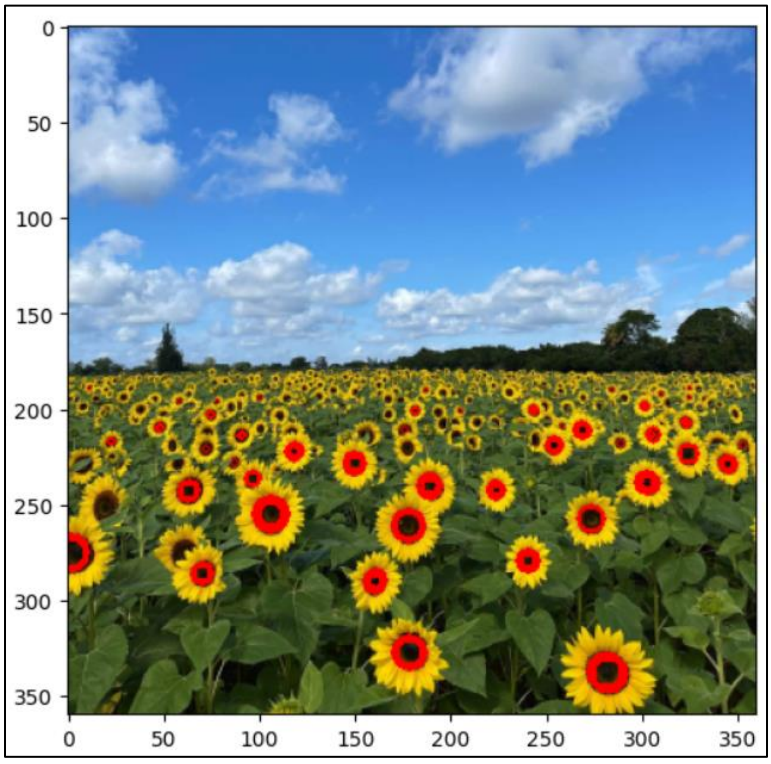
R.M.S.Madhushan
Index : 200363R

Q1)

```python
threshold = 80

sigma_values = np.arange(1, 25, 1)
scale_space = np.zeros((h, w, len(sigma_values)))

for i, sigma in enumerate(sigma_values):

    log_hw = 3 * sigma
    X, Y = np.meshgrid(np.arange(-log_hw, log_hw + 1), np.arange(-log_hw, log_hw + 1))
    X = X.astype(np.float32)
    Y = Y.astype(np.float32)

    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_gray, -1, log)
    scale_space[:, :, i] = np.absolute(f_log)


for i in range(h) :
    for j in range(w) :
        maximum = np.max(scale_space[i,j,:])
        if threshold < maximum :
            ind = np.argmax(scale_space[i,j,:])
            sigma = sigma_values[ind]
            radius = int((2**0.5)*sigma)
            cv.circle(image, (j,i), radius, [255,0,0],1)
```

The parameters of the largest circle are:
(207, 323)
sigma value correspond to the maximum response : 2
radius : 2



R.M.S.Madhushan
Index : 200363R

Q2) (a)Line Fitting using RANSAC

```python
def line_equation_from_points(x1, y1, x2, y2):
    del_x = x2 - x1
    del_y = y2 - y1
    mag = math.sqrt(del_x**2 + del_y**2)
    a,b,c = (del_y / mag), (-del_x / mag), ((a * x1) + (b * y1))
    return a, b, d

def line_tls(x, indices):
    a, b, d = x[0], x[1], x[2]
    return np.sum(np.square(a*X_[indices,0] + b*X_[indices,1] - d))

def g(x):
    return x[0]**2 + x[1]**2 - 1

cons = ({'type': 'eq', 'fun': g})

def consensus_line(X_, x, th):
    a, b, d = x[0], x[1], x[2]
    return np.absolute(a*X_line[:,0] + b*X_line[:,1] - d) < th

th, dis  = 1 , 0.84*N

inliers_line = []
iteration = 500
best_model_line = []
best_error = np.inf
best_sample_line = []
best_inliers_line = []
x0 = np.array([1, 1, 0])
```

```python
for i in range(iteration):
    indices = np.random.randint(0, N, 2)

    res = minimize(fun = line_tls, args = indices, x0 = x0, tol= 1e-6, constraints=cons, options={'disp': True})
    inliers_line = consensus_line(X_line, res.x, th)

    if np.sum(inliers_line) > dis:
        x0 = res.x
        print('no of inliners = ',np.sum(inliers_line), d)
        res = minimize(fun = line_tls, args = inliers_line, x0 = x0, tol= 1e-6, constraints=cons, options={'disp': True})

        if res.fun < best_error:
            best_error = res.fun
            (variable) best_inliers_line: Any
            best_inliers_line = inliers_line


print('Best model : ', best_model_line)
```

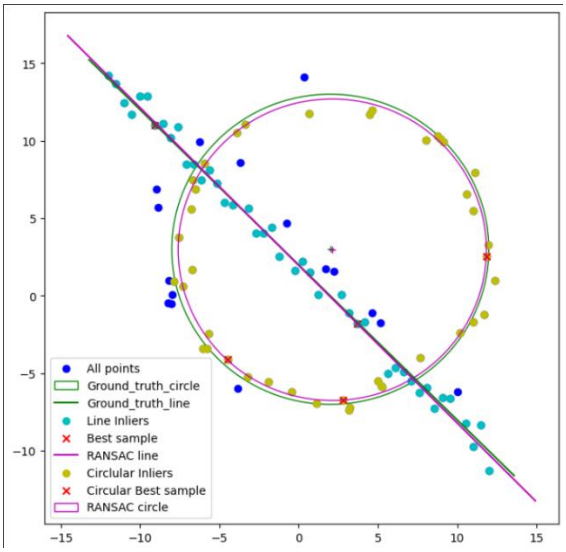
(b)Circle Fitting using RANSAC

```python
other_points = X_line[~best_inliers_line]
circler_X = np.vstack((X_circ, other_points))

def consensus_circle(data, model,t):
    center_x, center_y, radius = model
    distances = np.sqrt((data[:, 0] - center_x)**2 + (data[:, 1] - center_y)**2)
    return np.abs(distances - radius) < t

def circle_tls(x, indices):
    x_c, y_c, r = x[0], x[1], x[2]
    data = circler_X[indices]
    error = np.sum((data - np.array([x_c, y_c]))**2, axis=1)
    error = np.abs(error - r**2)
    return np.sum(error)

th = 1.0
dis = 0.8*N

cons_circle = ({'type': 'ineq', 'fun': lambda x: x[2] - t})

circle_inliers = []
iteration = 500
best_error = np.inf
best_sample_circle = []
best_model_circle = []
res_only_with_sample_circle = []
best_circle_inliers = []
x0 = np.array([1, 1, 1])
```

```python
for i in range(iteration) :

    indices = np.random.randint(0, N, 3)
    res = minimize(fun = circle_tls, args = indices, x0 = x0, tol= 1e-6, constraints=cons_circle, options={'disp': True})
    circle_inliers = consensus_circle(circler_X, res.x, t)

    if np.sum(circle_inliers) > dis:
        x_c = res.x
        # print('no of inliners for circle: ', np.sum(circle_inliers), d)

        res = minimize(fun = circle_tls, args = circle_inliers, x0 = x0, tol= 1e-6, constraints=cons_circle, options={'disp': True})

        if res.fun < best_error:
            best_model_circle = res.x
            best_error = res.fun
            best_sample_circle = circler_X[indices,:]
            best_circle_inliers = circle_inliers


print('best model : ', best_model_circle, ', no of inliners = ', np.sum(best_circle_inliers))
```

R.M.S.Madhushan
Index : 200363R

(c) Plotting Line and Circle

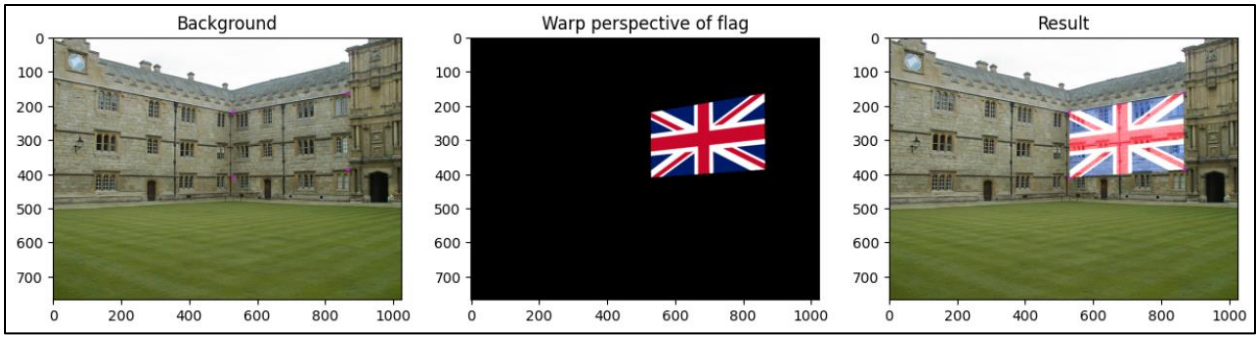


d) When circle is first modeling using RANSAC, It can be taken the line inlier points as inliers to the circle and can predict a circular model which has a large radius. Then when the line model is predicting, due to the lack of inliers points, line prediction could be inaccurate. Therefore it is better to prioritize the dominance model when fitting.

Q3)

```
corners_bg = []
corners_flag = []

def mouse_callback(event, x,y, flags, param) :
    global corners_bg
    key = cv.waitKey(1) & 0xFF

    if (event == cv.EVENT_LBUTTONDOWN) :
        corners_bg.append((x,y))        # corner selecting pattern : left-upper -> right-upper -> right-bottom -> left-bottom
        cv.circle(image3_bg, (x,y), 3, (255,0,255),2)
        cv.imshow("Image", image3_bg)

    if (len(corners_bg) == 4) :
        cv.destroyAllWindows()


cv.namedWindow("Image")
cv.imshow("Image", image3_bg)
cv.setMouseCallback("Image", mouse_callback)
cv.waitKey(0)


corners_flag = [(0,0), (w,0), (w,h), (0,h)]
```

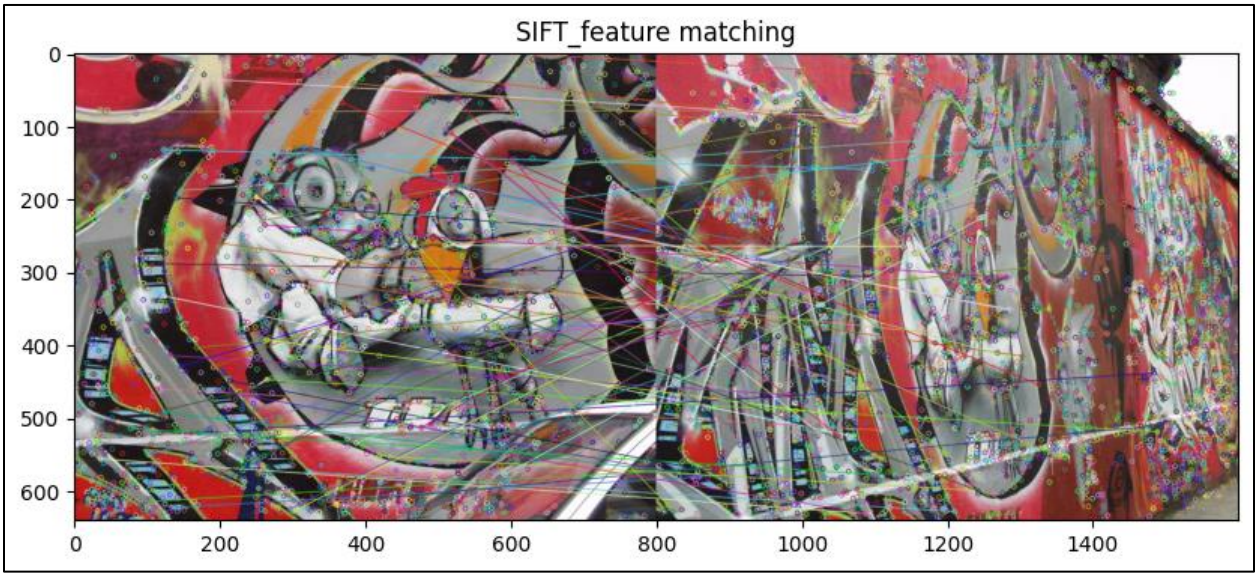
```
corners_bg = np.array(corners_bg, dtype=np.float32)
corners_flag = np.array(corners_flag, dtype=np.float32)
homography,_  = cv.findHomography(corners_flag, corners_bg, method=cv.RANSAC)

warp = cv.warpPerspective(image3_flag, homography, (wb,hb))
warp = cv.cvtColor(warp, cv.COLOR_BGR2RGB)

alpha = 0.1
cvt_image3_bg = cv.cvtColor(image3_bg, cv.COLOR_BGR2RGB)
blended_image = cv.addWeighted(cvt_image3_bg, 1, warp, 1-alpha, 0)
```



R.M.S.Madhushan
Index : 200363R

Q4)

```python
image4_1 = cv.imread("graf/img1.jpg")
image4_5 = cv.imread("graf/img5.jpg")

gray_1 = cv.cvtColor(image4_1, cv.COLOR_BGR2GRAY)
gray_5 = cv.cvtColor(image4_5, cv.COLOR_BGR2GRAY)

sift = cv.SIFT_create()

keypoints1, des1 = sift.detectAndCompute(gray_1, None)
keypoints5, des5 = sift.detectAndCompute(gray_5, None)

bf = cv.BFMatcher()
matches = bf.knnMatch(des1, des5, k=2)

th_matches = []

for m,n in matches :
    if (m.distance < 0.75*n.distance) :
        th_matches.append(m)

matched = cv.drawMatches(gray_1, keypoints1, gray_5, keypoints5, th_matches,None)

matched = cv.cvtColor(matched, cv.COLOR_BGR2RGB)
```



Github link : https://github.com/SadeepRathnayaka/EN3160-Image-Processing-and-Machine-Vision/tree/main/Assignment%202