Assignment 02- Fitting and Alignment-200148M-GitHub

Question-01

```
# Load the image
im = cv.imread('the_berry_farms_sunflower_field.jpeg',
cv.IMREAD_REDUCED_COLOR_4)
# Apply Gaussian blur to reduce noise
blurred = cv.GaussianBlur(im, (9, 9), 0.1)
# Convert the image to grayscale
gray = cv.cvtColor(blurred,
cv.COLOR_BGR2GRAY)
# Define parameters for blob detection
min_sigma = 3
max_sigma = 30
threshold = .1
# Detect blobs using Laplacian of Gaussians
blobs = blob_log(gray, min_sigma=min_sigma,
max_sigma=max_sigma, threshold=threshold)
blobs[:, 2] = blobs[:, 2] * sqrt(2)
max_radius_index = np.argmax(blobs[:, 2])
largest_circle_params =
blobs[max_radius_index]
y, x, r = largest_circle_params
print(f"Radius (r): {r}")
print(f"Center (x, y): ({x}, {y})")
# Draw circles on the original color image
for blob in blobs:
  y, x, r = blob  # Blob format is (y, x, r)
  cv.circle(im, (int(x), int(y)), int(r), (0, 0, 255), 2)
```

Figure 1

parameters of the largest circles.

- Radius (r): 42.42640687
- Center (x, y): (234.0, 0.0)

Question-02

Figure 2 Noisy Point Set Generation

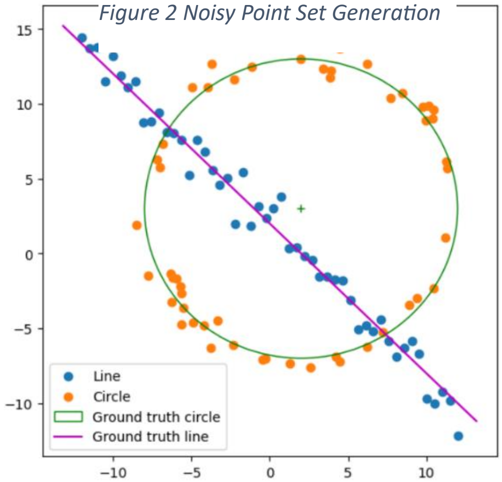
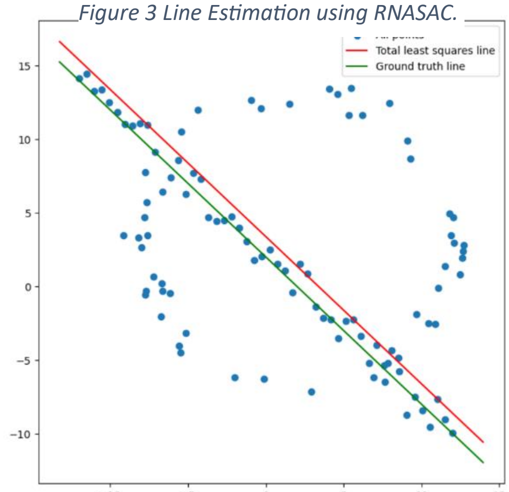
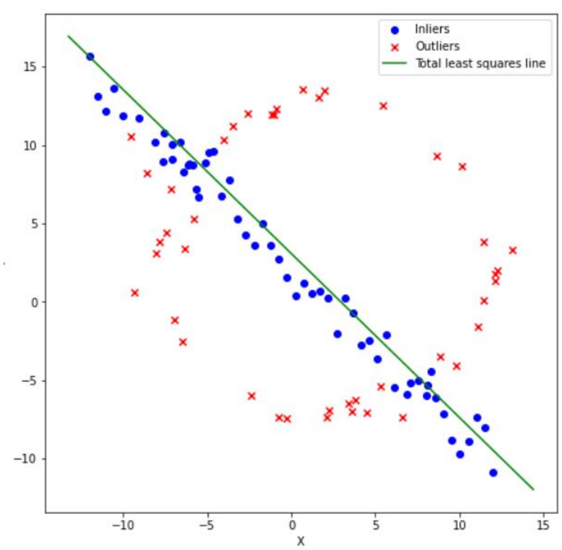
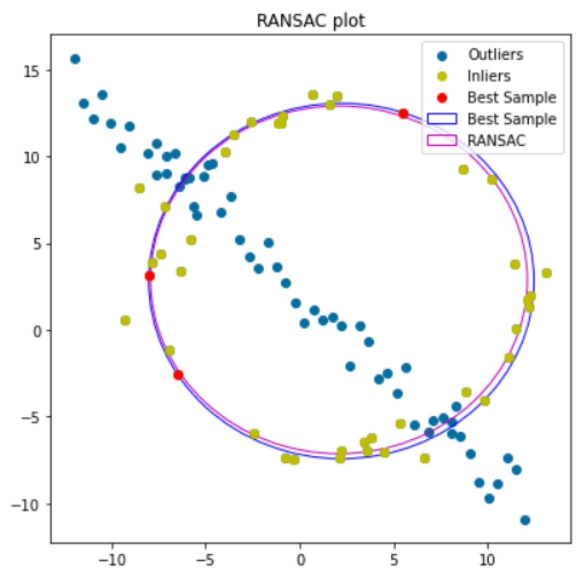


Figure 3 Line Estimation using RNASAC.



```python
def circle_equation(points):
    """ Return the center and radius of the circle from three points """
    p1,p2,p3 = points[0], points[1], points[2]
    temp = p2[0] * p2[0] + p2[1] * p2[1]
    bc = (p1[0] * p1[0] + p1[1] * p1[1] - temp) / 2
    cd = (temp - p3[0] * p3[0] - p3[1] * p3[1]) / 2
    det = (p1[0] - p2[0]) * (p2[1] - p3[1]) - (p2[0] - p3[0]) * (p1[1] - p2[1])

    # Center of circle
    cx = (bc*(p2[1] - p3[1]) - cd*(p1[1] - p2[1])) / det
    cy = ((p1[0] - p2[0]) * cd - (p2[0] - p3[0]) * bc) / det

    radius = np.sqrt((cx - p1[0])**2 + (cy - p1[1])**2)
    return ((cx, cy), radius)

def get_inliers(data_list, center, r):
    """ Returns the list of inliers to a model of a circle from a set of points. The threshold 
value is taken as 1/5th of the radius """
    inliers = []
    thresh = r//3

    for i in range(len(data_list)):
        error = np.sqrt((data_list[i][0]-center[0])**2 + (data_list[i][1]-center[1])**2) - r
        if error < thresh:
            inliers.append(data_list[i])

    return np.array(inliers)

def random_sample(data_list):
    """ Returns a list of 3 random samples from a given list """
    sample_list = []
    random.seed(0)
    rand_nums = random.sample(range(1, len(data_list)), 3)
    for i in rand_nums:
        sample_list.append(data_list[i])
    return np.array(sample_list)
```

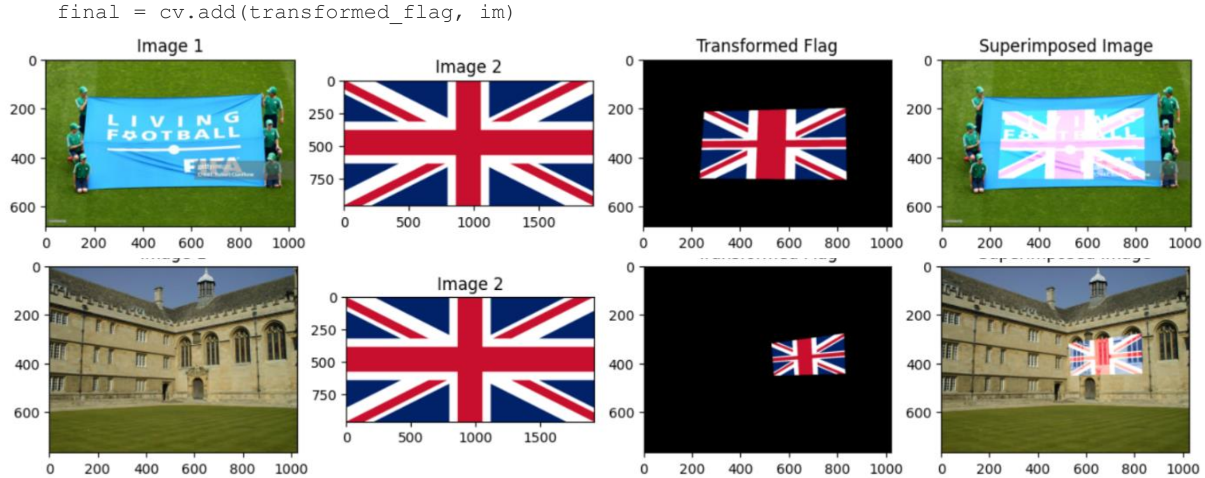




Part(d) - Focusing on fitting the circle first may lead to a less accurate line fit because it leaves fewer data points for detecting and modeling the line where it intersects with the circle.

Question-03

```
for i in range(2):
    im = cv.imread(ims[i])
    cv.imshow("Wadham College",im)
    corners = []
    cv.setMouseCallback("Wadham College", mouse_click), cv.waitKey(0), cv.destroyAllWindows()

    h, w = np.shape(im)[0], np.shape(im)[1]
    zero_matrix = np.array([[0], [0], [0]])

    x1, y1, x2, y2, x3, y3, x4, y4 = corners[0][0], corners[0][1], corners[1][0], corners[1][1],
corners[2][0], corners[2][1], corners[3][0], corners[3][1]

    flag_im = cv.imread("Flag_of_the_United_Kingdom.png")
    fh, fw , ch= flag_im.shape
    f1, f2, f3, f4 = np.array([[0, 0, 1]]), np.array([[fw-1, 0, 1]]), np.array([[fw-1, fh-1,
1]]), np.array([[0, fh-1, 1]])
 matrix_A = np.concatenate((np.concatenate((zero_matrix.T,f1, -y1*f1), axis = 1),
np.concatenate((f1, zero_matrix.T, -x1*f1), axis = 1),
                                np.concatenate((zero_matrix.T,f2, -y2*f2), axis = 1),
np.concatenate((f2, zero_matrix.T, -x2*f2), axis = 1),
                                np.concatenate((zero_matrix.T,f3, -y3*f3), axis = 1),
np.concatenate((f3, zero_matrix.T, -x3*f3), axis = 1),
                                np.concatenate((zero_matrix.T,f4, -y4*f4), axis = 1),
np.concatenate((f4, zero_matrix.T, -x4*f4), 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))
    transformed_flag = cv.warpPerspective(flag_im, H, (w, h))
    final = cv.add(transformed_flag, im)
```



Question-04

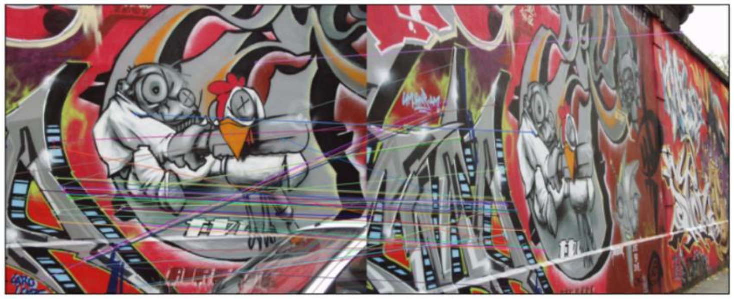


Figure 2 ***Question 4(a)***

Part(b)

```python
def homography(pts1, pts2):
    mean1, mean2 = np.mean(pts1, axis=0), np.mean(pts2, axis=0)
    s1, s2 = len(pts1)*np.sqrt(2)/np.sum(np.sqrt(np.sum((pts1-mean1)**2, axis=1))), len(pts1)*np.sqrt(2)/np.sum(np.sqrt(np.sum((pts2-mean2)**2, axis=1)))
    tx1, ty1, tx2, ty2 = -s1*mean1[0], -s1*mean1[1], -s2*mean2[0], -s2*mean2[1]
    T1, T2 = np.array(((s1, 0, tx1), (0, s1, ty1), (0, 0, 1))), np.array(((s2, 0, tx2), (0, s2, ty2), (0, 0, 1)))
    A = []

    for i in range(len(pts1)):
        X11, X21 = T1 * np.concatenate((pts1[i], [1])).reshape(3, 1), T2 * np.concatenate((pts2[i], [1])).reshape(3, 1)
        A.append((-X11[0][0], -X11[1][0], -1, 0, 0, 0, X21[0][0]*X11[0][0], X21[0][0]*X11[1][0], X21[0][0]))
        A.append((0, 0, 0, -X11[0][0], -X11[1][0], -1, X21[1][0]*X11[0][0], X21[1][0]*X11[1][0], X21[1][0]))

    A = np.array(A)
    U, S, V = np.linalg.svd(A, full_matrices=True)
    h = np.reshape(V[-1], (3, 3))
    H = linalg.inv(T2) * h * T1
    H = (1 / H.item(8)) * H
    return H
```

```
[[-2.97283674e-01 -4.17396920e-01  3.57204005e+02]
 [-3.57169400e-01  1.69792767e-01  2.10563639e+02]
 [-1.15364931e-03 -6.40276575e-04  1.00000000e+00]]
```

Figure 3 Computed homography

Part(c)