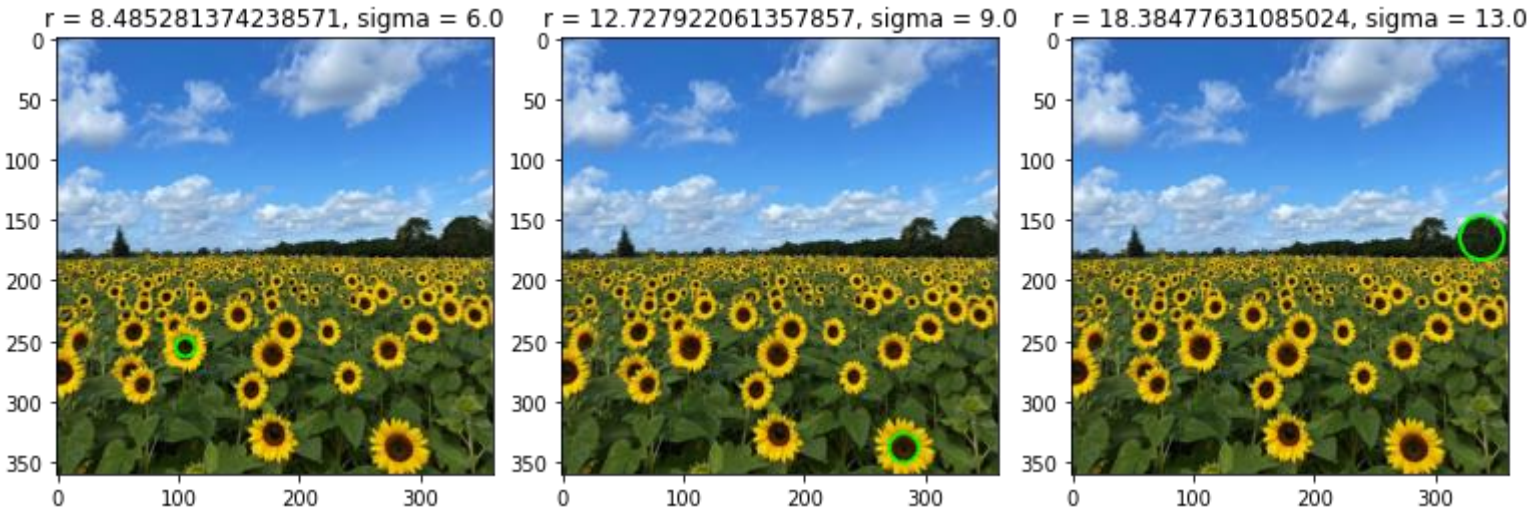
EN3160 Assignment 2 on Fitting and Alignment

200041E – Anuradha A.K.

Question 1

In question 1, we need to develop blob detection on the sunflower field image using Laplacian of Gaussian and scale-space extrema detection. To obtain blobs we need to check the largest blobs in various sigma ranges. When sigma increases, we can see the blob size is also increased. The following code snippet shows how to get the blobs in high-intensity areas.

```
im_gray = cv.cvtColor(im, cv.COLOR_RGB2GRAY)
K = np.arange(5,16,0.5)
for k in K:
    im_ = im.copy()
    scale_space = np.empty((im.shape[0] ,im.shape[1] , 500), dtype=np.float64)
    sigmas = np.arange(k,k+0.5, 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(im_gray, 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)

    #draw a circle around the detected blob
    cv.circle(im_, (int(indices[1]), int(indices[0])), int(r), (0,255,0), 2)
```



In the provided code, I utilized various sigma ranges to detect the largest blobs within a specific region. I divided the overall range into smaller segments, specifically within the [5:16] range, and then further subdivided it into 0.5 by 0.5 increments. Within each of these subranges, I assessed the presence of blobs by incrementing the sigma value by 0.1. After evaluating the region using the specified sigma values, the code selects the largest blob within that region.

Question 2

a) In the first part of the question, we need to approximate a line from the data points using the RANSAC algorithm. First, the code selects two random points to form a line and calculates the error between this line and the other data points. Then, a threshold error value is defined to separate the inliers from the data points. The line with the maximum number of inliers is considered the best-approximated line.

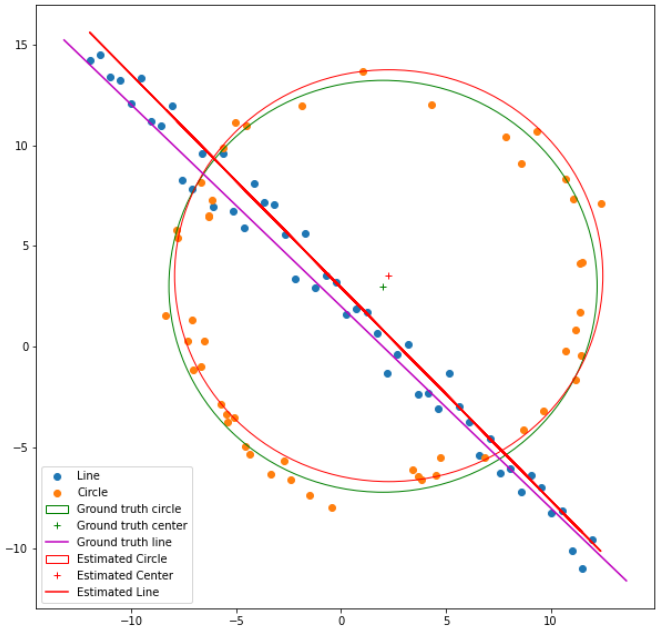
b) In the next part of the question, we aim to approximate a circle by randomly selecting three points. I then set a threshold of 15 for inlier selection, requiring a minimum of 10 inliers to fit the model.

d) When we fit the circle first it also randomly chooses the points that belong to the line. Then RANSAC approximates a circle with a very large radius to fit the three points belonging to line data points. We are also hard to understand it is a circle because that circle with a very large radius is locally a line. Since we need to approximate the line first and then circle.

Here is the code for approximating the circle to the data points.

```
for _ in range(max_iterations):
    # Randomly select three points
    random_indices = np.random.choice(len(X_circ), 3, replace=False)
    random_points = X_circ[random_indices]

    # Fit a circle to the three random points
    xc, yc, r = fit_circle(random_points)

    # Calculate the distance between the estimated circle and all data points
    distances = np.sqrt((X_circ[:, 0] - xc)**2 + (X_circ[:, 1] - yc)**2)

    # Count inliers (points that are within the threshold)
    inliers = np.sum(distances < inlier_threshold)

    if inliers >= min_inliers and inliers > best_inliers_circle:
        best_circle = (xc, yc, r)
        best_inliers_circle = inliers
```

Here, the fit_circle function takes three data points and outputs the center of the circle and the radius of the circle.



Question 3

In question 3, we superimpose one image onto another image. To accomplish this task, we define a function using EVENT_LBUTTONDOWN from the OpenCV library to capture four points by clicking the left mouse key. We then create a homography transformation between the two images and generate the final composite image using that transformation.

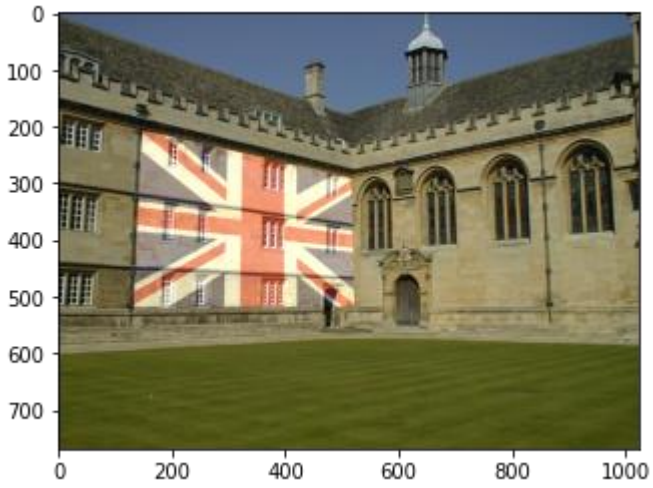
Here is the function to record the points using the mouse click event,

```
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) # Draw a red circle at the
clicked point (x, y)
            cv.imshow('Image', image_1)
            if len(coordinates) == 4:
                cv.destroyAllWindows()
```

then we define another function for making the superimposed image using homography,

```
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]

    image_2_warped = cv.warpPerspective(image_2, homography_matrix, (image_1.shape[1],
image_1.shape[0]))
    blended_image = cv.addWeighted(image_1, alpha, image_2_warped, beta, 0)

    return blended_image
```

the output of the code,



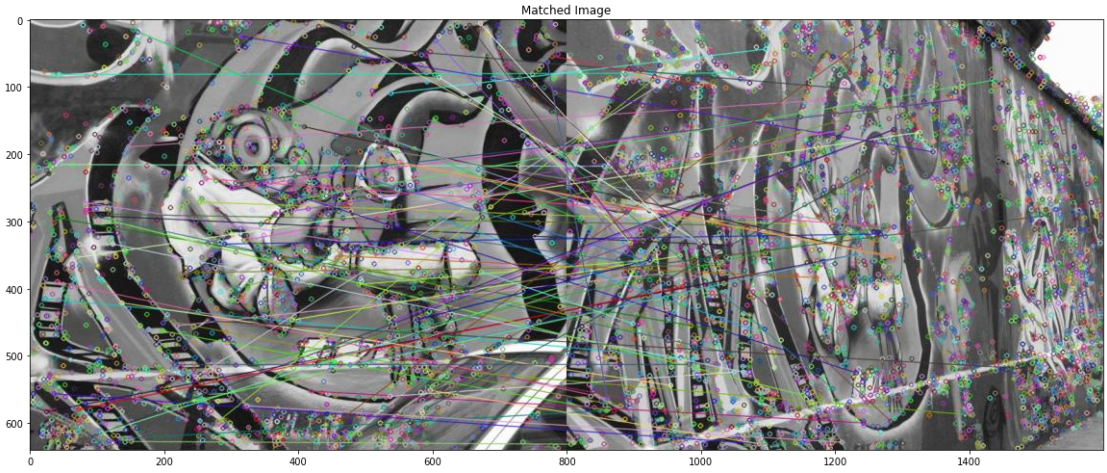
Question 4

In the first part of the question, the code needs to find the SIFT features of the two images. To accomplish that task we can use $sift.detectAndCompute$ function to extract the features. Then we can draw the key points on the image by using $drawKeypoints$ in the OpenCV library. This is the output of the first part of the code,

```
# Find the keypoints and descriptors with SIFT
keypoints1, descriptors1 = sift.detectAndCompute(gray1, None)
```

Using the brute Force Matcher in the OpenCV library we can match the same features between the two images.

```
bf = cv.BFMatcher()

# Match descriptors between the two images
matches = bf.knnMatch(descriptors1, descriptors5, k=2)
```



After computing the homography using the SIFT features, we can get this output,

```
Computed Homography:
[[-4.88959310e-01 -2.21662832e+00  5.30506378e+02]
 [ 1.57998254e-01 -1.40780616e+00  9.41269576e+01]
 [-6.00322235e-04 -4.93287400e-03  1.00000000e+00]]

Known 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]]
```