EN2550 – Fundamentals of Image Processing and Machine Vision

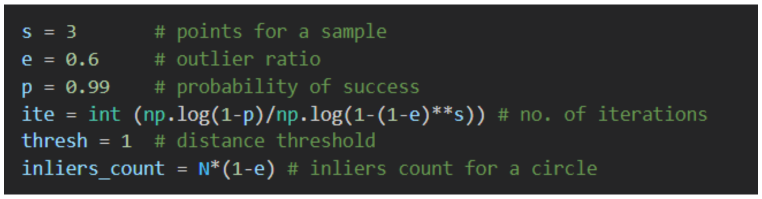
Assignment 02

Name: ADIKARI A.M.A.D.

Index No: 190021A

❖ Question 01

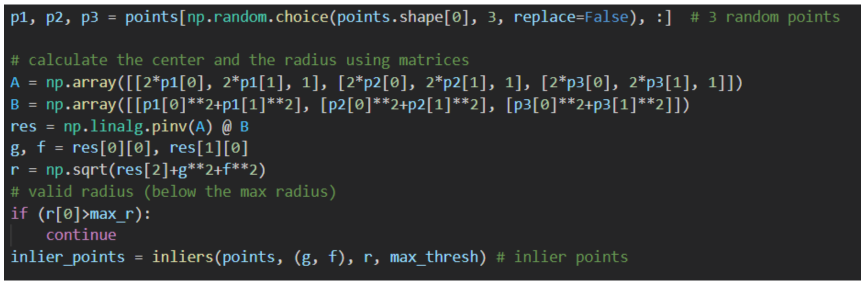
- In this question, we must find the best-fitting circle using the RANSAC algorithm for given data points. One hundred data points have been generated with noise.


```
s = 3       # points for a sample
e = 0.6     # outlier ratio
p = 0.99    # probability of success
ite = int (np.log(1-p)/np.log(1-(1-e)**s)) # no. of iterations
thresh = 1  # distance threshold
inliers_count = N*(1-e) # inliers count for a circle
```

- We need three points for a sample and for the parameter values in this figure, we can calculate the number of iterations that we need to run to get an accurate circle. For these values, the minimum number of iterations is 69.

• RANSAC algorithm

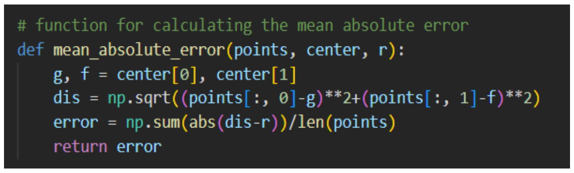
-By using the RANSAC() function, first, we get a sample with three random data points and then we determine the circle for the selected points. According to the following figure, the radius and center of the circle can be found by using matrices. Then the radius is checked to whether it is a valid value since if the sample points are very closer to each other, the radius would be higher. After that, we can find the inliers for the circle by using the inliers() function. Inliers are found at a distance from the center within a region of radius ± threshold. For this example, 1 is given as the threshold to get enough inliers count.

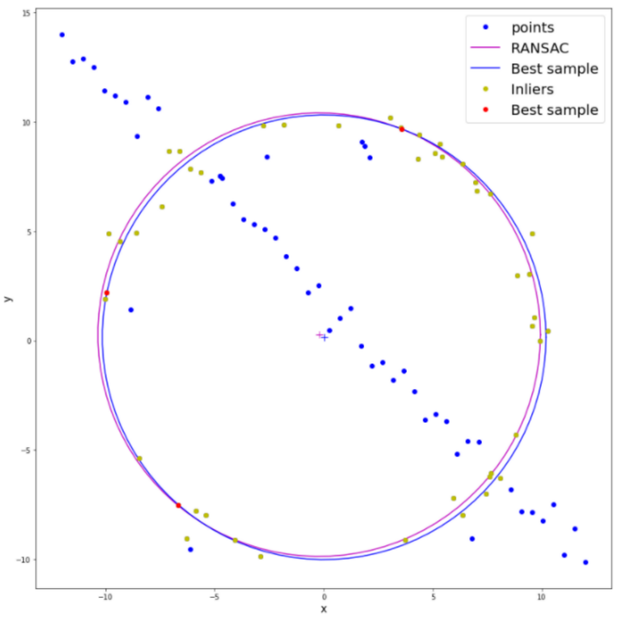

```
p1, p2, p3 = points[np.random.choice(points.shape[0], 3, replace=False), :]  # 3 random points

# calculate the center and the radius using matrices
A = np.array([[2*p1[0], 2*p1[1], 1], [2*p2[0], 2*p2[1], 1], [2*p3[0], 2*p3[1], 1]])
B = np.array([[p1[0]**2+p1[1]**2], [p2[0]**2+p2[1]**2], [p3[0]**2+p3[1]**2]])
res = np.linalg.pinv(A) @ B
g, f = res[0][0], res[1][0]
r = np.sqrt(res[2]+g**2+f**2)
# valid radius (below the max radius)
if (r[0]>max_r):
    continue
inlier_points = inliers(points, (g, f), r, max_thresh) # inlier points
```

- If the inliers count is greater or equal to the given threshold count, the above RANSAC process is repeated with those inlier points to find a more accurate circle with higher inliers count. All these circles are listed.

- After all the iterations perform, the circle with the highest inliers count will be selected as the best circle by using the best_RANSAC() function. If several circles have similar highest inliers count, then the circle with the minimum mean absolute error will be selected. It is calculated by $\sum_{n=1}^{N} |r' - r| / N$ while r'=distance from the center, r=radius, and N=number of data points.


```
# function for calculating the mean absolute error
def mean_absolute_error(points, center, r):
    g, f = center[0], center[1]
    dis = np.sqrt((points[:, 0]-g)**2+(points[:, 1]-f)**2)
    error = np.sum(abs(dis-r))/len(points)
    return error
```



- This figure shows the results of the RANSAC circle fitting. For this example, the Inliers count is 48. + signs represent the centers of the circles. Two circles are the first and second results of running the RANSAC function. Circle points are got by using the circle_points() function.

➢ Some of these given data points are according to a line outside the circle. If we use a method other than RANSAC, those points will be affected to the circle fitting due to spread. Therefore, the RANSAC method is especially useful and valuable for fittings in data points with widespread outliers. And in RANSAC, if we use more iterations, a more accurate circle fitting can be performed.

❖ Question 02

- In this question, we need to find a homography for image pairs and by using that, warp and blend one image to a plane of the other image.

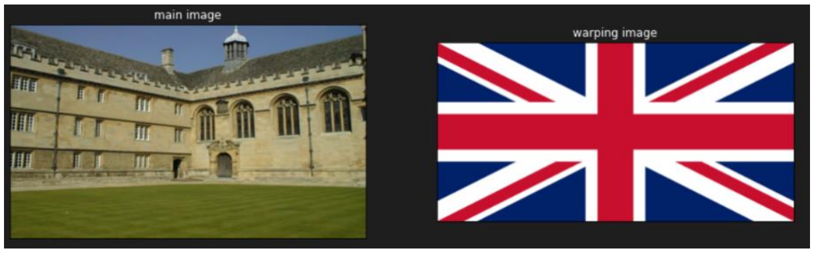
• Algorithm

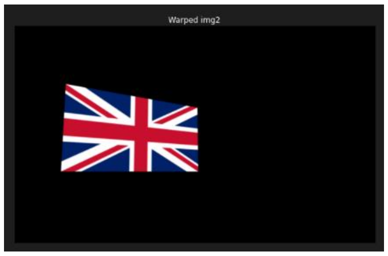
- First, the main image is shown using the OpenCV window. Then we can click 4 points (for the corner points of the warping image) using the mouse in the desired plane and the clicked points are shown in red circles. The mouse_click() and mousePoints() functions are used for this purpose.

- In the warpAndBlend() function, a homography is determined using the findHomography() built-in function for the image pair. Then by using that, we can warp and blend the warping image to the selected plane of the main image as shown in the following figure. While warping, the warping image is resized to the shape of the main image.

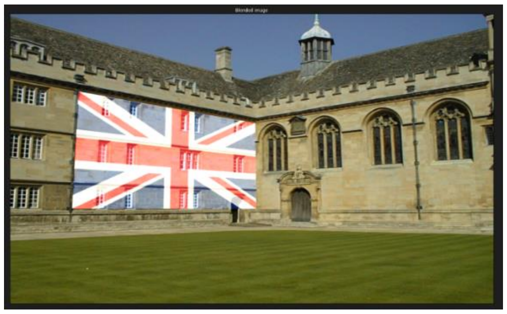
```
h, status = cv.findHomography(img2_points, clicked_points)                    # homography
output_img = cv.warpPerspective(img2, h, (img1.shape[1], img1.shape[0]))       # warping img2 into img1
blend_img = cv.addWeighted(img1, alpha, output_img, beta, gamma)            # blending img2 into img1
```

- Here, alpha and beta parameters in the addWeighted() function are the weights of the two images, and pixel values are calculated by $I = \frac{alpha*p1+beta*p2}{alpha+beta}$ while p1, p2 are pixel values of the main image and the warping image. Gamma is the value that is added to the final image. Then the results are shown using matplotlib.





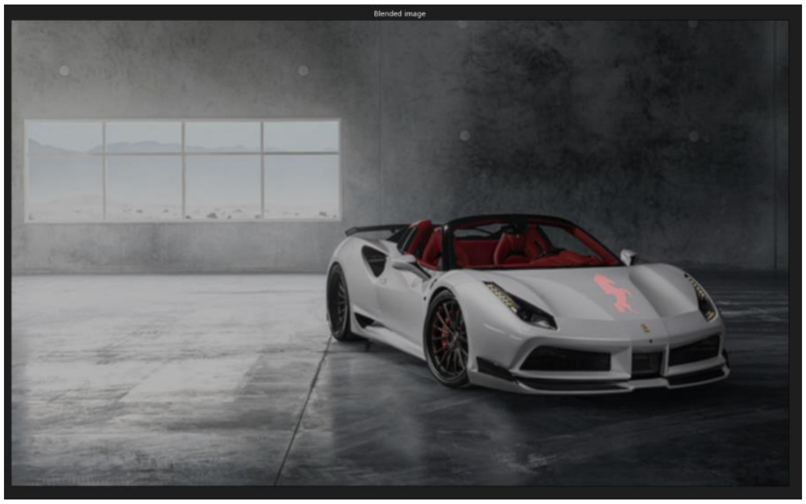
○ Building and the flag



- The above figures are the original images of the main image and the warping image and the warped image that is generated using the computed homography. The figure on the left side shows the final output image. The flag is superimposed on a wall of the building. For this example, the parameter values in the blending are as follows.

alpha = 1, beta = 0.8, gamma = 0

○ Car and the logo



- In this image, the logo of a horse is superimposed on the bonnet of the car. In this logo image, the horse is not rectangular and if we use the logo directly, a rectangular shape of the logo will be shown on the bonnet with an unmatched color. Therefore, the outside region of the horse is converted to black color by using a mask. Then the black region will not affect when we add the weights. For this example, weight parameters are used as alpha=0.7, beta=0.3, and gamma=0 to get a good and some kind of dark output image. We can use this logo image as a sticker on the bonnet.

- **Wall and the headset**

Blended image

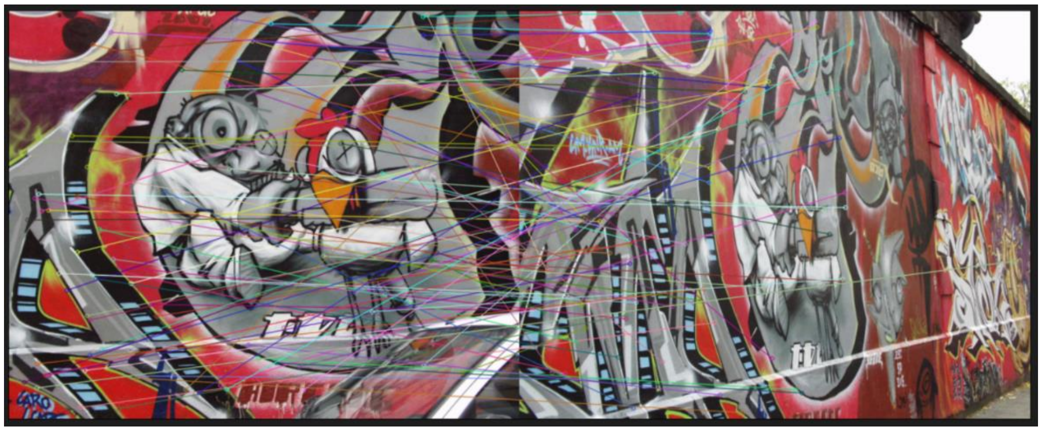
- In this image, the design of the headset is superimposed on the wall. In the original image of the headset, the headset was black, and the outside region was white. To get this image, a mask is used to convert the headset to white color and the outside region to black. 1, 1, and 0 are respectively used as the values of the parameters when adding weights. This wall image is an image of a wall that has been taken in a music studio. Therefore, the headset design is matching to the scenario, and the parameter values have been selected as it stands out with the guitar.

➢ Here, we can use any plane in the main image to warp and blend another image on that selected plane. Blending is done by adding weights to each image. In black regions, pixel values are 0 then, for any weight it gives only 0. Therefore, black image parts cannot be blended. By using this warping and blending, we can superimpose an image or a design on another image.

❖ Question 03

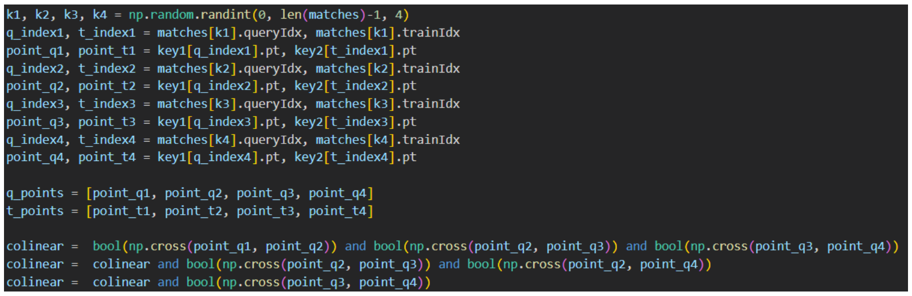
- In this question, we need to stitch img1.ppm onto img5.ppm.

Part a) SIFT algorithm is used to detect and match the features. Due to the higher inclination of the wall in img5.ppm relative to img1.ppm, SIFT algorithm gives inaccurate matching pairs between these two images.



This image includes some of those matchings. Most of those are incorrectly matched. Due to this matter, we have to use other graffiti images also for this problem. Therefore, img2.ppm and img4.ppm are also taken for the SIFT matching and the next parts.

Part b) In this part we need to compute the homography between two images. RANSAC algorithm has been used to compute the homography and the homoAndGraph() function is used to run the RANSAC, implement the graph including data points, and return the best homography. First, using the matches given by the SIFT algorithm are abstracted into two lists as points of the query image and train image. The number of iterations in RANSAC is calculated by using the parameters. In RANSAC() function, first, four query points are randomly selected and then the train points are selected according to those. After that, the collinearity of those points is checked.



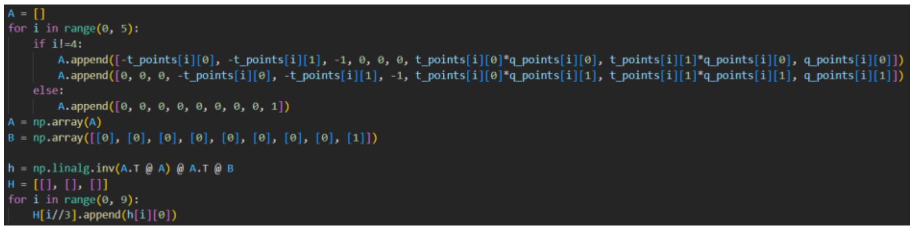
```python
k1, k2, k3, k4 = np.random.randint(0, len(matches)-1, 4)
q_index1, t_index1 = matches[k1].queryIdx, matches[k1].trainIdx
point_q1, point_t1 = key1[q_index1].pt, key2[t_index1].pt
q_index2, t_index2 = matches[k2].queryIdx, matches[k2].trainIdx
point_q2, point_t2 = key1[q_index2].pt, key2[t_index2].pt
q_index3, t_index3 = matches[k3].queryIdx, matches[k3].trainIdx
point_q3, point_t3 = key1[q_index3].pt, key2[t_index3].pt
q_index4, t_index4 = matches[k4].queryIdx, matches[k4].trainIdx
point_q4, point_t4 = key1[q_index4].pt, key2[t_index4].pt

q_points = [point_q1, point_q2, point_q3, point_q4]
t_points = [point_t1, point_t2, point_t3, point_t4]

colinear =  bool(np.cross(point_q1, point_q2)) and bool(np.cross(point_q2, point_q3)) and bool(np.cross(point_q3, point_q4))
colinear =  colinear and bool(np.cross(point_q2, point_q3)) and bool(np.cross(point_q2, point_q4))
colinear =  colinear and bool(np.cross(point_q3, point_q4))
```

- Then to find the homography, A and B matrices are used as shown in the following figure. Here, the Ah=B matrices format is used to compute the homography and the A and B matrices are formed as shown in the figure. The x and y are the points in the train image and the x' and y' are in the query image. The last row in A is used to calculate this in Ah=B format, otherwise, it will be in Ah=0 format. Since h9=0, we can use that last row in A and the last row in B as 1. After the A and B matrices

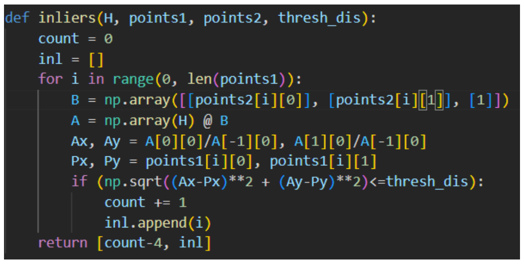
$$\begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1x'_1 & y_1x'_1 & x'_1 \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & x_1y'_1 & y_1y'_1 & y'_1 \\ -x_2 & -y_2 & -1 & 0 & 0 & 0 & x_2x'_2 & y_2x'_2 & x'_2 \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & x_2y'_2 & y_2y'_2 & y'_2 \\ -x_3 & -y_3 & -1 & 0 & 0 & 0 & x_3x'_3 & y_3x'_3 & x'_3 \\ 0 & 0 & 0 & -x_3 & -y_3 & -1 & x_3y'_3 & y_3y'_3 & y'_3 \\ -x_4 & -y_4 & -1 & 0 & 0 & 0 & x_4x'_4 & y_4x'_4 & x'_4 \\ 0 & 0 & 0 & -x_4 & -y_4 & -1 & x_4y'_4 & y_4y'_4 & y'_4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} h1 \\ h2 \\ h3 \\ h4 \\ h5 \\ h6 \\ h7 \\ h8 \\ h9 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$



```
A = []
for i in range(0, 5):
    if i!=4:
        A.append([-t_points[i][0], -t_points[i][1], -1, 0, 0, 0, t_points[i][0]*q_points[i][0], t_points[i][1]*q_points[i][0], q_points[i][0]])
        A.append([0, 0, 0, -t_points[i][0], -t_points[i][1], -1, t_points[i][0]*q_points[i][1], t_points[i][1]*q_points[i][1], q_points[i][1]])
    else:
        A.append([0, 0, 0, 0, 0, 0, 0, 0, 1])
A = np.array(A)
B = np.array([[0], [0], [0], [0], [0], [0], [0], [0], [1]])

h = np.linalg.inv(A.T @ A) @ A.T @ B
H = [[], [], []]
for i in range(0, 9):
    H[i//3].append(h[i][0])
```

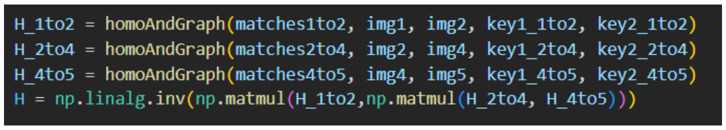
are formed, the homography matrix components can be determined by H = $(A^TA)^{-1}A^TB$ as shown in the figure. Then the homography matrix (3x3) can be formed.



```
def inliers(H, points1, points2, thresh_dis):
    count = 0
    inl = []
    for i in range(0, len(points1)):
        B = np.array([[points2[i][0]], [points2[i][1]], [1]])
        A = np.array(H) @ B
        Ax, Ay = A[0][0]/A[-1][0], A[1][0]/A[-1][0]
        Px, Py = points1[i][0], points1[i][1]
        if (np.sqrt((Ax-Px)**2 + (Ay-Py)**2)<=thresh_dis):
            count += 1
            inl.append(i)
    return [count-4, inl]
```

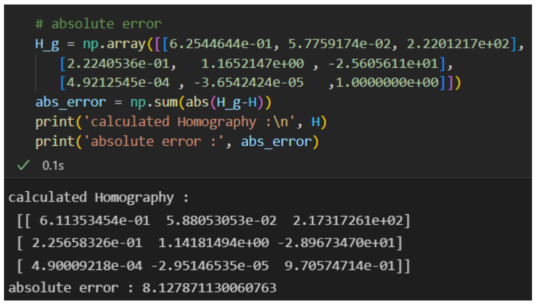
- After that, using the inliers() function, the homography matrix is used for every point in the query image given by the SIFT algorithm to get matching points in the train image. Those points are compared with the train image points given by the SIFT with a given threshold pixel value. Then the function returns the inliers count. After all the iterations are performed in the RANSAC() function, the homography with the highest inliers count will be selected as the best and returned it. After that, the homoAndGraph() function plots the graph with data points and shows the images with the best sample of four points.

- Due to the problem explained in part a), the img1to5 homography should be determined by using img1.ppm, img2.ppm, img4.ppm and img5.ppm. The homographies of img1to2, img2to4, and img4to5 are calculated from the above process with the separate matching points given by the SIFT algorithm. The final img1to5 homography is



```
H_1to2 = homoAndGraph(matches1to2, img1, img2, key1_1to2, key2_1to2)
H_2to4 = homoAndGraph(matches2to4, img2, img4, key1_2to4, key2_2to4)
H_4to5 = homoAndGraph(matches4to5, img4, img5, key1_4to5, key2_4to5)
H = np.linalg.inv(np.matmul(H_1to2,np.matmul(H_2to4, H_4to5)))
```

determined by the inverse of the matrix multiplication of those three homographies as shown in the figure.



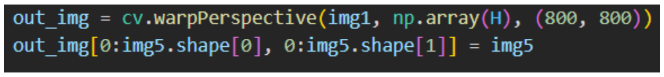
```
# absolute error
H_g = np.array([[6.2544644e-01, 5.7759174e-02, 2.2201217e+02],
    [2.2240536e-01,   1.1652147e+00 , -2.5605611e+01],
    [4.9212545e-04 , -3.6542424e-05   ,1.0000000e+00]])
abs_error = np.sum(abs(H_g-H))
print('calculated Homography :\n', H)
print('absolute error :', abs_error)
✓ 0.1s

calculated Homography :
 [[ 6.11353454e-01  5.88053053e-02  2.17317261e+02]
 [ 2.25658326e-01  1.14181494e+00 -2.89673470e+01]
 [ 4.90009218e-04 -2.95146535e-05  9.70574714e-01]]
absolute error : 8.127871130060763
```

- After determining the final homography, that is compared with the given img1to5 homography. It is done by using an absolute error calculation between the components of two matrices. The absolute error is calculated by getting the sum of the absolute difference in each corresponding element. This error is changed for every run in finding homography. For the shown example, the calculated homography is shown in the figure. The homography components are almost similar to the given one and the absolute error is 8.1278.

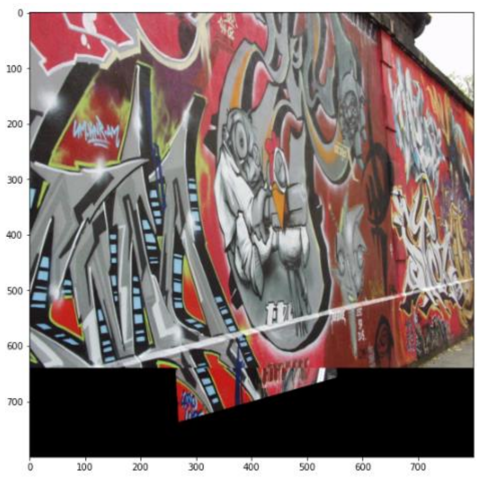
Part c) In this part, the img1.ppm needs to be stitched on img5.ppm.

- Before the stitching, img1.ppm needs to be warped into the plane of img5.ppm by using the calculated homography. The size of the warped image must be larger than img5.ppm as the final stitched image can be larger than it. Therefore, (800, 800) is given as the size of the warped image. The original size of the img5.ppm is (800,



```
out_img = cv.warpPerspective(img1, np.array(H), (800, 800))
out_img[0:img5.shape[0], 0:img5.shape[1]] = img5
```

640). When stitching, the warped image and img5.ppm are joined together.



- The final output image is shown here. A part of img1.ppm has been added at the bottom as a separate part. This happened since that part is not available in img5.ppm.

➢ In this question, due to the high inclination of the wall in img5.ppm relative to img1.ppm, a homography cannot be calculated directly as the SIFT algorithm gives inaccurate and incorrect matchings. Therefore, SIFT is not suitable for this kind of image pair to direct calculations, otherwise more images are needed. The final output image needs a size more than the original images since some parts of one image may be unavailable in the other image.

- GitHub link: https://github.com/ashend99/Image-Processing/tree/main/Assignments/Assignment%202