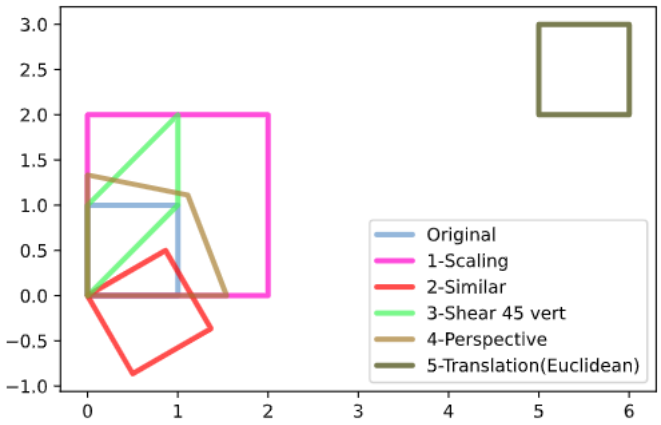
1 2-D tansformations




```
H_1 = [[2. 0. 0.],[0. 2. 0.],[0. 0. 1.]]                          #scaling
H_2 = [[ 0.5 0.8660254  0.],[-0.8660254  0.5 0.],[ 0. 0. 1.]] #similar,rotation
H_3 = [[-1,0,0], [0,1,0], [0., 0., 1.]]                           #shear
H_4 = [[2.  0.  0. ],[0.  2.  0. ],[0.3 0.5 1. ]]                 #perspective
H_5 = [[1,0,7],[0,1,3],[0,0,1]]                                   #translation
```

Transform matrices

2 Transform Graffiti

The im1.ppm and im5.ppm are stitched using the given homography matrix in the file H1to5p. The part of im1.ppm can be seen at the bottom, the result has been cropped in the above figure. First the image coordinates are converted to homogeneous coordinates(lines 1-3). Then using numpy np.dot() method the homogeneous coordinates are matrix multiplied to generate the transformed homogeneous coordinates(line 4), it increases efficiency. The values of image 1 are mapped to the transformed coordinates in the `nested for loop`.

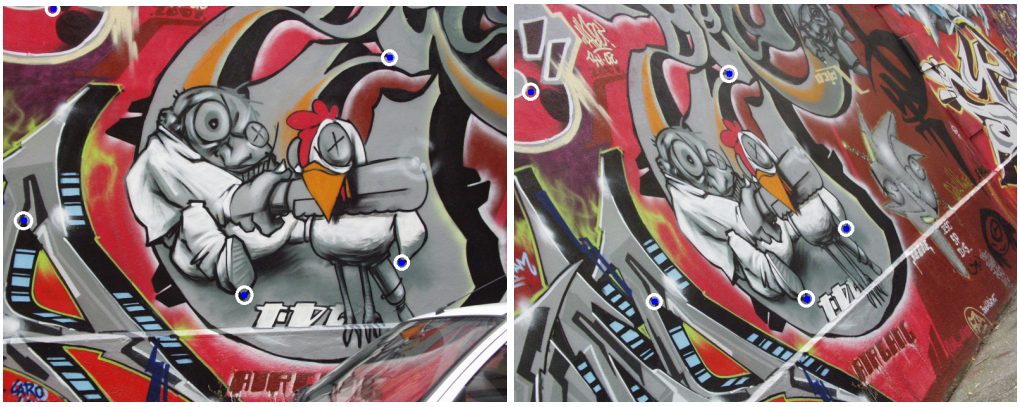
```
h, w = im1.shape[0:2]
Y, X = np.indices((h,w))
homo_coods = np.stack((X.ravel(), Y.ravel(), np.ones(Y.size)))
homo_coods_trans = H.dot(homo_coods)
homo_coods_trans /= homo_coods_trans[2,:]
iteration =0
for row in range(im1.shape[0]):
    for col in range(im1.shape[1]):
        v = homo_coods_trans[0][iteration]
```

```
10          u = homo_coods_trans[1][iteration]
11          u = int(round(u))
12          v = int(round(v))
13          canvas[u][v] = im1[row,col]
14          iteration += 1
15  canvas[0:im1.shape[0], 0:im1.shape[1]] = im5
```

Code snippet #1

3 Mouse clicking and selecting the matching points

The im4.ppm is warped and stitched onto im1.ppm.



(a) Selected points on image 1 (b) Selected points on image 2

Figure 2: Stitched image

The homography matrix is computed using the openCV function `cv.findHomography()`. The least squared method was used(third argument '0' in the function). The transformation was carried out using the openCV function `cv.warpPerspective()`. The stitching was carried out in a similar manner to line 15 in code snippet #1.

```
1  (H, status) = cv.findHomography(p1, p2,0)
2  im4_warped = cv.warpPerspective(im4, np.linalg.inv(H), (1000,1000))
3  im4_warped[0:im1.shape[0], 0:im1.shape[1]] = im1
```

Code snippet #2

The given ground truth homography in file H1to4p is,

```
[[0.66378505, 0.68003334, -31.230335],
    [-0.144955, 0.97128304, 148.7742],
[0.00042518504, -1.3930359e-05, 1.0]]
```

The generated Homography H

```
[[ 6.56577154e-01  6.66930489e-01 -2.80017613e+01]
 [-1.44387811e-01  9.55250451e-01  1.49509799e+02]
 [ 4.13200970e-04 -3.47799483e-05  1.00000000e+00]]
```

The values are very accurate.

4 Computing Homography using SVD decomposition

The homography matrix can be solved using the method 'Singular Value Decomposition' in linear algebra. First the set of points are converted to homogeneous corrdinates by appending a 1. ay, ax are calculated according to the formula which can be found in linear algebra. This setup is done to solve the equation $A \times h = 0$.

```
A = []
for i in range(len(p1)):
    pt1 = np.matrix([p1[i].item(0), p1[i].item(1), 1]) #convert to homogeneous
    pt2 = np.matrix([p2[i].item(0), p2[i].item(1), 1]) #convert to homogeneous

    ay = [0, 0, 0, -pt2.item(2) * pt1.item(0), -pt2.item(2) * pt1.item(1), -pt2.item
    (2) * pt1.item(2),
            pt2.item(1) * pt1.item(0), pt2.item(1) * pt1.item(1), pt2.item(1) * pt1.
    item(2)]
    ax = [-pt2.item(2) * pt1.item(0), -pt2.item(2) * pt1.item(1), -pt2.item(2) * pt1.
    item(2), 0, 0, 0,
            pt2.item(0) * pt1.item(0), pt2.item(0) * pt1.item(1), pt2.item(0) * pt1.
    item(2)]
    A.append(ax)
    A.append(ay)
matrixA = np.matrix(A)
#svd composition to calculate H using least squares
u, s, v = np.linalg.svd(matrixA)
# the last item of v orthogonal matrices gives the answer to H, reshape it into 3x3
H = np.reshape(v[8], (3, 3))
H = (1/H.item(8)) * H #normalizing
```

Homography matrix calculation using Singular Value Decomposition

The generated homography matrix is,

```
[[ 6.57694299e-01  6.70396380e-01 -2.89880486e+01]
 [-1.45762855e-01  9.54603223e-01  1.50889439e+02]
 [ 4.32185018e-04 -5.12931346e-05  1.00000000e+00]]
```

(a) Homography using SVD (b) Stitch in Q3, using the openCV function

5 BONUS: Stitch using SuperGlue, Homography calculation using RANSAC

The SuperGlue script[**1**] is run without resizing images.

```
>>>python match_pairs.py --resize -1
```

The script generates the numpy library files in the path specified in the script under '–output_dir'. The file names of the pairs of images to be matched must be included in a text file which the path should be included in the '–input_pairs'. The default values in the script file was changed instead of passing the parameters in the command line for the ease of use.
The generated numpy library contains the following.

```
['keypoints0', 'keypoints1', 'matches', 'match_confidence']
```

The `'keypoints0'` and `'keypoints1'` contains the coordinates for the feature points in image 1 and image 2 respectively. `'matches'` contains the indices for the matching feature points in `'keypoints1'` for each coordinate in `'keypoints0'`. If the value is -1 there is no match. The `'match_confidence'` contains a measurement of how certain the matching is. The matching points are selected with a threshold in `'match_confidence'`. For the RANSAC algorithm a large number of data points is preferred, however to disregard the totally unreliable matches a threshold is applied in selecting the matching points. Then they are put to two separate arrays kp0, kp1 such that the corresponding indices of the arrays contains a matching pair of point coordinates.This is done in the `for loop`, line 6- 19.

```
1  path = 'C:/Users/menuw/Documents/Modules/Semester_4/EN2550_Computer_Vision/
       SuperGluePretrainedNetwork/dump_match_pairs/img1_img4_matches.npz'
2
3  npz = np.load(path)
4  threshold = 0.8
5  length = 0
6  for i in range(npz['matches'].shape[0]):
7      if npz['match_confidence'][i] > threshold:
8          length += 1
9  kp0 = np.empty((length,2),dtype='float32')
10 kp1 = np.empty((length,2),dtype ='float32')
11 j=0
12 for i in range(npz['keypoints0'].shape[0]):
13     if npz['matches'][i] > -1:
14         if npz['match_confidence'][i] > threshold:
15             kp0[j][0]=npz['keypoints0'][i][0]
16             kp0[j][1]=npz['keypoints0'][i][1]
17             kp1[j][0]=npz['keypoints1'][npz['matches'][i]][0]
18             kp1[j][1]=npz['keypoints1'][npz['matches'][i]][1]
19             j +=1
20 (Homo, status) = cv.findHomography(kp0, kp1,cv.RANSAC,6)
```

SuperGlue for feature matching

The RANSAC algorithm is used to calculate the homography, it is carried out by passing the `cv.RANSAC` argument to the `cv.findHomography()` function. The number of iterations used in RANSAC algorithm is 6. The rest of the stitching is carried out using the method used in the previous sections.
The calculated homography matrix.

```
[[ 6.82138130e-01  6.97584233e-01 -3.75023822e+01]
 [-1.41716073e-01  9.87390300e-01  1.47696141e+02]
 [ 4.50428956e-04  4.62475756e-06  1.00000000e+00]]
```

Figure 4: Stitched image

6 BONUS:Stitching multiple images using mouse clicked points

The code used in question 3 is developed to carry out 3 images. The selected images are *img1.ppm*, *img4.ppm*, *img6.ppm* .
First *img4.ppm* is stitched to *img1.ppm*. Then the *img6.ppm* is brought to *img1.ppm*'s perspective (lines 16,17) and the previosly stitched image is stitched to the last image (lines 19,20). The homography matrices H1 and H2 corresponds to these stitchings.

```
1  im6 = cv.imread('images/assn2/graf/img6.ppm', cv.IMREAD_ANYCOLOR)
2  p3 = np.empty((N,2))
3  im6copy = im6.copy()
4  param = [p3, im6copy]
5  n = 0
6  cv.namedWindow("Image 6", cv.WINDOW_AUTOSIZE)
7  cv.setMouseCallback('Image 6',draw_circle, param)
8
9  while(1):
10     cv.imshow("Image 6", im6copy)
11     if n == N:
12         break
13     if cv.waitKey(20) & 0xFF == 27:
14         break
15 (H1, status) = cv.findHomography(p1, p2,0)
16 im4_warped = cv.warpPerspective(im4, np.linalg.inv(H1), (900,900))
17 im4_warped[0:im1.shape[0], 0:im1.shape[1]] = im1
18 (H2, status) = cv.findHomography(p1, p3,0)
19 im6_warped = cv.warpPerspective(im6, np.linalg.inv(H2), (2000,2000))
20 im6_warped[0:im4_warped.shape[0], 0:im4_warped.shape[1]] = im4_warped
```

(a) Image 4 (b) Image 6

Figure 6: Stitched image, the canvas has been cropped to emphasize the stitching area