0.0s
```

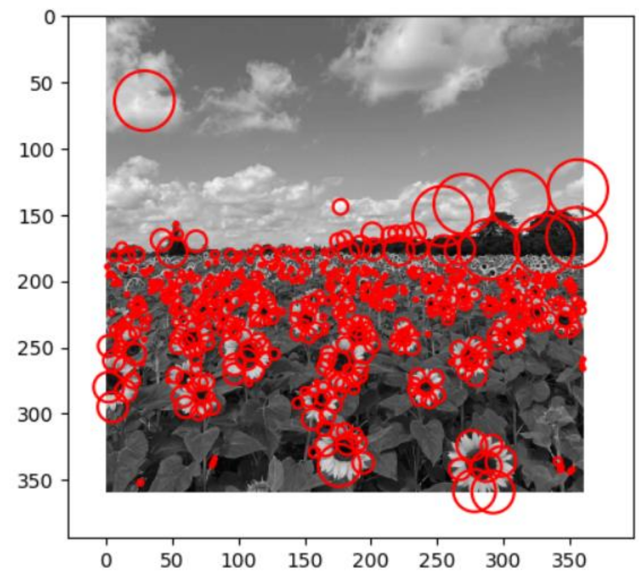
The blobs can be found by looping over the scale space and finding the local extremums.

```
for i in range(1,h):
    for j in range(1,w):
        slice_img = log_image_np[:,i-1:i+2,j-1:j+2] #9*3*3 slice
        result = np.amax(slice_img) #finding maximum
        if result >= 0.03: #threshold
            z,x,y = np.unravel_index(slice_img.argmax(),slice_img.shape)
            co_ordinates.append((i+x-1,j+y-1,k**z*sigma)) #finding co-rdinates
return co_ordinates
```

For the sake of clarity, overlapping blobs were removed by the following code:

```
#determining whether two blobs overlaps by finding the area between them
def blob_overlap(blob1, blob2):
    n_dim = len(blob1) - 1
    root_ndim = sqrt(n_dim)

    # radius of two blobs
    r1 = blob1[-1] * root_ndim
    r2 = blob2[-1] * root_ndim

    #finding the distance between their two centers
    d = sqrt(np.sum((blob1[:-1] - blob2[:-1])**2))

    #no overlap between two blobs
    if d > r1 + r2:
        return 0
    # one blob is inside the other, the smaller blob must die
    elif d <= abs(r1 - r2):
        return 1
    else:
        #computing the area of overlap between blobs
        ratio1 = (d ** 2 + r1 ** 2 - r2 ** 2) / (2 * d * r1)
        ratio1 = np.clip(ratio1, -1, 1)
        acos1 = math.acos(ratio1)
        ratio2 = (d ** 2 + r2 ** 2 - r1 ** 2) / (2 * d * r2)
        ratio2 = np.clip(ratio2, -1, 1)
        acos2 = math.acos(ratio2)
        a = -d + r2 + r1
        b = d - r2 + r1
        c = d + r2 - r1
        d = d + r2 + r1
        area = (r1 ** 2 * acos1 + r2 ** 2 * acos2 -0.5 * sqrt(abs(a * b * c * d)))
        return area/(math.pi * (min(r1, r2) ** 2))
```

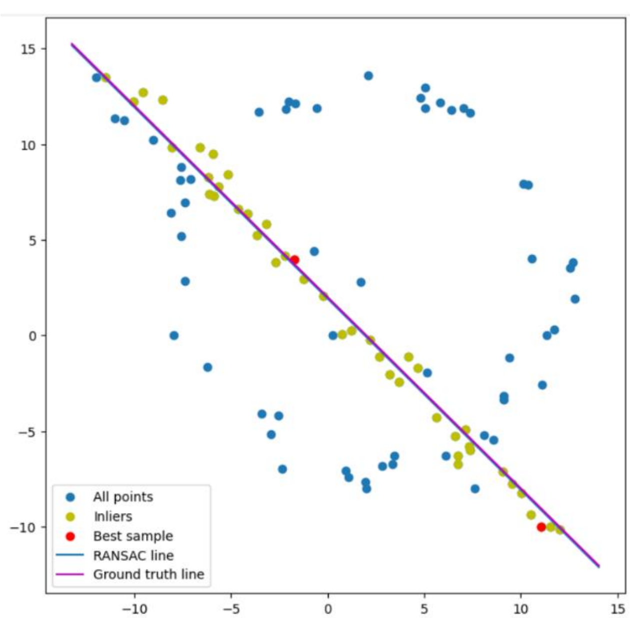
T



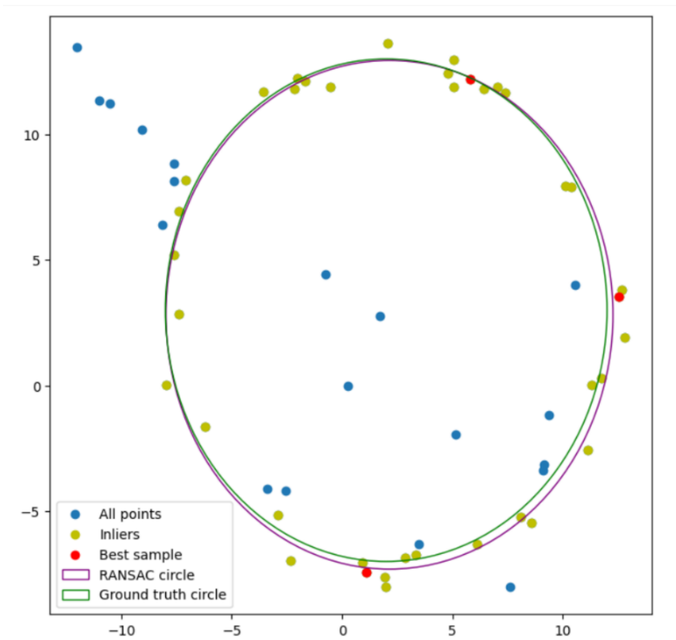
2 a) Estimating the line using the RANSAC method. Number of points in the consensus was picked to be 0.4*(total number of points).

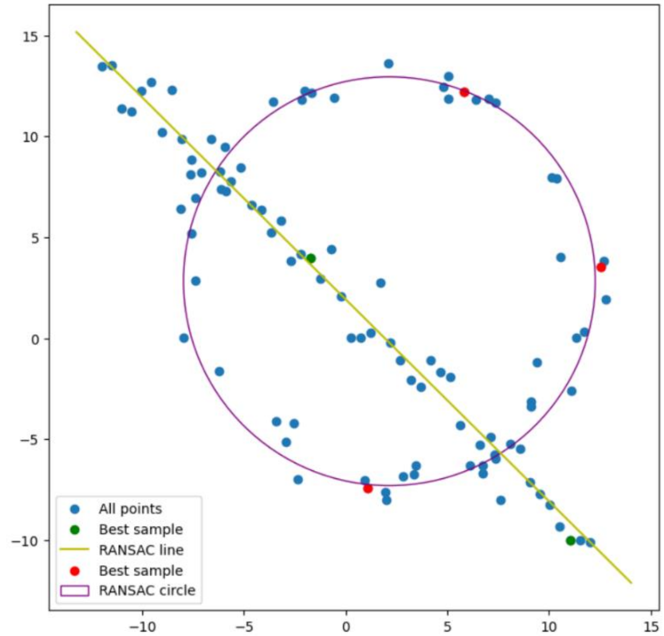
```
while iteration < max_iterations:
    indices = np.random.randint(0, N, s) # A sample of two (s) points selected at random
    x0 = np.array([1, 1, 0]) # Initial estimate
    res = minimize(fun = line_tls, args = indices, x0 = x0, tol= 1e-6, constraints=cons)
    inliers_line = consensus_line(X_, res.x, t) # Computing the inliers
    if inliers_line.sum() > d:
        x0 = res.x
        # Computing the new model using the inliers
        res = minimize(fun = line_tls, args = inliers_line, x0 = x0, tol= 1e-6, constraints=cons)
        print(res.x, res.fun)
        if res.fun < best_error:
            print('A better model found ... ', res.x, res.fun)
            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
```

The line was that was finally drawn is as follows:



2 c) And finally the circle and line both in the same graph.



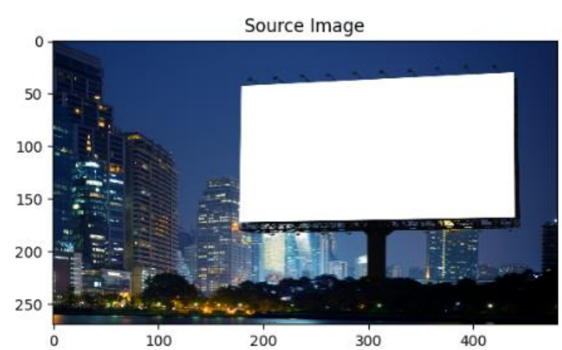


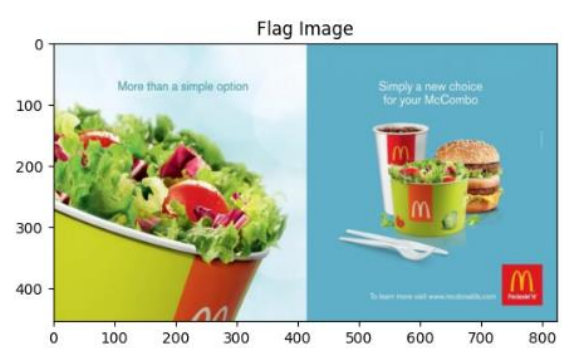
2 d) If we try to fit the circle first, due to higher number of degrees of freedom of the circle, a circle with an incredibly large radius might be fitted in which the dataset will only fit a small portion of its circumference.

3) Superposition can be carried out using the following functions.

```
#Finding the homography
H, _= cv.findHomography(p, p_flag)

# Warping image of flag
warped_img = cv.warpPerspective(flag, np.linalg.inv(H), (bground.shape[1],bground.shape[0]))
```

The source image, the overlapped image and the final image can be shown as follows.





4) The functions used for the question were sift_match, homography, RANSAC_homography and dist respectively. The code for their implementation is given below.

```python
def sift_match(im1, im2):
    '''used to sift match two images. We can adjust the good match percentage required to our liking'''

    GOOD_MATCH_PERCENT = 0.8

    # Detect sift features
    sift = cv.SIFT_create(nOctaveLayers = 3, contrastThreshold = 0.1, edgeThreshold = 25, sigma = 1)
    keypoint_1, descriptors_1 = sift.detectAndCompute(im1,None)
    keypoint_2, descriptors_2 = sift.detectAndCompute(im2,None)

    # Match features.
    matcher = cv.BFMatcher()
    matches = matcher.knnMatch(descriptors_1, descriptors_2, k = 2)

    # Filter good matches using ratio test in Lowe's paper
    good_matches, points1, points2 = [], [], []

    for a,b in matches:
        if a.distance < GOOD_MATCH_PERCENT*b.distance:
            good_matches.append(a)
            points1.append(keypoint_1[a.queryIdx].pt)
            points2.append(keypoint_2[a.trainIdx].pt)

    good_matches, points1, points2 = np.array(good_matches), np.array(points1), np.array(points2)
```

```python
def homography(pts1, pts2):
    '''determining the homography between the two images'''
    A = []
    for i in range(len(pts1)):
        x1, y1, x2, y2 = pts1[i][0], pts1[i][1], pts2[i][0], pts2[i][1]
        A.append([-x1, -y1, -1, 0, 0, 0, x2*x1, x2*y1, x2])
        A.append([0, 0, 0, -x1, -y1, -1, y2*x1, y2*y1, y2])

    A = np.matrix(A)
    U, S, V = np.linalg.svd(A)
    H = np.reshape(V[-1], (3, 3))
    H = (1/H.item(8))*H
    return H
```

```python
def RANSAC_homography(points1, points2):
    inlier_count, selected_inliers = 0, None

    points = np.hstack((points1, points2))
    num_iterations = int(np.log(1 - 0.95)/np.log(1 - (1 - 0.5)**4))
    threshold = 100


    for i in range(100):
        samples1 = []
        samples2 = []
        for k in range(4):
            idx = np.random.randint(0, len(points1))
            samples1.append(points1[idx])
            samples2.append(points2[idx])


        H = homography(samples1, samples2)

        inliers1, inliers2 = [], []

        for j in range(len(points1)):
            distance = dist(points1[j], points2[j], H)
            if distance < 5:
                inliers1.append(points1[j])
                inliers2.append(points2[j])


        if len(inliers1) > threshold:
            max_inliers1 = inliers1
            max_inliers2 = inliers2
            H = homography(max_inliers1, max_inliers2)

    return H
```

```python
def dist(P1, P2, H):
    '''determing the distance between two points belonging to the two images'''
    p1 = np.transpose(np.matrix([P1[0], P1[1], 1]))
    estimatep2 = np.dot(H, p1)
    estimatep2 = (1/estimatep2.item(2))*estimatep2

    p2 = np.transpose(np.matrix([P2[0], P2[1], 1]))
    error = p2 - estimatep2
    return np.linalg.norm(error)
```

4 a) The matched features of the two images is as follows.

4 b) The computed homography is as follows:

```
[[-3.88187650e-01 -1.25070136e+00  3.79707045e+02]
 [-5.46530342e-01 -1.73649073e+00  5.34220004e+02]
 [-1.00077993e-03 -3.35899949e-03  1.00000000e+00]]
```

4 c) The stitched image is as follows.

img5.ppm

Stitched Image