Assignment 2

1) $H = \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$

The properties of the shape could be changed by changing the values of the elements in the above matrix.

```
t = np.pi/3
H = np.identity(3)
H[0,0] = 2 #changes the width (size in x-direction)
H[0,1] = np.sin(t) #changes the angle of vertical lines
H[0,2] = np.sin(t) #translation in x-direction
H[1,0] = np.sin(t) #changes the angle of the horizontal lines
H[1,1] = 2 #changes the height (size in y-direction)
H[1,2] = np.sin(t) #translation in y-direction
H[2, 0] = 0.3 #changes the parallelism between horizontal lines
H[2, 1] = 0.5 #changes the parallelism between horizontal lines
H[2,2] = np.sin(t) #zooming property
```

Figure 1 code showing what each element value stands for

Following images are obtained by commenting some of the above steps of the code.

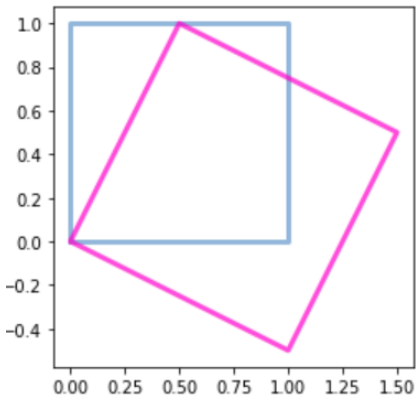

Figure 2 rotation

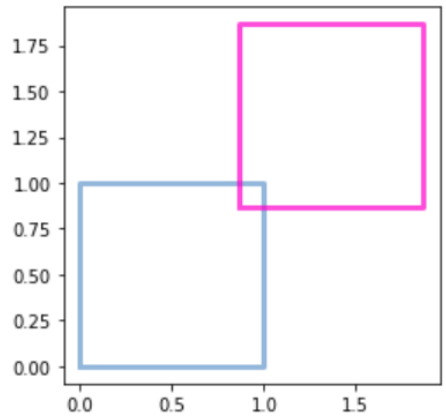

Figure 3 translation

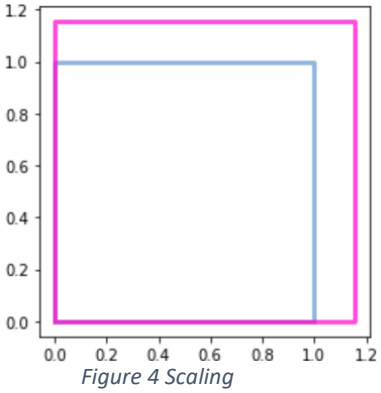

Figure 4 Scaling

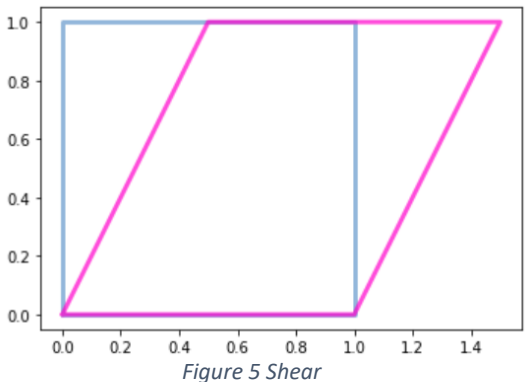

Figure 5 Shear

2) Warping has been coded here instead of the inbuilt code. Transforming img1 to plane of img5.

```
def img_to_array(img):
    r, c, ch = img.shape
    im_array = np.zeros((c,r, ch), dtype= img.dtype)
    for i in range(r):
        im_array[:,i] = img[i]
    return im_array
def array_to_img(arr):
    r,c,ch = arr.shape
    image = np.zeros((c,r,ch) , dtype=arr.dtype)
    for i in range(c):
        image[i] = arr[:,i]
    return image
def warpPerspective(img, H, size):
    r, c = size
    R, C = img.shape[0], img.shape[1]
    new_img = np.zeros((r,c, img.shape[2]), dtype=img.dtype)
    img_array = img_to_array(img)
    for i in range(r):
        for j in range(c):
            p = np.matmul( np.linalg.inv(H), np.array([i, j ,1]))
            p = p/p[2]
            i1 = int(p[0])
            j1 = int(p[1])
            if (i1 >= 0 and i1 < C) and  (j1 >= 0 and j1 < R):
                new_img[i,j] = img_array[i1, j1]
    new_img = array_to_img(new_img)
    return new_img
```

Figure 6 warping code

```
im1_warped = warpPerspective(im1, H, (im1.shape[1]+im5.shape[1],im1.shape[0]))
```

Figure 7 transform img1 to plane of img5

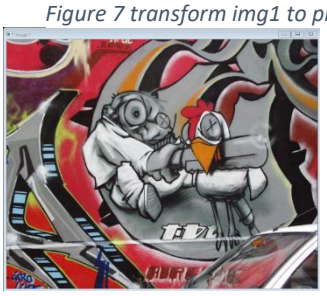

Figure 8 img1

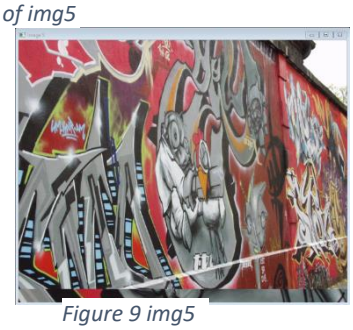

Figure 9 img5

Figure 10 warped image 1

Transforming img5 to plane of img1 and stitching. The size has been changed in the code to join major part of the 2nd image to 1st.

```
im5_warped = warpPerspective(im5, np.linalg.inv(H), (im1.shape[1]+im5.shape[1],im1.shape[0]))
im5_warped[0:im1.shape[0], 0:im1.shape[1]] = im1
```

Figure 2 Tranform img5 to img1 and stitch

```
[[ 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]]
```

Figure 3 H

Figure 6 transformed and stitched image1

3) In the given code variable im4 was changed to im5. And the following code is continued. Here, for warping the inbuilt function was used instead of the function that was used in question 2. To avoid ambiguity problem, the points were selected leaving a considerable distance between them.

```
H, status = cv.findHomography(p2, p1,cv.RANSAC)
im5_warped = cv.warpPerspective(im5,H,(im1.shape[1] + im5.shape[1], im1.shape[0]))
im5_warped[0:im1.shape[0], 0:im1.shape[1]] = im1
print(H)
```

```
[[ 1.58246972e+00 -7.78141500e-02 -3.56006151e+02]
 [-3.65609064e-01  7.31362708e-01  9.90403605e+01]
 [-8.73765354e-04  1.13260727e-04  1.00000000e+00]]

Out[23]: (-0.5, 1599.5, 639.5, -0.5)
```

Figure 7 appended code

Figure 8 Selected points

Figure 9 Warped image

4) Same points in the image were selected as the previous question. Following code was added to the given code.

```
H = []
for i in range(0, len(p1)):
    x, y = p1[i][0], p1[i][1]
    u, v = p2[i][0], p2[i][1]
    H.append([x, y, 1, 0, 0, 0, -u*x, -u*y, -u])
    H.append([0, 0, 0, x, y, 1, -v*x, -v*y, -v])
H = np.asarray(H)
U, S, Vh = np.linalg.svd(H)
L = Vh[-1,:] / Vh[-1,-1]
H = L.reshape(3, 3)
print(H)

im4_warped = cv.warpPerspective(im4, np.linalg.inv(H), (im1.shape[1]+im4.shape[1],im1.shape[0]))
im4_warped[0:im1.shape[0], 0:im1.shape[1]] = im1
cv.destroyAllWindows()

fig,(ax1,ax2,ax3) = plt.subplots(3,1, figsize=(18,18) )
ax1.imshow(cv.cvtColor(im1,  cv . COLOR_BGR2RGB))
ax1.set_title("image 1")
ax1.axis("off")
ax2.imshow(cv.cvtColor(im4,  cv . COLOR_BGR2RGB))
ax2.set_title("image 4")
ax2.axis("off")
ax3.imshow(cv.cvtColor(im4_warped,  cv . COLOR_BGR2RGB))
ax3.set_title("Image 4 Warped")
ax3.axis("off")
```

```
[[687. 117.]
 [622. 447.]
 [332. 191.]
 [172.  46.]
 [121. 498.]]
[[493. 204.]
 [495. 494.]
 [381. 235.]
 [307.  64.]
 [314. 555.]]
[[ 6.18817402e-01  4.28204928e-02  2.23298585e+02]
 [ 2.36048059e-01  1.12383991e+00 -2.32875427e+01]
 [ 4.88395873e-04 -8.23473970e-05  1.00000000e+00]]
```

Out[5]: (-0.5, 1599.5, 639.5, -0.5)

Figure 10 Added code

The calculated H is similar to the H found in question 2 using the inbuilt function.

Figure 11 Warped Image

5) Stitching using SuperGlue

```
opt = easydict.EasyDict({
    'input_pairs': 'assets/graf_find.txt',
    "input_dir" : 'assets/graf/',
    'output_dir':'matched_pairs/',
    'max_length':-1,'resize':[-1],'resize_float':False,'superglue':'indoor','max_keypoints':1024,
    'keypoint_threshold':0.005,'nms_radius':4,'sinkhorn_iterations':20,'match_threshold':0.2,'viz' :True,
    'eval':False,'fast_viz' : False,'cache':False,'show_keypoints':True,'viz_extension':'png','opencv_display':False,
    'shuffle':False,'force_cpu':True,
})
```

Figure 82 Input parameters in dictionary

```
dumppath = 'matched_pairs/img1_img5_matches.npz'
npz = np.load(dumppath)
npz.files
print(npz['keypoints0'].shape)
print(npz['keypoints1'].shape)
print(npz['matches'].shape)
print(npz['match_confidence'].shape)
print(max(npz['match_confidence']))
npz.files
print(npz['keypoints0'][10:20])
print(npz['keypoints1'][10:20])
```

```
0.95516264
```

Figure 13 putting the strongly matched points of images into arrays and match confidence is given

```
p1=[]
p2=[]
k=0
for i in range(len(npz['matches'])):
    k1=npz['matches'][i]
    if k1 != -1 :
        p1.append(npz['keypoints0'][i])
        p2.append(npz['keypoints1'][k1])

p1=np.asarray(p1)
p2=np.asarray(p2)
print(len(p1))
print(len(p2))
print(p1[:5])
print(p2[:5])
```

```
312
312
```

Figure 14 putting the strongly matched points of images into arrays

```
Hs, status = cv.findHomography(p1, p2,cv.RANSAC,5)
print(Hs)
print(np.linalg.inv(Hs))
```

```
[[ 6.23182856e-01  5.68041238e-02  2.21486504e+02]
 [ 2.18423926e-01  1.15793552e+00 -2.29043805e+01]
 [ 4.84892539e-04 -3.96319331e-05  1.00000000e+00]]
```

Figure 9 find homography

```
im5_warped = cv.warpPerspective(im5,np.linalg.inv(Hs), (10000,10000))
```

Figure 16 warping

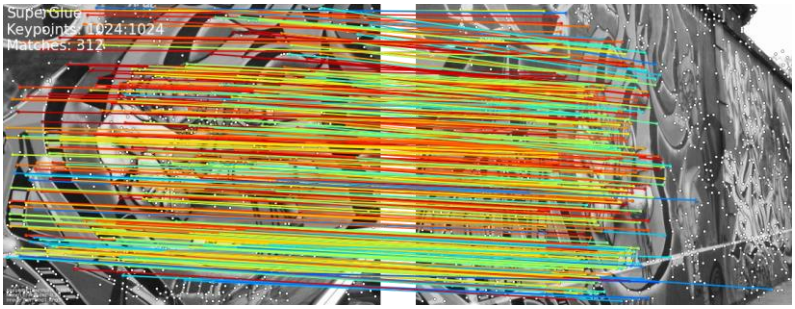

Figure 17 finding matching points using superglue

As shown above using the superglue we find the strongly matched poitns of both images and put them into arrays and compute the homography of that and warp the image using the inbuilt warp function.

Figure 18 img1

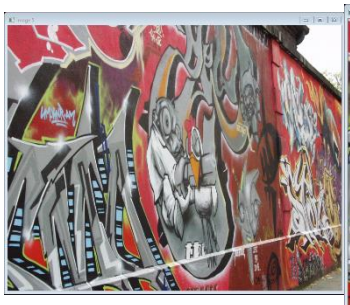

Figure 19 img5

Figure 20 warping img1 and img5

Stitching more images using mouse clicked points

Figure 21 2 images warped using Superglue homography

Here we have stitched image 1 and 5 using mouse clicks and homography calculated using superglue and then the stitched image and img6 are warped and stitched together.

Figure 22 3 images stiched