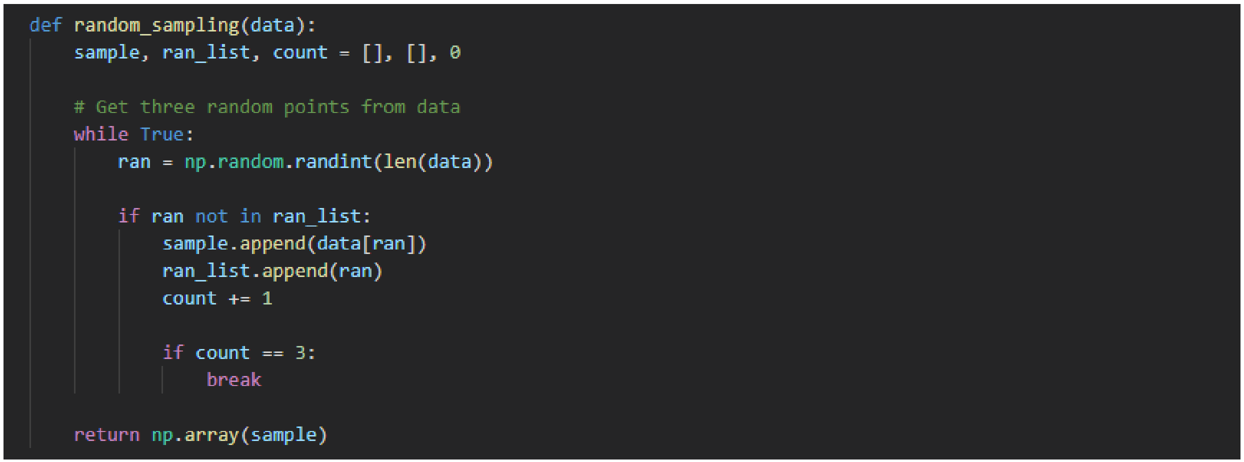
RANATUNGA R.G.S.M.
190504H

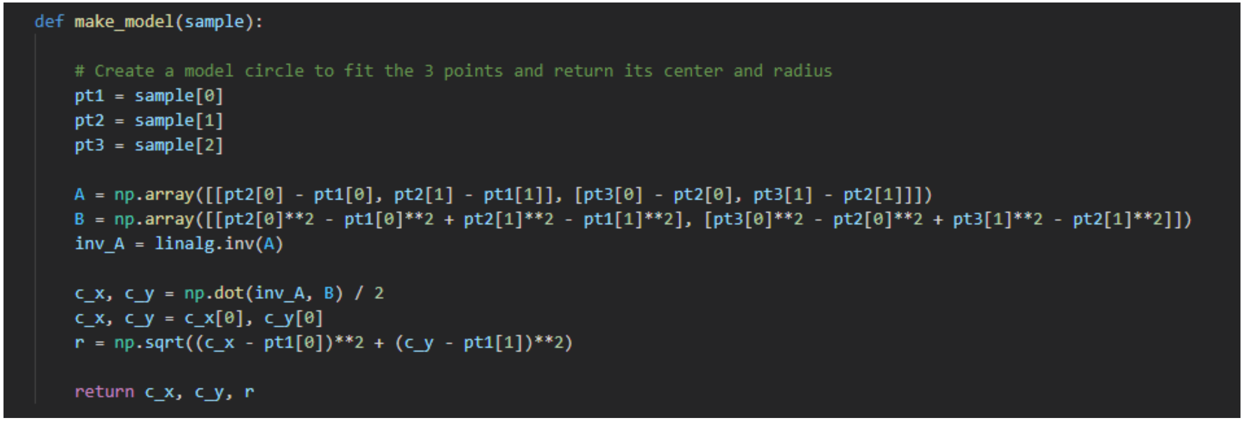
EN2550 - Fundamentals of Image Processing and Machine Vision
Assignment 2

Question 1

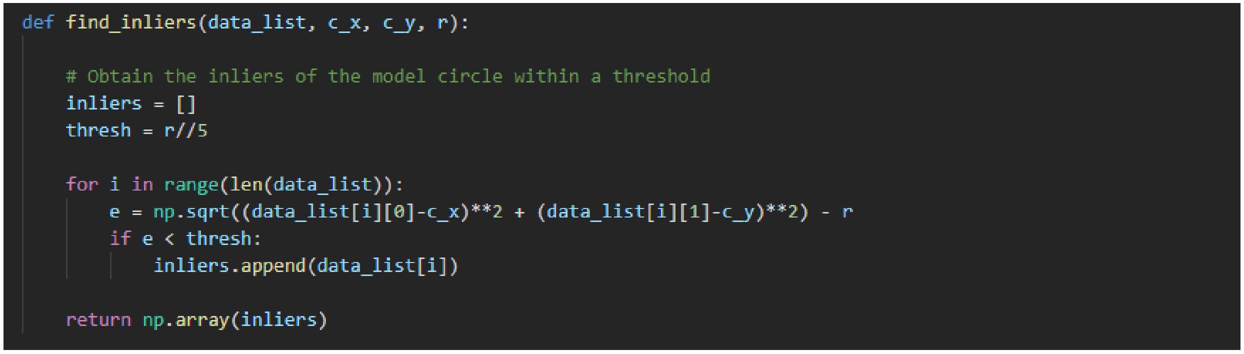
Code to estimate the circle center and radius from a set of inliers in a dataset using the RANSAC algorithm.



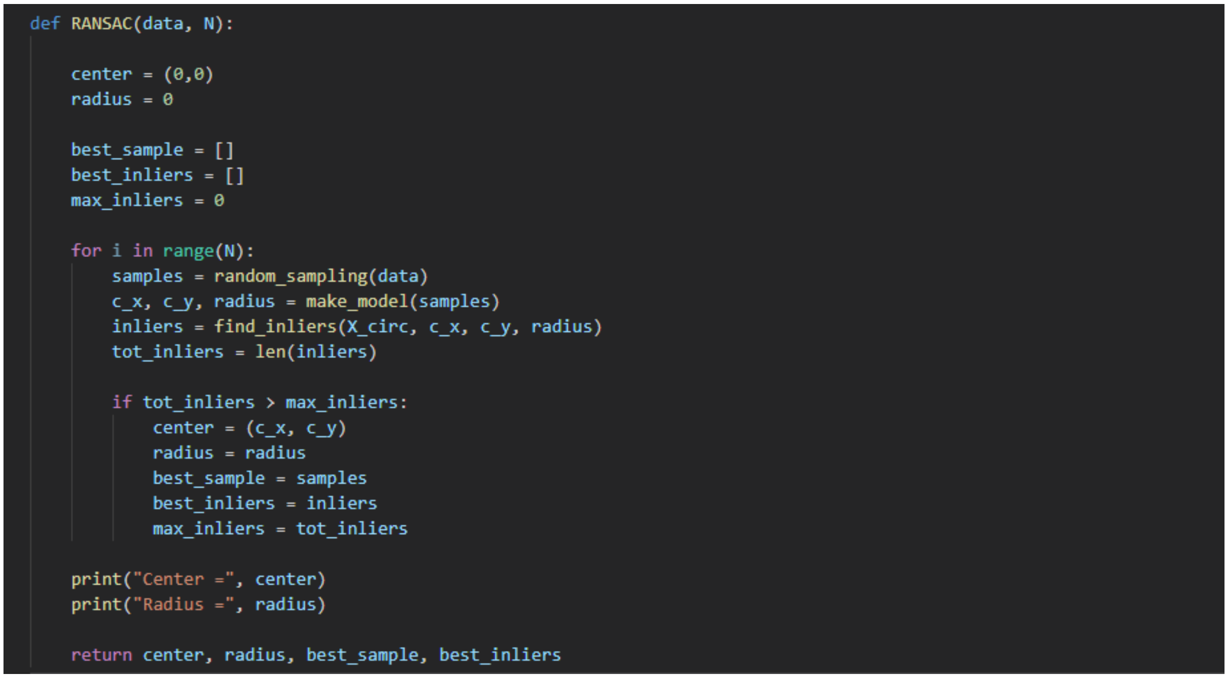
```python
def random_sampling(data):
    sample, ran_list, count = [], [], 0

    # Get three random points from data
    while True:
        ran = np.random.randint(len(data))

        if ran not in ran_list:
            sample.append(data[ran])
            ran_list.append(ran)
            count += 1

            if count == 3:
                break

    return np.array(sample)
```



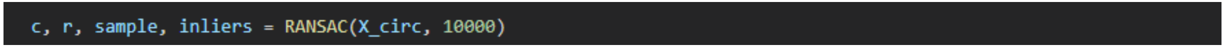
```python
def make_model(sample):

    # Create a model circle to fit the 3 points and return its center and radius
    pt1 = sample[0]
    pt2 = sample[1]
    pt3 = sample[2]

    A = np.array([[pt2[0] - pt1[0], pt2[1] - pt1[1]], [pt3[0] - pt2[0], pt3[1] - pt2[1]]])
    B = np.array([[pt2[0]**2 - pt1[0]**2 + pt2[1]**2 - pt1[1]**2], [pt3[0]**2 - pt2[0]**2 + pt3[1]**2 - pt2[1]**2]])
    inv_A = linalg.inv(A)

    c_x, c_y = np.dot(inv_A, B) / 2
    c_x, c_y = c_x[0], c_y[0]
    r = np.sqrt((c_x - pt1[0])**2 + (c_y - pt1[1])**2)

    return c_x, c_y, r
```



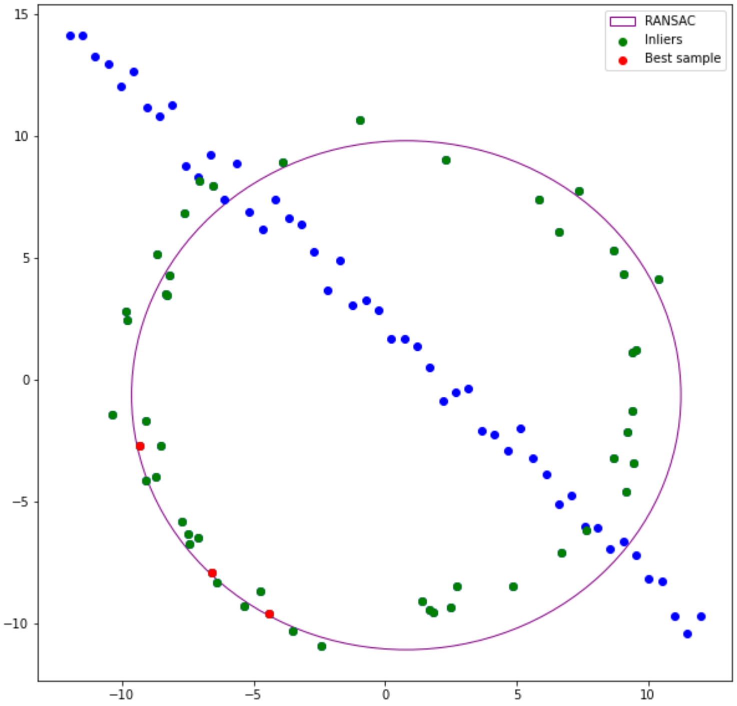
```python
def find_inliers(data_list, c_x, c_y, r):

    # Obtain the inliers of the model circle within a threshold
    inliers = []
    thresh = r//5

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

    return np.array(inliers)
```



```python
def RANSAC(data, N):

    center = (0,0)
    radius = 0

    best_sample = []
    best_inliers = []
    max_inliers = 0

    for i in range(N):
        samples = random_sampling(data)
        c_x, c_y, radius = make_model(samples)
        inliers = find_inliers(X_circ, c_x, c_y, radius)
        tot_inliers = len(inliers)

        if tot_inliers > max_inliers:
            center = (c_x, c_y)
            radius = radius
            best_sample = samples
            best_inliers = inliers
            max_inliers = tot_inliers

    print("Center =", center)
    print("Radius =", radius)

    return center, radius, best_sample, best_inliers
```



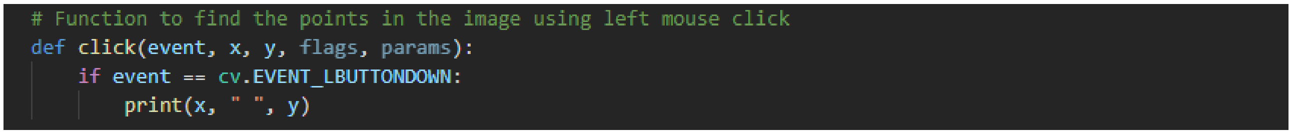
```python
c, r, sample, inliers = RANSAC(X_circ, 10000)
```



This RANSAC algorithm evaluates many different circles and returns the circle with the largest inlier set. Here the circle is estimated using 3 points. (by running the function for 3 random points until the best fit is found.)

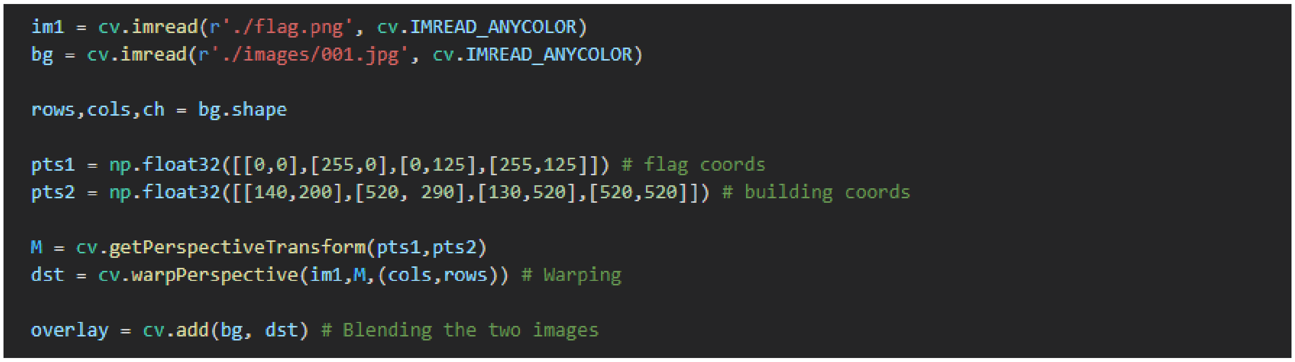
Question2

Code to superimpose two images by clicking four points on a planar surface in an image, computing a homography that maps the other image to this plane, and warping that image, and blending on to the first image.

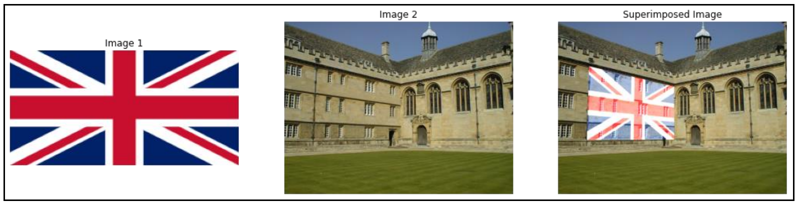


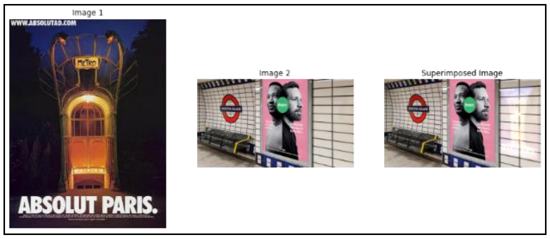
```python
# Function to find the points in the image using left mouse click
def click(event, x, y, flags, params):
    if event == cv.EVENT_LBUTTONDOWN:
        print(x, " ", y)
```

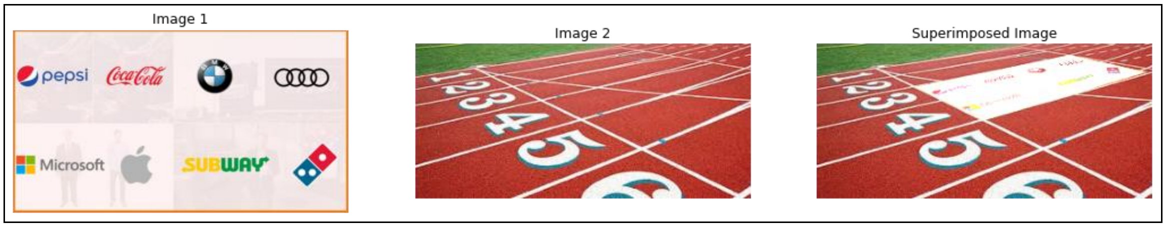
By clicking 4 points and obtaining their coordinates the homography matrix can be calculated to warp the images.



```python
im1 = cv.imread(r'./flag.png', cv.IMREAD_ANYCOLOR)
bg = cv.imread(r'./images/001.jpg', cv.IMREAD_ANYCOLOR)

rows,cols,ch = bg.shape

pts1 = np.float32([[0,0],[255,0],[0,125],[255,125]]) # flag coords
pts2 = np.float32([[140,200],[520, 290],[130,520],[520,520]]) # building coords

M = cv.getPerspectiveTransform(pts1,pts2)
dst = cv.warpPerspective(im1,M,(cols,rows)) # Warping

overlay = cv.add(bg, dst) # Blending the two images
```





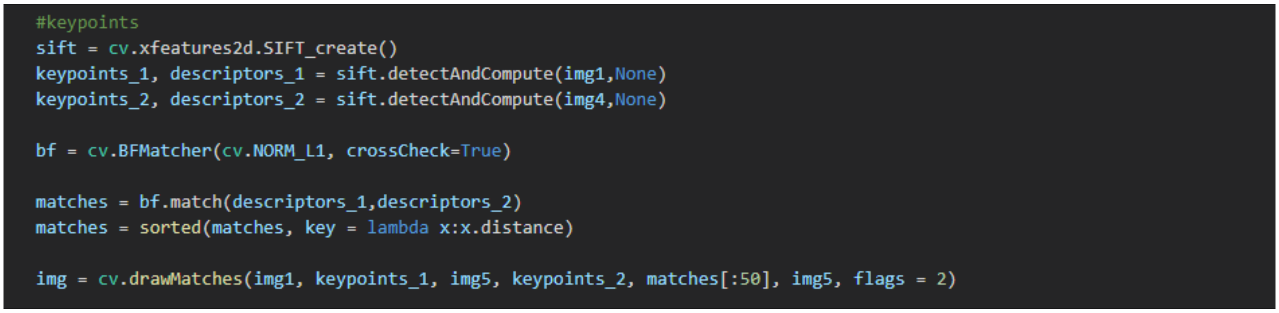


The homography matrix of the images depend on orientation, rotation and scaling of the images. The above shows several examples for warped imgaes with different homographies.

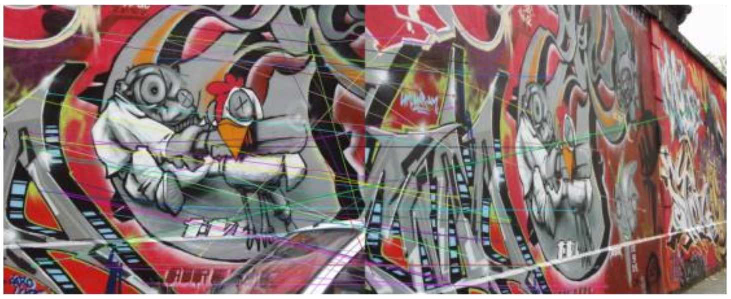
Question3

Stitching two graffiti images.

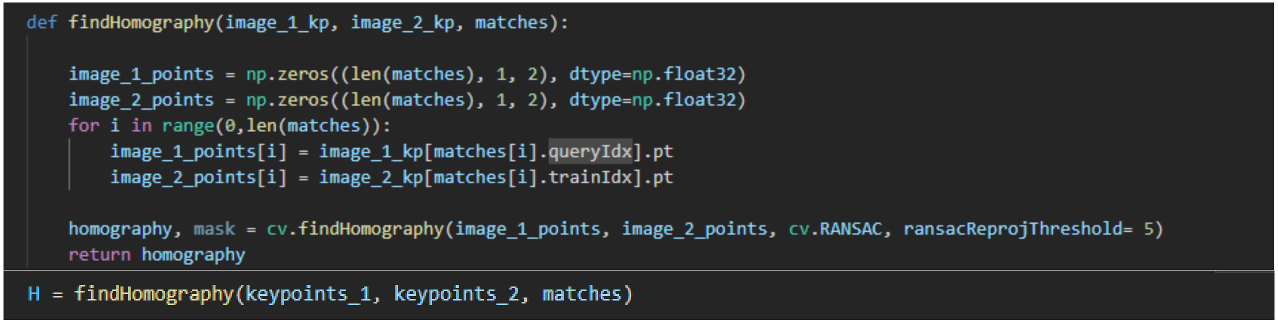
a) Matching SIFT features between the two images



```python
#keypoints
sift = cv.xfeatures2d.SIFT_create()
keypoints_1, descriptors_1 = sift.detectAndCompute(img1,None)
keypoints_2, descriptors_2 = sift.detectAndCompute(img4,None)

bf = cv.BFMatcher(cv.NORM_L1, crossCheck=True)

matches = bf.match(descriptors_1,descriptors_2)
matches = sorted(matches, key = lambda x:x.distance)

img = cv.drawMatches(img1, keypoints_1, img5, keypoints_2, matches[:50], img5, flags = 2)
```



b) Compute the homography using your own code within RANSAC and comparing with the homography given in the dataset.



```
def findHomography(image_1_kp, image_2_kp, matches):

    image_1_points = np.zeros((len(matches), 1, 2), dtype=np.float32)
    image_2_points = np.zeros((len(matches), 1, 2), dtype=np.float32)
    for i in range(0,len(matches)):
        image_1_points[i] = image_1_kp[matches[i].queryIdx].pt
        image_2_points[i] = image_2_kp[matches[i].trainIdx].pt

    homography, mask = cv.findHomography(image_1_points, image_2_points, cv.RANSAC, ransacReprojThreshold= 5)
    return homography
```

```
H = findHomography(keypoints_1, keypoints_2, matches)
```

Homography matrix for image 1 to 5 was obtained by matrix multiplication of the homograhies of 1 to 2, 2 to 3, 3 to 4, and 4 to 5.

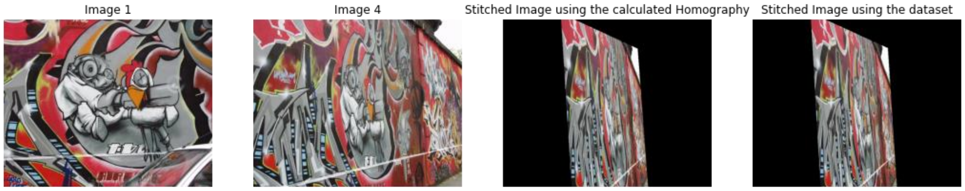
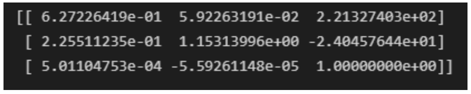
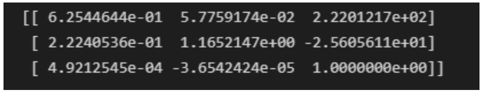


Image 1
Image 4
Stitched Image using the calculated Homography
Stitched Image using the dataset

Homographies obtained by the code when compared to that in the dataset were approximately equal.

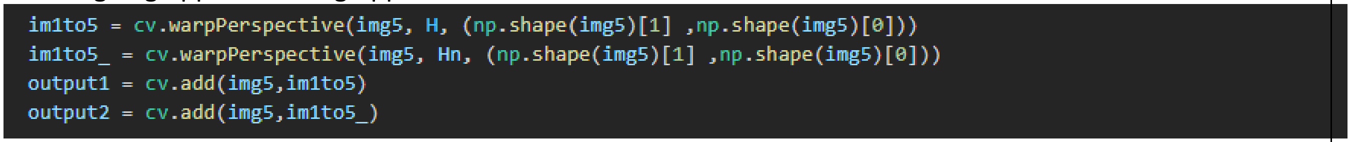


```
[[ 6.27226419e-01  5.92263191e-02  2.21327403e+02]
 [ 2.25511235e-01  1.15313996e+00 -2.40457644e+01]
 [ 5.01104753e-04 -5.59261148e-05  1.00000000e+00]]
```

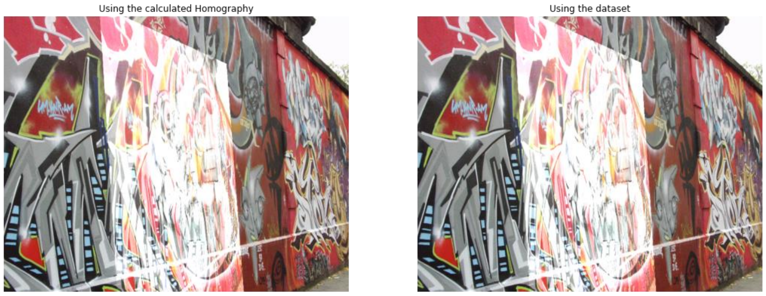


```
[[ 6.2544644e-01  5.7759174e-02  2.2201217e+02]
 [ 2.2240536e-01  1.1652147e+00 -2.5605611e+01]
 [ 4.9212545e-04 -3.6542424e-05  1.0000000e+00]]
```

c) Stitching img1.ppm onto img5.ppm.



```
im1to5 = cv.warpPerspective(img5, H, (np.shape(img5)[1] ,np.shape(img5)[0]))
im1to5_ = cv.warpPerspective(img5, Hn, (np.shape(img5)[1] ,np.shape(img5)[0]))
output1 = cv.add(img5,im1to5)
output2 = cv.add(img5,im1to5_)
```



Above diagram shows the final stitched image of 1 on the image 5.

Github Link: https://github.com/SenuriMR/Fundamentals-of-Image-Processing-and-Machine-Vision/tree/main/Assignment%202