Question 01 [Git hub repo]() **200500L**

```python
for i in range(5, 12, 1):
    #fig, ax = plt.subplots(2, s, figsize=(20,5))
    scale_space = np.empty((image.shape[0], image.shape[1], 100),dtype=np.float32)
    sigmas = np.arange(i, i+1, 0.01)
    for i, sigma in enumerate(sigmas):
        log_hw = 3*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(gray_image, -1, log)
        scale_space[:, :, i] = f_log
```

```
Parameters of the largest circle:
Center (x, y): (612, 428)
Radius: 7.0710678118654755
Parameters of the largest circle:
Center (x, y): (538, 422)
Radius: 8.485281374238571
Parameters of the largest circle:
Center (x, y): (448, 485)
Radius: 9.899494936611665
Parameters of the largest circle:
Center (x, y): (606, 478)
Radius: 11.313708498984761
Parameters of the largest circle:
Center (x, y): (379, 481)
Radius: 12.727922061357857
Parameters of the largest circle:
Center (x, y): (105, 345)
Radius: 14.142135623730951
Parameters of the largest circle:
Center (x, y): (106, 346)
Radius: 15.556349186104047
```

In this code, a Laplacian of Gaussians (LOG) filter is generated and applied to an input image with varying sigma values within the range of 5 to 12, with a step size of 0.01. The objective is to identify local maxima extrema for different sigma values and subsequently detect and plot the blobs in the image for the specified sigma range

Maximum radius – 15.55 at (106,346)

Question 02

Using given code generated a noisy points along with

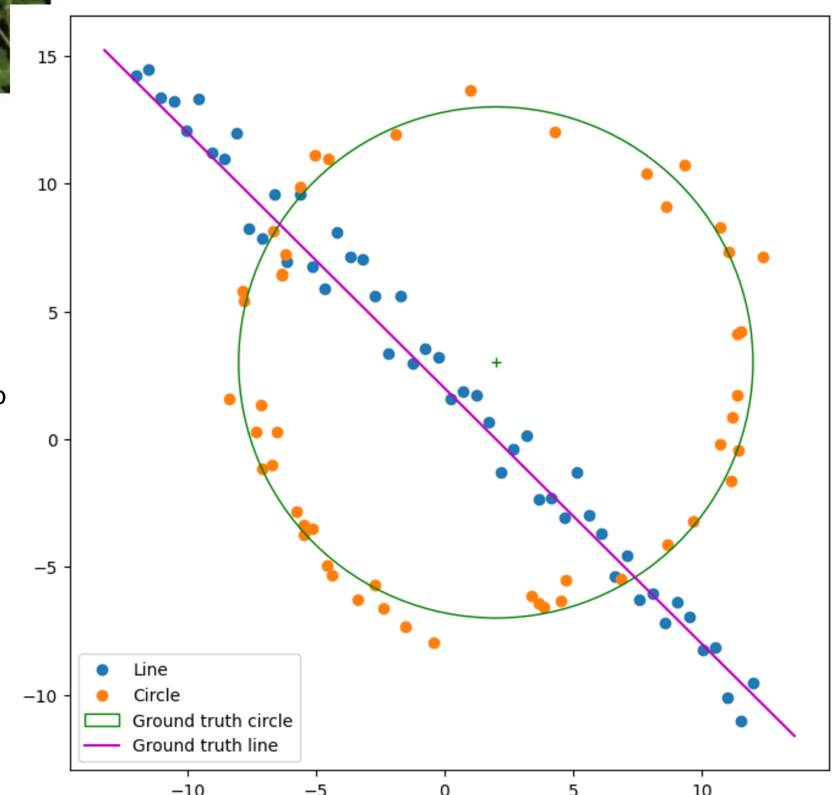
the line and the circle.

In this approach I used a function to find a line for given two

Points and find the distances from each data point to the

Line. Based on that distances I decided the particular data

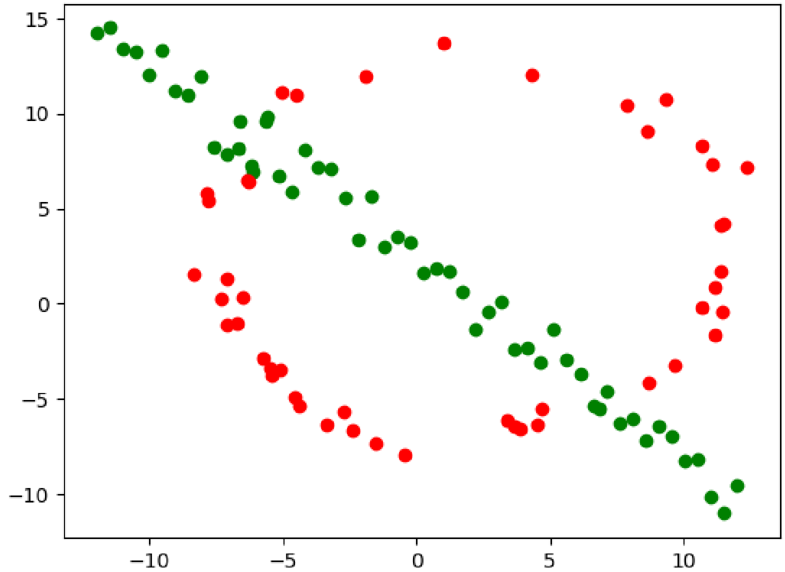
point is an outlier or not.



ransac line function

```python
def ransac_line(x1,y1):
    # find line parameters
    m = (y1[1] - y1[0])/(x1[1] - x1[0])
    c = y1[0] - m*x1[0]
    a = -m
    b = 1
    #normalise
    d1 = np.sqrt(a**2 + b**2)
    a = a/d1
    b = b/d1
    d = np.abs(a*x_all+ b*y_all - d1)/np.sqrt(a**2 + b**2)
    return a, b, d,c
```

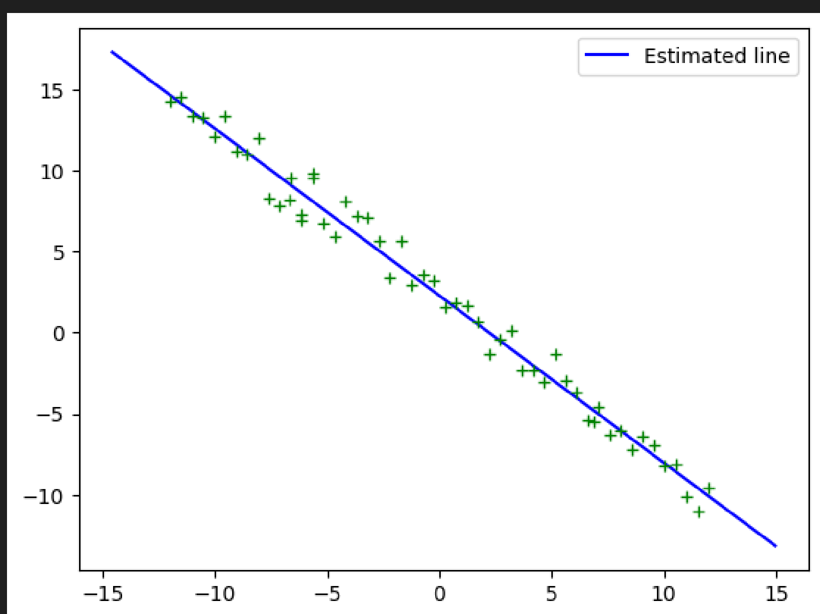
from given data points identified 54 inliers



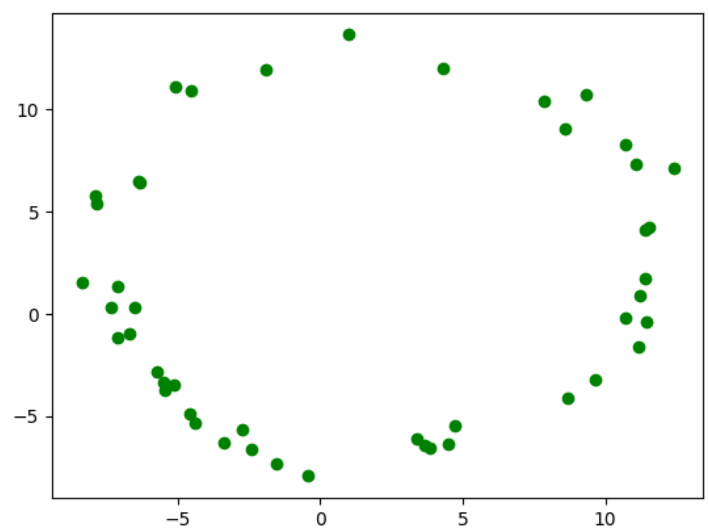
Best fitting line(used ransc algorithm for 1000 iteration)

Used 1.7 as the threshold

```
m, c: -1.0308850964382035 2.271133961749208
```



Outliers



Similar approach can be used for find best fitting circle for rest of the data points (after ransac line fitting).

```python
def ransac_circle(x1, y1):
    # Find circle parameters
    xa, xb, xc = x1
    ya, yb, yc = y1

    A = np.array([[2*xa, 2*ya, 1], [2*xb, 2*yb, 1], [2*xc, 2*yc, 1]])
    b = np.array([xa**2 + ya**2, xb**2 + yb**2, xc**2 + yc**2])


    inv_A = np.linalg.inv(A)
    x0 = np.dot(inv_A[0], b)
    y0 = np.dot(inv_A[1], b)

    r = np.sqrt((x0 - xa)**2 + (y0 - ya)**2)

    d = np.abs(np.sqrt((x_all - x0)**2 + (y_all - y0)**2) - r)
    return x0, y0, r, d
```

Chose randomly 3 points and analyzed each data

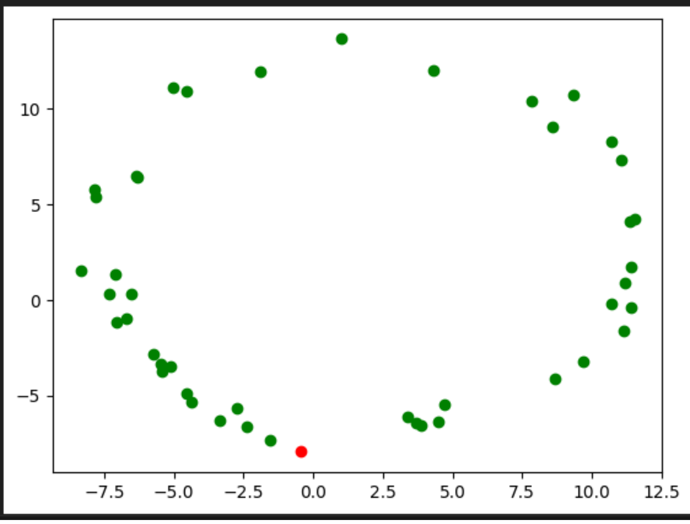
Points based on the distance from the center of

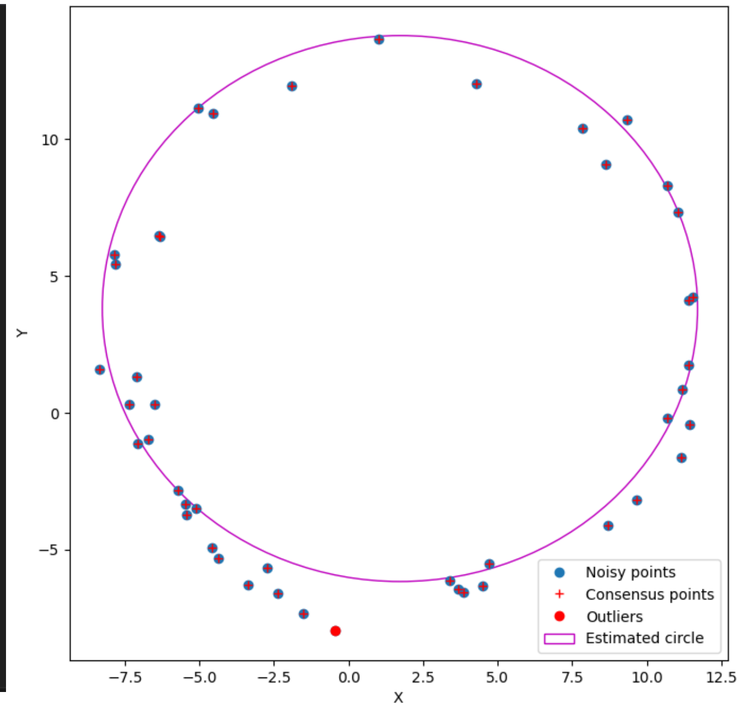
Predicted circle.

44 inliers were detected

Used 1.7 as threshold

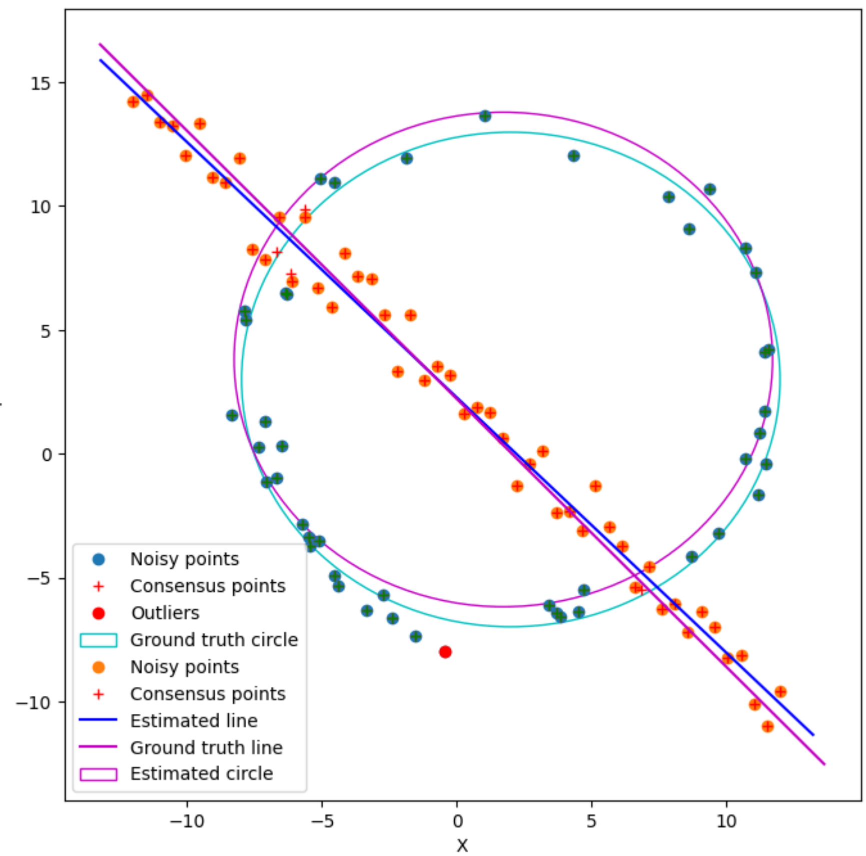
```
xs: [-5.73249084 -6.30877504 -6.34152069] ys: [-2.82575284  6.43627445  6.48394589]
inliers: 8
outliers: 37
xs: [11.05786244 -5.05274181 -5.73249084] ys: [ 7.33454976 11.13161333 -2.82575284]
inliers: 44
outliers: 1
```






```
estimated circle parameters: center (x, y): (1.7235513812991181, 3.8063599804050092) radius: 9.978851942008907
```

```
plot all points and the estimated circle with the ground truth
circle and line with the ground truth line
```



When we first fit the circle before the line and then perform the line fitting there are some differences in the generated line because of the affected outliers are different.

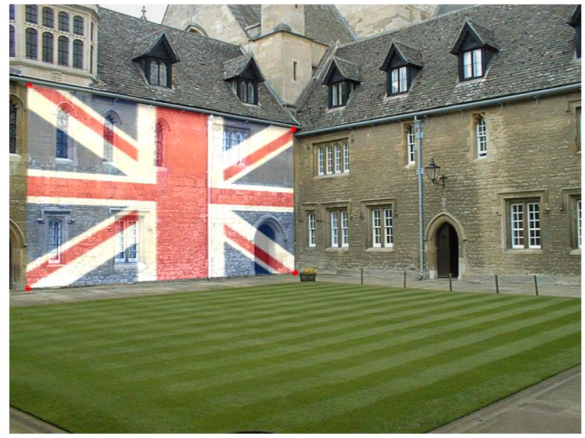
Also the circle fitting algorithm is little bit slower, since it has many points to consider than previous method

Question 03

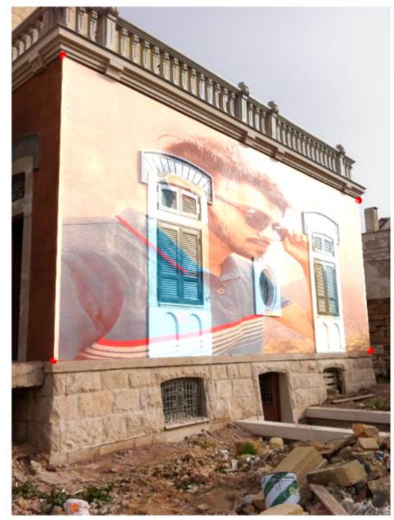
In this task, the objective is to overlay one image onto another. Specifically, the scenario involves combining an architectural image with a flag image. The initial step is to select four points on the architectural image that correspond to the four corners of the flag image. To achieve this, the code utilizes OpenCV, enabling the user to click on the architectural image to obtain coordinates for these four points.

Then I took coordinates from the architectural image and used them to make a special map called a homography matrix. This map helped me move the flag image to fit perfectly onto the architectural image. It's like making a puzzle piece fit just right. Afterward, I combined the flag image with the architectural image to create a new picture where the flag looks like it's part of the building.

```python
def select_corresponding_points(image_architectural, image_flag):
    # Find points on the architectural image that correspond to the flag image
    corresponding_points = []

    def click_event(event, x, y, flags, param):
        nonlocal corresponding_points
        if event == cv.EVENT_LBUTTONDOWN:
            corresponding_points.append((x, y))
            cv.circle(image_architectural, (x, y), 5, (0, 0, 255), -1)
            cv.imshow('Architectural Image', image_architectural)
            if len(corresponding_points) == 4:
                cv.destroyAllWindows()

    cv.imshow('Architectural Image',image_architectural)
    cv.setMouseCallback('Architectural Image', click_event)

    while len(corresponding_points) < 4:
        cv.waitKey(1)

    points_architectural = np.array(corresponding_points, dtype=np.float32)

    width_flag = image_flag.shape[1]
    height_flag = image_flag.shape[0]
    points_flag = np.array([[0, 0], [width_flag, 0], [width_flag, height_flag], [0, height_flag]], dtype=np.float32)

    return points_architectural, points_flag
```



```python
def superimpose_images_with_homography(architectural_image, flag_image, points_architectural, points_flag, alpha, beta):
    # Calculate the homography matrix
    homography_matrix = cv.findHomography(points_flag, points_architectural)[0]

    # Warp the flag image to fit the architectural image
    flag_image_warped = cv.warpPerspective(flag_image, homography_matrix, (architectural_image.shape[1], architectural_image.shape[0]))

    # Blend the two images
    blended_image = cv.addWeighted(architectural_image, alpha, flag_image_warped, beta, 0)

    return blended_image
```



Question 04

Employed a SIFT detector to identify key features in both images and generate descriptions for these features. then utilized a FlannBasedMatcher to compare these features and select only the strong matches.

```python
# Apply Lowe's ratio test to select good matches
good_matches = []
for match1, match2 in matches:
    if match1.distance < 0.75 * match2.distance:
        good_matches.append(match1)
```

```python
# Compute the homography using my own code within RANSAC
print("number of good_matches",len(good_matches))
```

```
✓ 0.0s

number of good_matches 87
```

```
final homography [[-1.20540962e-01 -3.33150818e-01  1.13459649e+02]
 [-3.72551868e-01 -9.66907229e-01  3.43500660e+02]
 [-1.08363963e-03 -2.74031830e-03  1.00000000e+00]]
```


```python
for i in range(iterations):
    # Randomly select 4 correspondences (matches) as a sample from good_matches
    sample = random.sample(good_matches, 4)
    #Extract the keypoints' coordinates for these samples from both img1 and img5
    img1_pts = np.float32([keypoints1[m.queryIdx].pt for m in sample])
    img5_pts = np.float32([keypoints5[m.trainIdx].pt for m in sample])

    # Compute the homography using the 4 correspondences using RANSAC, based on the sampled correspondences.
    H = cv.findHomography(img1_pts, img5_pts,cv.RANSAC, threshold)

    # Apply the computed homography to all keypoints in img1
    transformed_img1_pts = cv.perspectiveTransform(np.float32([keypoints1[m.queryIdx].pt for m in good_matches]).reshape(-1, 1, 2), H[0])

    # count inliers and brake if suitable homography is found
    mid_inliers = []
    for i in range(len(good_matches)):
        if np.linalg.norm(transformed_img1_pts[i] - np.float32([keypoints5[m.trainIdx].pt for m in good_matches])[i]) < threshold:
            mid_inliers.append(good_matches[i])

    if len(mid_inliers) > len(inliers):
        inliers = mid_inliers
        homograpy = H[0]

    #break if suitable homography is found
    if len(inliers) > len(good_matches) * 0.9:
        break
```