180427N

Assignment 02

1.

```
H = [[np.cos(t), np.sin(t), 0.], [-np.sin(t), np.cos(t), 0.], [0., 0., 1.]] # Rotataion
H = [[1., 0., 0.5], [0.,1., 0.5], [0., 0., 1.]] # translation
H = [[0.5*np.cos(t), 0.5*np.sin(t), 0.5], [-0.5*np.sin(t), 0.5*np.cos(t), 0.5], [0., 0., 1.]] # similarity
H = [[ 1, 2, 0.5], [ 1 , 0.75, 0.5], [0., 0., 1.]] # Affinity
H = [[ 1, 2, 0.5], [ 1 , 0.75, 0.5], [0.5, 0.75, 1.]] # projetive transformation
```

The above part of the code represents the different homography matrix that I used to experiment with different kinds of 2D transformation. The following figures shows the results of the above transformations.

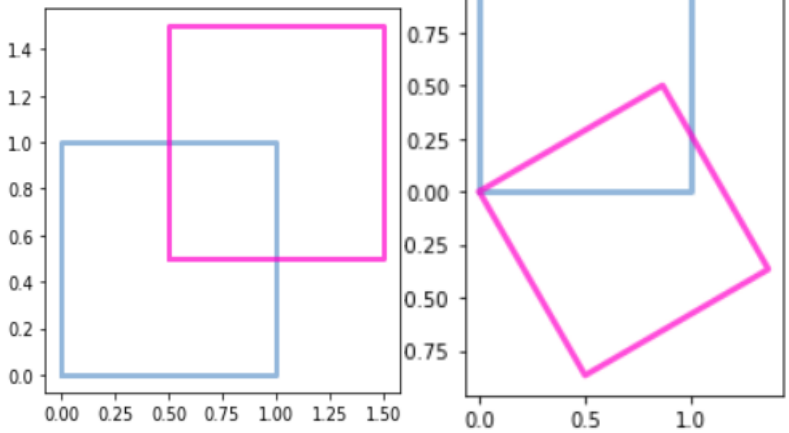


Figure 1: Translation

Figure2: Rotation

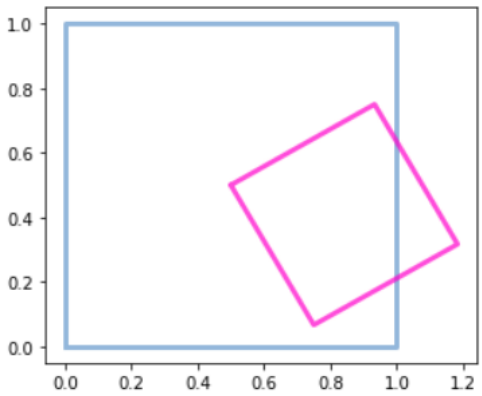


Figure 3: Similarity

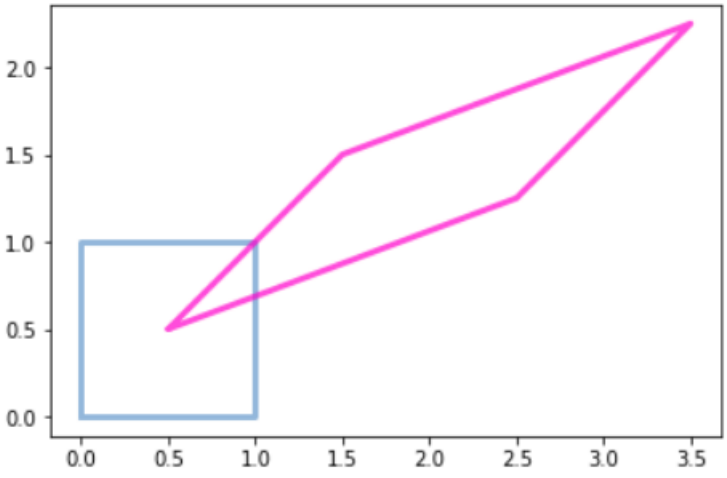


Figure 4 : Affinity

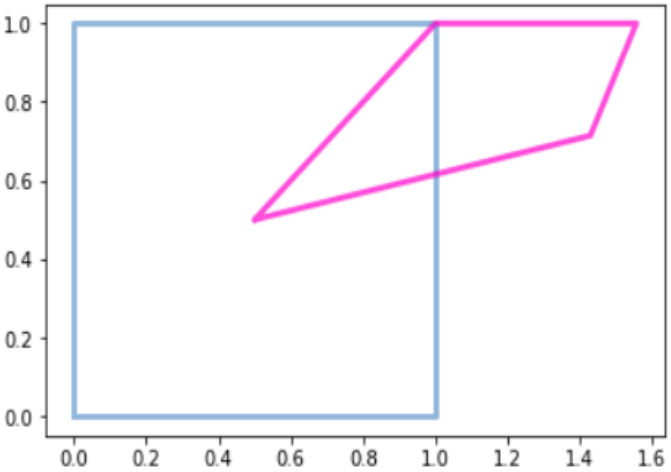


Figure 5: Projective transformation

The below table compares the parameters that are affected by these different transformations.

Transformation	Angle	Area	Parallelism	DOF
Translation	✓	✓	✓	2
Rotation (Euclidean)	✓	✓	✓	3
Similarity	✓	X	✓	4
Affinity	X	X	✓	6
Projective	X	X	X	8

Table 1

✓ : Indicates parameters that are preserved after transformation.

X : Indicates parameters that are not preserved after transformation.

2. Warp perspective code

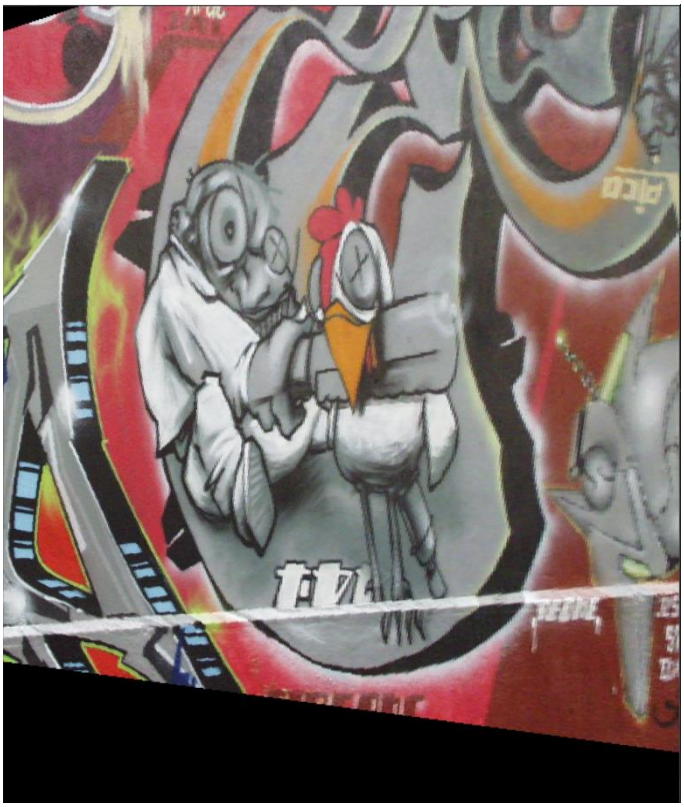
```
def img_to_array(img):
    r, c, ch = img.shape
    im_array = np.zeros((c,r, ch), dtype= img.dtype)
    for i in range(r):
        im_array[:,i] = img[i]
    return im_array

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
```

```
im5_warped  = warpPerspective(im5,np.linalg.inv(H), (1000,1000))
```



Img1 has been taken as the front view of the wall but img5 has been taken as a side view of the wall. We want to transform img1 onto img5 that means we need front view of img5. For this purpose, I used the above code, and I obtained the resulting image as I expected that is a front view of img5. That is shown in figure 6, where we can notice now the white line is properly transformed. Further, we can verify this transformation by stitching those images that is done in coming questions.

Figure 6 : Transformed img1 onto img5

3.

cv.getPerspectiveTransform (pt1, p2) : computes homography matrix

```
pt1 = np.float32([p1[0], p1[1], p1[2], p1[3]])
pt2 = np.float32([p2[0], p2[1], p2[2], p2[3]])
H = cv.getPerspectiveTransform(pt1, pt2)
print(H)
```

```
[[ 6.35026463e-01  4.96122725e-02  2.21313925e+02]
 [ 2.24266601e-01  1.14989898e+00 -2.21108978e+01]
 [ 4.94314936e-04 -4.82531227e-05  1.00000000e+00]]
```

Computed homography

```
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
```

Given homography.

Computed homography using mouse click is pretty much similar to the given homography except slight variation in decimal points. when we mark corresponded points using mouse click always there is a chance to mark incorrectly. This is the reason for that small variation between computed and given holographic matrices.

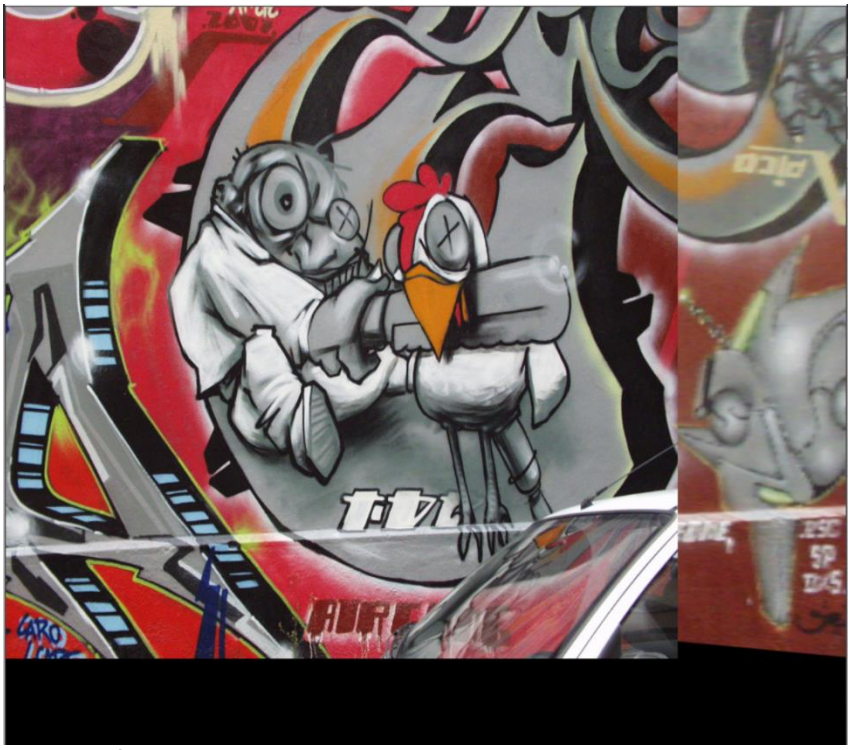


Figure 7

Figure 7 is the image obtained after stitching img1 with img5. These images are stitched as we excepted. We can observe there is a white line in both images that is properly stitched whereas the curve is not properly stitched due to the small variation in computed homography. We can get an even better result by choosing corresponded points precisely.

4.

First Img1 is stitched with img3 and then the resulting image is stitched with img5. Figure 8 is the final image after stitching img1, img3 and img5. In that we can observe the white straight line, face (top middle) and curve shape are properly stitched as we expected. For this transformation, we need two homography matrices, one for img1 to img3 and one for the resulted image of img1 and img3 to img5. These matrices are shown below.

```
[[ 7.45096697e-01 -3.02842522e-01  2.31330716e+02]
 [ 3.21218487e-01  9.93973009e-01 -6.94788058e+01]
 [ 3.32589718e-04 -3.96286579e-05  1.00000000e+00]]
```

H – img1 to img3

```
[[ 6.68256165e-01  7.89104289e-02  2.16566570e+02]
 [ 2.56504839e-01  1.21489283e+00 -3.64501001e+01]
 [ 5.60292630e-04  1.02995633e-05  1.00000000e+00]]
```

H – result of img1 and img3 to img5

homography_mat function returns the homography matrix.

```
def homography_mat(orgP, transP):
    homo = np.zeros((3,3), dtype="float32")
    coe_mat = []
    b = []
    for i in range(8):
        k = [0]*8
        if i < 4:
            x,y = orgP[i]
            u,v = transP[i]
            k[0:3] = [x,y,1]
            k[-2:] = -x*u, -y*u
            b.append(u)
        else:
            x,y = orgP[abs(4 - i)]
            u,v = transP[abs(4-i)]
            k[3:6] = [x,y,1]
            k[-2:] = -x*v, -y*v
            b.append(v)
        coe_mat.append(k)
    x = np.linalg.solve(np.array(coe_mat), np.array(b))
    x.resize((9,))
    x[8] = 1
    return x.reshape((3,3))
```

Further, to obtain stitched image I used different sizes,

1. (1000, 1000) for img1 and img3
2. (1500, 1500) for resulted image of first stitching and img5.

Figure 8: img1, img3 and img5 stitched together.