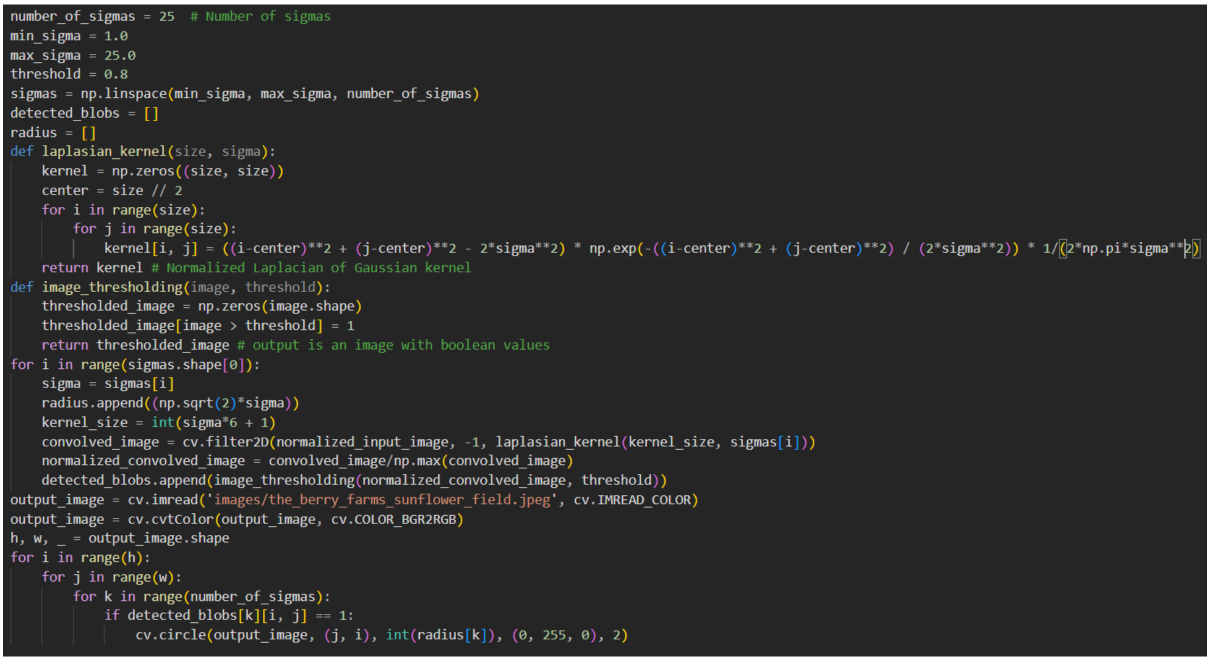
EN 3160 Image processing a Computer Vision Assignment 2

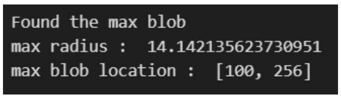
Index no: 200356A

Question 1

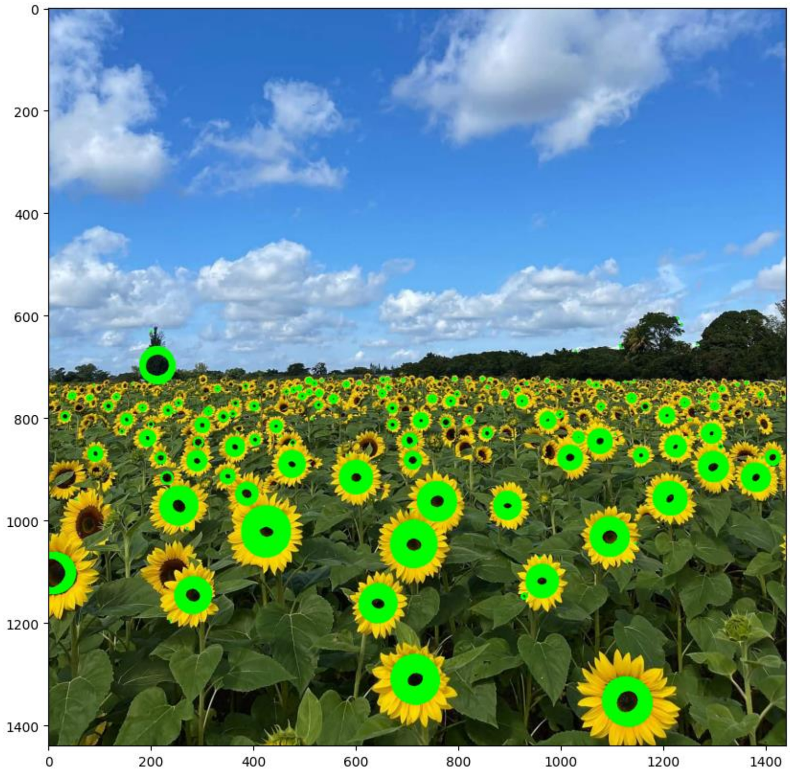


```
number_of_sigmas = 25  # Number of sigmas
min_sigma = 1.0
max_sigma = 25.0
threshold = 0.8
sigmas = np.linspace(min_sigma, max_sigma, number_of_sigmas)
detected_blobs = []
radius = []
def laplasian_kernel(size, sigma):
    kernel = np.zeros((size, size))
    center = size // 2
    for i in range(size):
        for j in range(size):
            kernel[i, j] = ((i-center)**2 + (j-center)**2 - 2*sigma**2) * np.exp(-((i-center)**2 + (j-center)**2) / (2*sigma**2)) * 1/(2*np.pi*sigma**2)
    return kernel # Normalized Laplacian of Gaussian kernel
def image_thresholding(image, threshold):
    thresholded_image = np.zeros(image.shape)
    thresholded_image[image > threshold] = 1
    return thresholded_image # output is an image with boolean values
for i in range(sigmas.shape[0]):
    sigma = sigmas[i]
    radius.append((np.sqrt(2)*sigma))
    kernel_size = int(sigma*6 + 1)
    convolved_image = cv.filter2D(normalized_input_image, -1, laplasian_kernel(kernel_size, sigmas[i]))
    normalized_convolved_image = convolved_image/np.max(convolved_image)
    detected_blobs.append(image_thresholding(normalized_convolved_image, threshold))
output_image = cv.imread('images/the_berry_farms_sunflower_field.jpeg', cv.IMREAD_COLOR)
output_image = cv.cvtColor(output_image, cv.COLOR_BGR2RGB)
h, w, _ = output_image.shape
for i in range(h):
    for j in range(w):
        for k in range(number_of_sigmas):
            if detected_blobs[k][i, j] == 1:
                cv.circle(output_image, (j, i), int(radius[k]), (0, 255, 0), 2)
```

Location and Radius of max blob



```
Found the max blob
max radius :  14.142135623730951
max blob location :  [100, 256]
```



Question 2

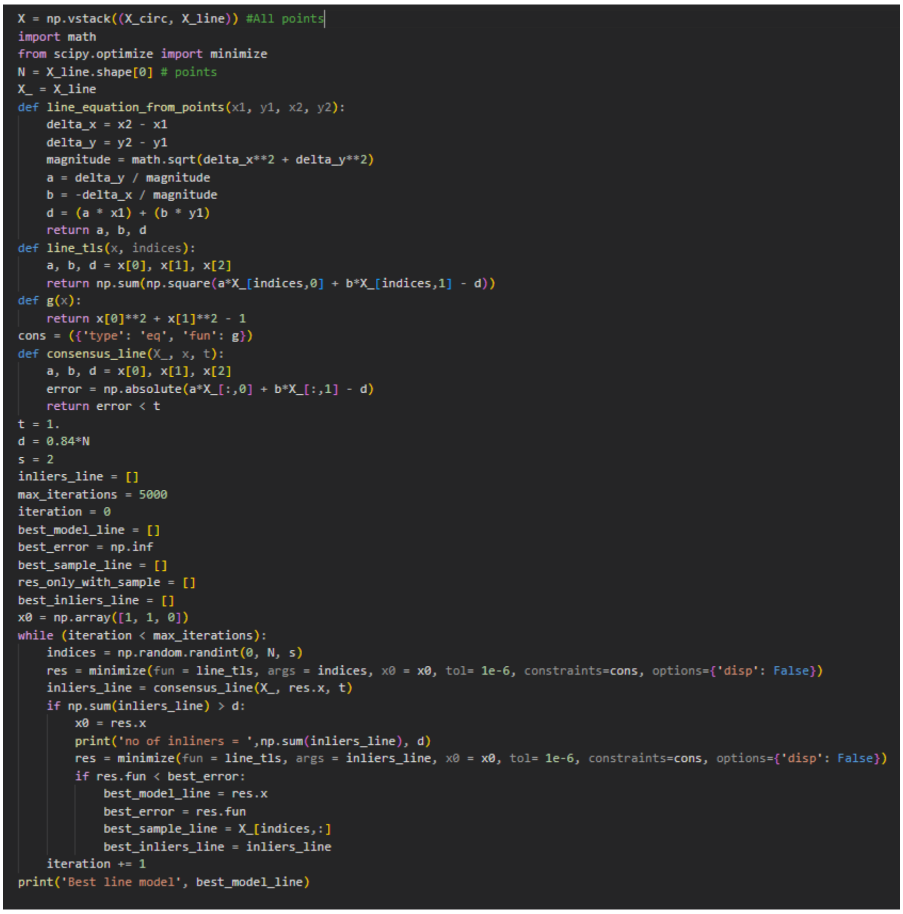
Best RANSAC Line a, b, d, and inliners values are

best model found: [2.1893539 3.16277678 9.85943152] , no of inliners = 46

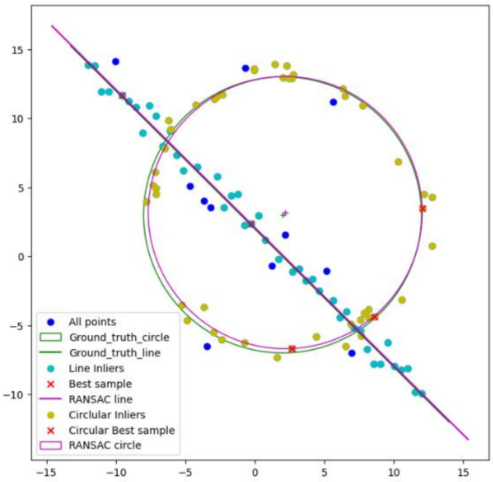
Best RANSAC Circle x center, y Center, and radius are

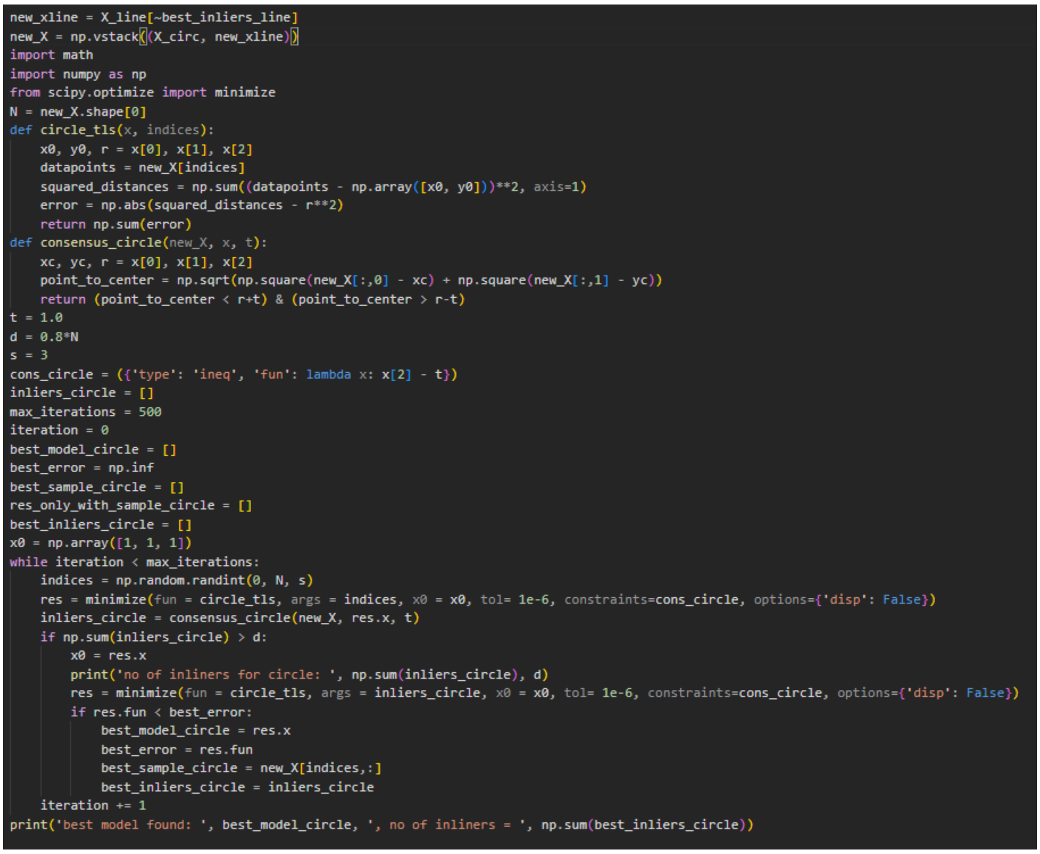
RANSAC circle: [2.1893539 3.16277678 9.85943152]

If we tried to find the circle first, circle might be very large to fit the points of the line to the circumference. Therefore, more convenient method is to find the RANSAC line first and then take the RANSAC circle.



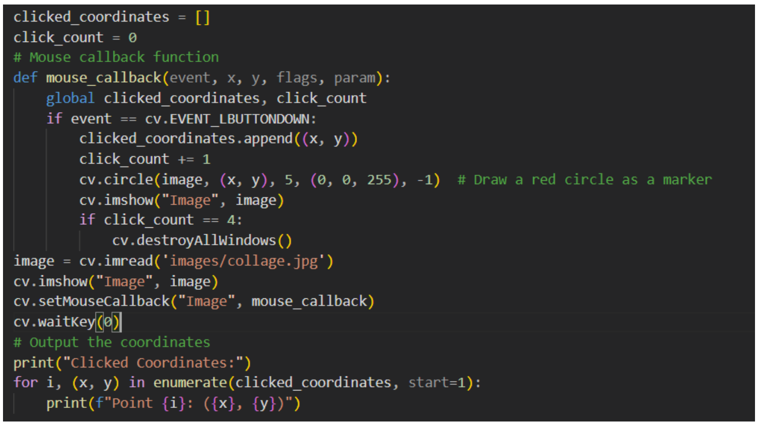
```
X = np.vstack((X_circ, X_line)) #All points
import math
from scipy.optimize import minimize
N = X_line.shape[0] # points
X_ = X_line
def line_equation_from_points(x1, y1, x2, y2):
    delta_x = x2 - x1
    delta_y = y2 - y1
    magnitude = math.sqrt(delta_x**2 + delta_y**2)
    a = delta_y / magnitude
    b = -delta_x / magnitude
    d = (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, t):
    a, b, d = x[0], x[1], x[2]
    error = np.absolute(a*X_[:,0] + b*X_[:,1] - d)
    return error < t
t = 1.
d = 0.84*N
s = 2
inliers_line = []
max_iterations = 5000
iteration = 0
best_model_line = []
best_error = np.inf
best_sample_line = []
res_only_with_sample = []
best_inliers_line = []
x0 = np.array([1, 1, 0])
while (iteration < max_iterations):
    indices = np.random.randint(0, N, s)
    res = minimize(fun = line_tls, args = indices, x0 = x0, tol= 1e-6, constraints=cons, options={'disp': False})
    inliers_line = consensus_line(X_, res.x, t)
    if np.sum(inliers_line) > d:
        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': False})
        if res.fun < best_error:
            best_model_line = res.x
            best_error = res.fun
            best_sample_line = X_[indices,:]
            best_inliers_line = inliers_line
    iteration += 1
print('Best line model', best_model_line)
```



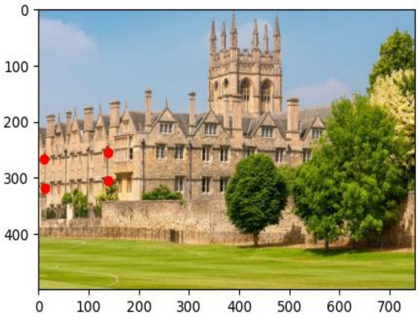


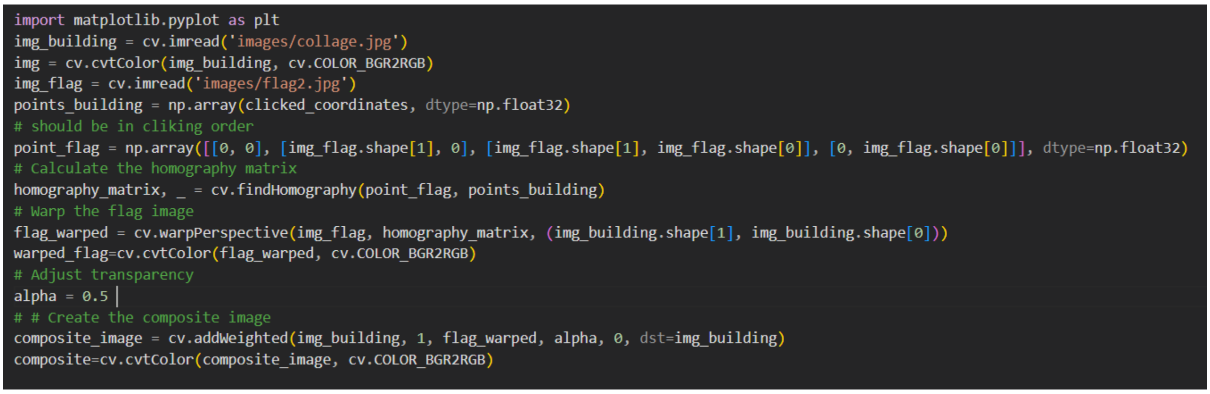
```python
new_xline = X_line[~best_inliers_line]
new_X = np.vstack((X_circ, new_xline))
import math
import numpy as np
from scipy.optimize import minimize
N = new_X.shape[0]
def circle_tls(x, indices):
    x0, y0, r = x[0], x[1], x[2]
    datapoints = new_X[indices]
    squared_distances = np.sum((datapoints - np.array([x0, y0]))**2, axis=1)
    error = np.abs(squared_distances - r**2)
    return np.sum(error)
def consensus_circle(new_X, x, t):
    xc, yc, r = x[0], x[1], x[2]
    point_to_center = np.sqrt(np.square(new_X[:,0] - xc) + np.square(new_X[:,1] - yc))
    return (point_to_center < r+t) & (point_to_center > r-t)
t = 1.0
d = 0.8*N
s = 3
cons_circle = ({'type': 'ineq', 'fun': lambda x: x[2] - t})
inliers_circle = []
max_iterations = 500
iteration = 0
best_model_circle = []
best_error = np.inf
best_sample_circle = []
res_only_with_sample_circle = []
best_inliers_circle = []
x0 = np.array([1, 1, 1])
while iteration < max_iterations:
    indices = np.random.randint(0, N, s)
    res = minimize(fun = circle_tls, args = indices, x0 = x0, tol= 1e-6, constraints=cons_circle, options={'disp': False})
    inliers_circle = consensus_circle(new_X, res.x, t)
    if np.sum(inliers_circle) > d:
        x0 = res.x
        print('no of inliners for circle: ', np.sum(inliers_circle), d)
        res = minimize(fun = circle_tls, args = inliers_circle, x0 = x0, tol= 1e-6, constraints=cons_circle, options={'disp': False})
        if res.fun < best_error:
            best_model_circle = res.x
            best_error = res.fun
            best_sample_circle = new_X[indices,:]
            best_inliers_circle = inliers_circle
    iteration += 1
print('best model found: ', best_model_circle, ', no of inliners = ', np.sum(best_inliers_circle))
```

Question 3

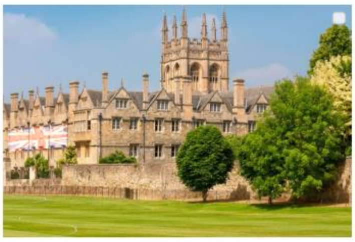


```python
clicked_coordinates = []
click_count = 0
# Mouse callback function
def mouse_callback(event, x, y, flags, param):
    global clicked_coordinates, click_count
    if event == cv.EVENT_LBUTTONDOWN:
        clicked_coordinates.append((x, y))
        click_count += 1
        cv.circle(image, (x, y), 5, (0, 0, 255), -1)  # Draw a red circle as a marker
        cv.imshow("Image", image)
        if click_count == 4:
            cv.destroyAllWindows()
image = cv.imread('images/collage.jpg')
cv.imshow("Image", image)
cv.setMouseCallback("Image", mouse_callback)
cv.waitKey(0)
# Output the coordinates
print("Clicked Coordinates:")
for i, (x, y) in enumerate(clicked_coordinates, start=1):
    print(f"Point {i}: ({x}, {y})")
```

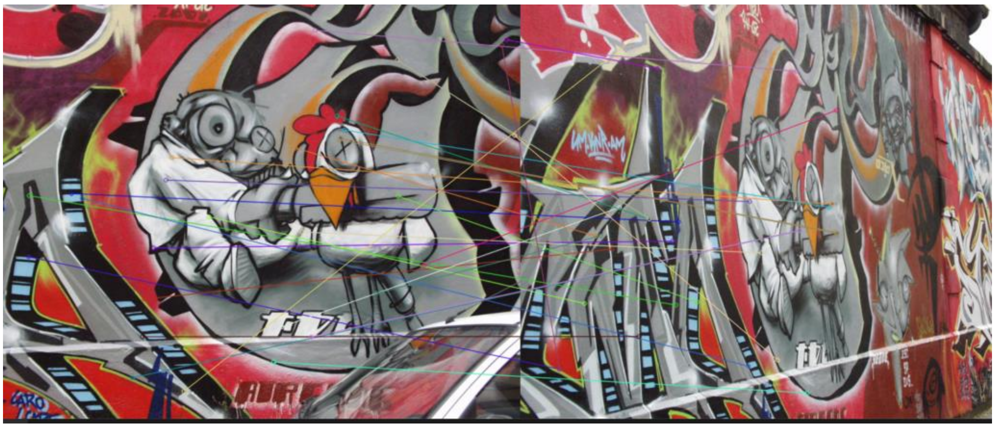


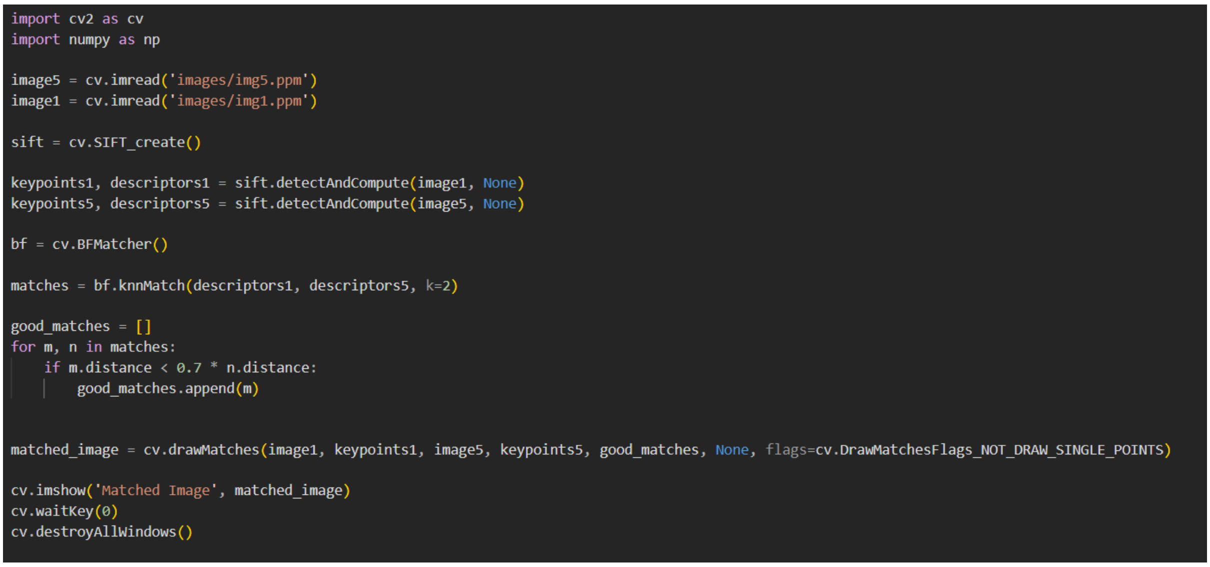


```python
import matplotlib.pyplot as plt
img_building = cv.imread('images/collage.jpg')
img = cv.cvtColor(img_building, cv.COLOR_BGR2RGB)
img_flag = cv.imread('images/flag2.jpg')
points_building = np.array(clicked_coordinates, dtype=np.float32)
# should be in cliking order
point_flag = np.array([[0, 0], [img_flag.shape[1], 0], [img_flag.shape[1], img_flag.shape[0]], [0, img_flag.shape[0]]], dtype=np.float32)
# Calculate the homography matrix
homography_matrix, _ = cv.findHomography(point_flag, points_building)
# Warp the flag image
flag_warped = cv.warpPerspective(img_flag, homography_matrix, (img_building.shape[1], img_building.shape[0]))
warped_flag=cv.cvtColor(flag_warped, cv.COLOR_BGR2RGB)
# Adjust transparency
alpha = 0.5
# # Create the composite image
composite_image = cv.addWeighted(img_building, 1, flag_warped, alpha, 0, dst=img_building)
composite=cv.cvtColor(composite_image, cv.COLOR_BGR2RGB)
```



Question 04





```
import cv2 as cv
import numpy as np

image5 = cv.imread('images/img5.ppm')
image1 = cv.imread('images/img1.ppm')

sift = cv.SIFT_create()

keypoints1, descriptors1 = sift.detectAndCompute(image1, None)
keypoints5, descriptors5 = sift.detectAndCompute(image5, None)

bf = cv.BFMatcher()

matches = bf.knnMatch(descriptors1, descriptors5, k=2)

good_matches = []
for m, n in matches:
    if m.distance < 0.7 * n.distance:
        good_matches.append(m)


matched_image = cv.drawMatches(image1, keypoints1, image5, keypoints5, good_matches, None, flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

cv.imshow('Matched Image', matched_image)
cv.waitKey(0)
cv.destroyAllWindows()
```

Github link - https://github.com/Dulan24/S5_EN3061_Image-processing/tree/master/Activity%2002