Camera Calibration & Homography Transformation

106034061
曾靖渝

1. Part 1. Camera Calibration (50%)
    a. Compute the projection matrix from a set of 2D-3D point correspondences by using the leastsquares (eigenvector) method for each image.

```
def Compute_Projection_Matrix():
    global Points_2D, index
    global A,Eigen_Values, Eigen_Vectors
    r, c = Points_3D.shape
    A = np.zeros((r*2, 12))
    for i in range(r):
        A[i*2, 0:3] = Points_3D[i, :]
        A[i*2, 3] = 1
        A[i*2, 8:-1] = -Points_2D[i][0] * Points_3D[i]
        A[i*2, -1]= -Points_2D[i][0]
        A[i*2+1, 4:7] = Points_3D[i,:]
        A[i*2+1, 7] = 1
        A[i*2+1, 8:-1] = -Points_2D[i][1] * Points_3D[i]
        A[i*2 + 1, -1] = -Points_2D[i][1]

    Eigen_Values, Eigen_Vectors =
        np.linalg.eig(A.transpose().dot(A))
    Projection_Matrix = Eigen_Vectors[:, np.argmin(Eigen_Values)]
    Projection_Matrix = Projection_Matrix.reshape((3,4))
    print("Projection_Matrix");print(Projection_Matrix)
```

Compute Projection Matrix by Eigenvalue.

```
Projection_Matrix
[[-1.16564570e-01  2.09946969e-02  3.37902294e-02 -2.78485085e-01]
 [ 1.17545741e-02 -2.08441821e-02  1.38412648e-01 -9.42091096e-01]
 [ 2.24311446e-06  1.92988487e-04  1.57477356e-04 -3.12967998e-03]]
```

```
Projection_Matrix
[[-9.86925890e-02 -3.93348474e-02  1.96557696e-02 -5.07065303e-01]
 [ 1.71852533e-02 -1.55990031e-02  1.07917703e-01 -8.47949326e-01]
 [-7.33883230e-05  1.45863903e-04  8.12698661e-05 -2.82621400e-03]]
```

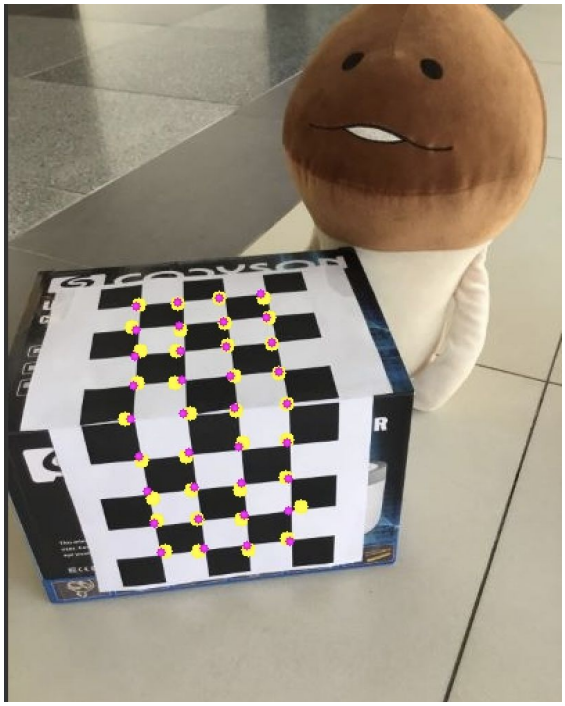
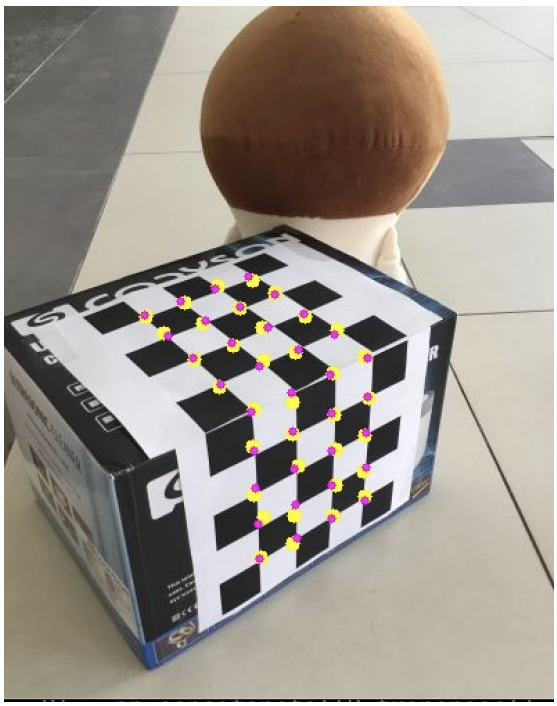
    b. Decompose the two computed projection matrices from (A) into the camera intrinsic matrices K, rotation matrices R and translation vectors t by using the

Gram-Schmidt process. Any QR decomposition functions are allowed. The bottom right corner of intrinsic matrix K should be normalized to 1. Also, the focal length in K should be positive.

```
def Decompose_into_KRT(Projection_Matrix):
    if np.linalg.det(Projection_Matrix[:,:-1])<0:
        Projection_Matrix *= -1
    K, R = scipy.linalg.rq(Projection_Matrix[:,:-1])
    D = np.zeros((3, 3))
    for i in range(3):
        D[i, i] = np.sign(K[i][i])
#    print("D");print(D)
    K = K.dot(D)
#    print("K");print(K)
    T = np.linalg.inv(K).dot(Projection_Matrix[:, -1])
    T=T[..., None]
    R = D.dot(R)
    K /= K[-1, -1]
#    print("K") ;print(K)
#    print("R") ;print(R)
#    print("T") ;print(T)
    return K,R,T
```

Decompose Projection Matrix into K, R, T Matrix.

c. Re-project 2D points on each of the chessboard images by using the computed intrinsic matrix, rotation matrix and translation vector. Show the results (2 images) and compute the point reprojection root-mean-squared errors.

Reprojection of 2D points

```
RMS
3.5664534282149054
```

```
RMS
3.459348852058409
```

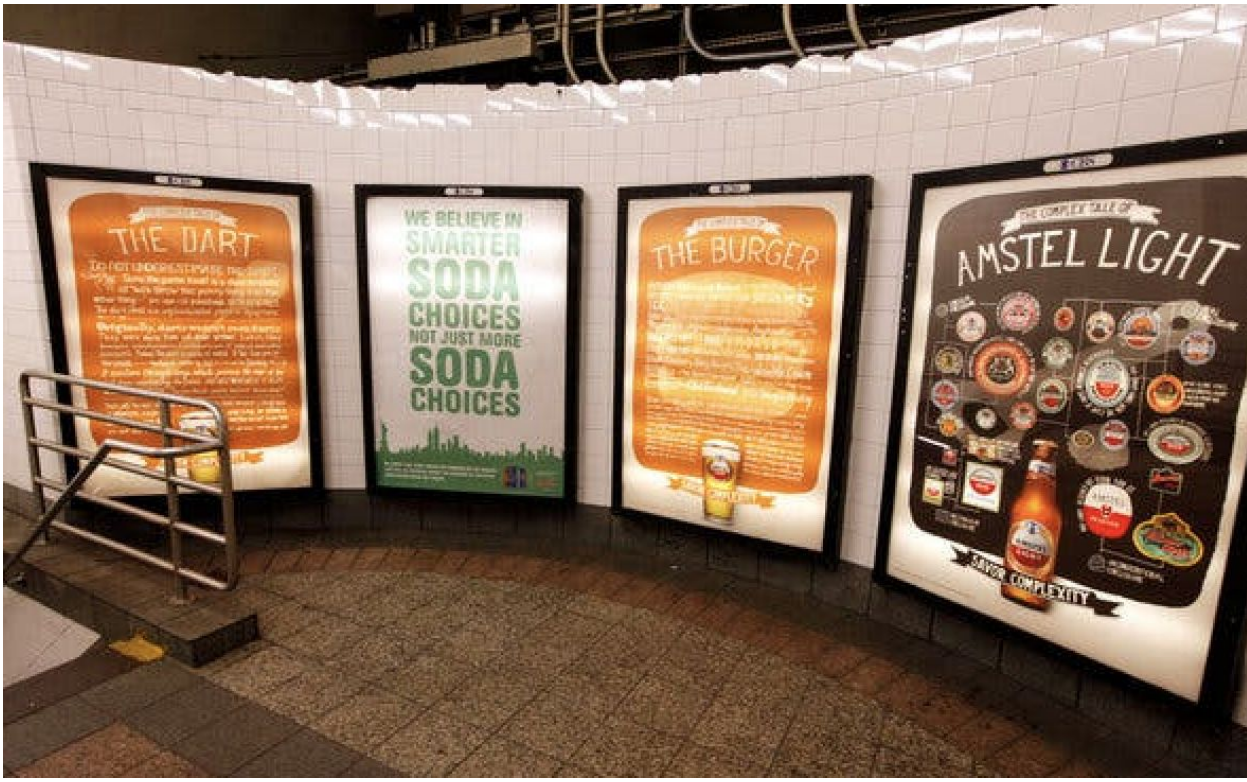
d. Plot camera poses for the computed extrinsic parameters (R, t) and then compute the angle between the two camera pose vectors.

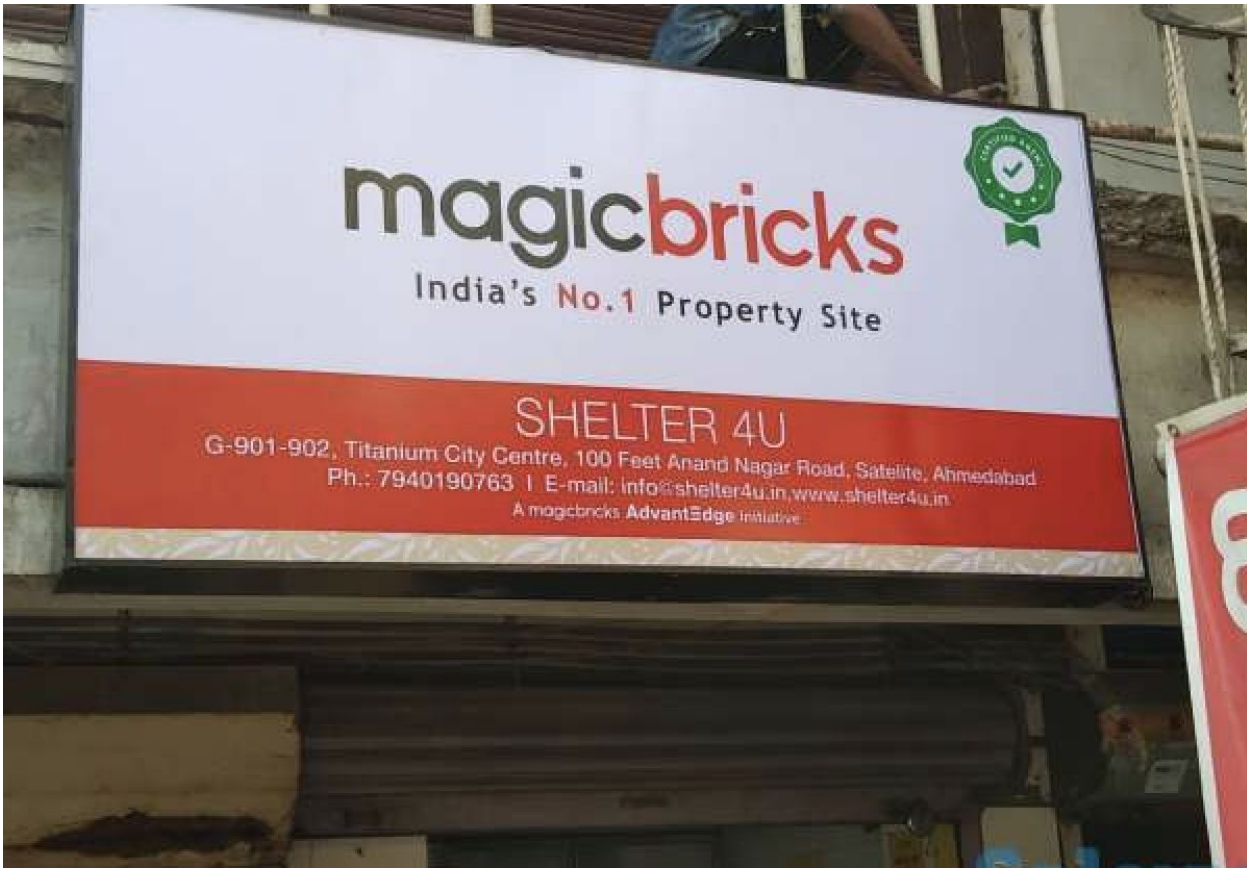
```
def Compute_Camera_Position(R, T):
    return R.transpose().dot(T)
```

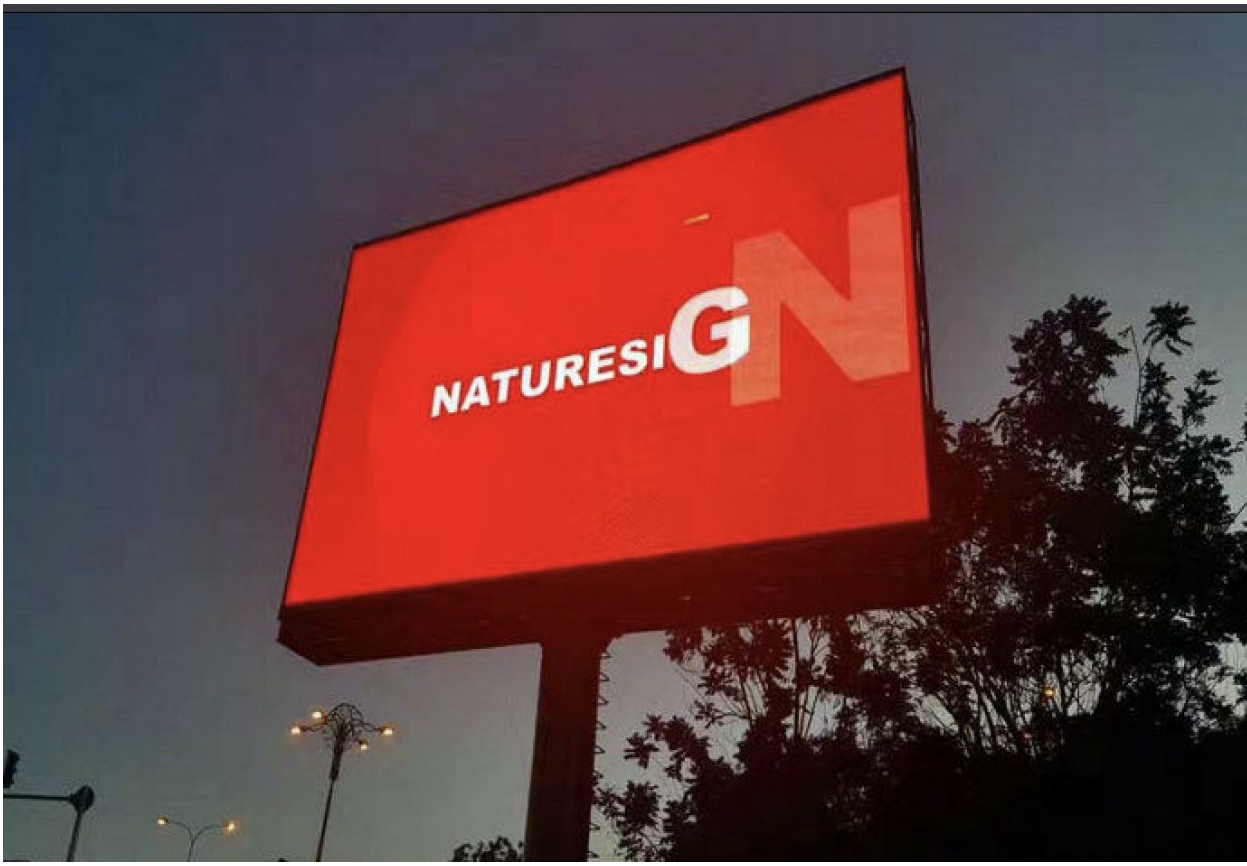
```
Camera Position
[[-1.68672409]
 [-9.5826743 ]
 [-8.10624892]]
```

```
Camera Position
[[  7.05314763]
 [-10.01673133]
 [-10.4284139 ]]
```

2. Part 2. Homography transformation (50%)
    a. Shoot three images A, B and C. Image A has to contain two objects. Image B and C should contain one object separately. Like the images shown above. (3 images)

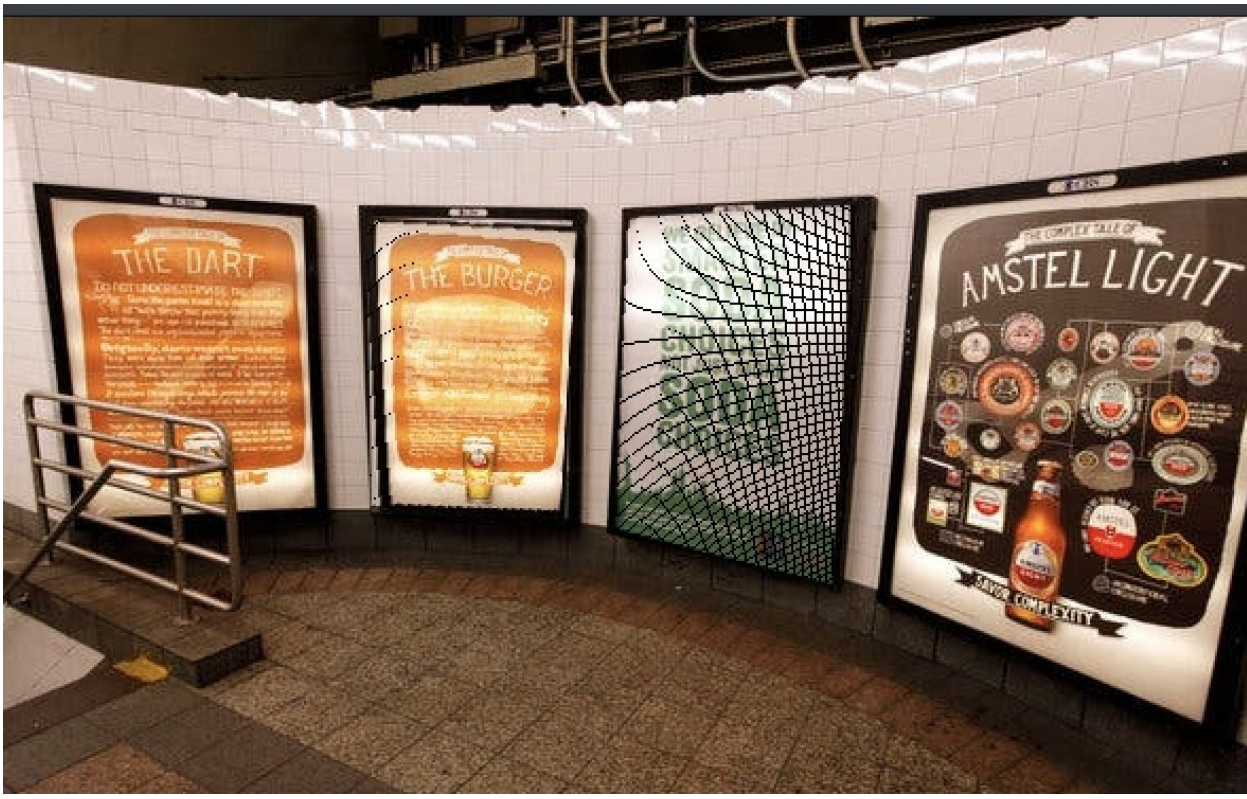

THE DART
WE BELIEVE IN
SMARTER
SODA
CHOICES
NOT JUST MORE
SODA
CHOICES
THE BURGER
AMSTEL LIGHT


magicbricks
India's No.1 Property Site
SHELTER 4U
G-901-902, Titanium City Centre, 100 Feet Anand Nagar Road, Satelite, Ahmedabad
Ph.: 7940190763 I E-mail: info@shelter4u.in, www.shelter4u.in
A magicbricks AdvantEdge Initiative

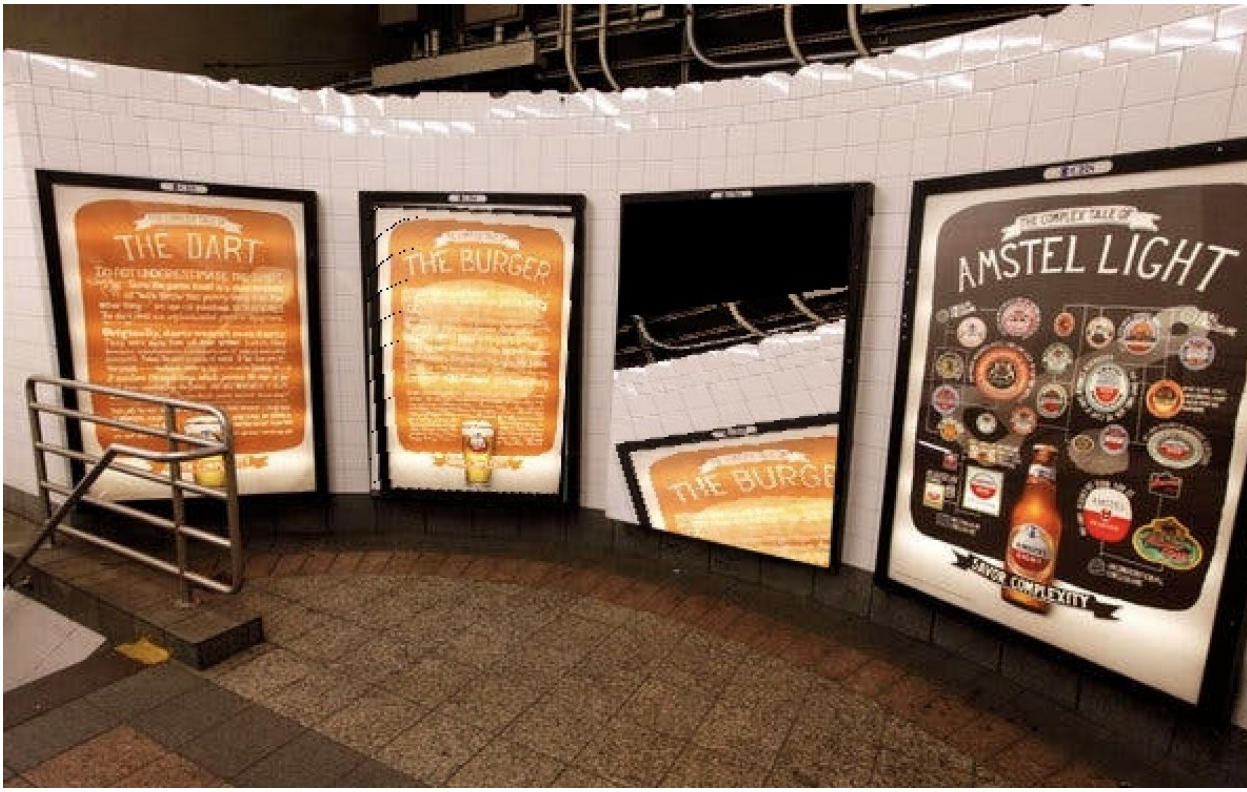


b. Compute homography transformation between the two objects in image A. Use both backward and forward warping to switch them, like what example 1 shows.

(2 images)



Forward Warping



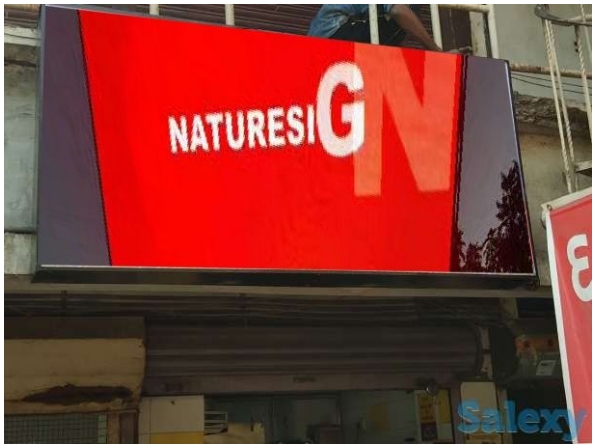
Backward Warping

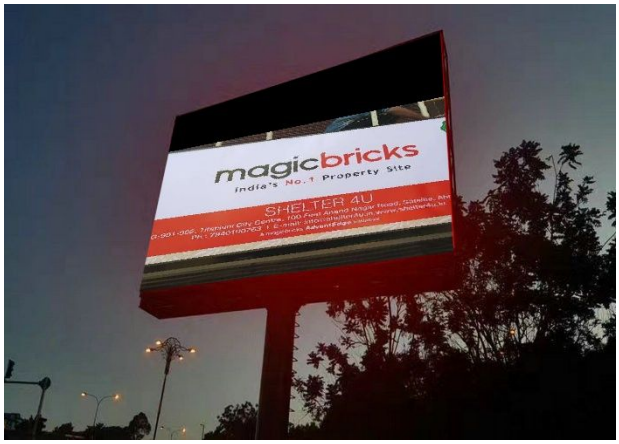
c. Compute homography transformation between the object in image B and the object in image
   Same method which used in hw1-1

```python
def Compute_Projection_Matrix(pts_src, pts_dst):#src::3D, ds::2D
    r, c = pts_src.shape
    A = np.zeros((r*2, 8))
    for i in range(r):
        A[i*2, 0:2] = pts_src[i, :]
        A[i*2, 2] = 1
        A[i*2, 6:] = -pts_dst[i][0] * pts_src[i]
#         A[i*2, -1]= -pts_dst[i][0]
        A[i*2+1, 3:5] = pts_src[i,:]
        A[i*2+1, 5] = 1
        A[i*2+1, 6:] = -pts_dst[i][1] * pts_src[i]
#         A[i*2 + 1, -1] = -pts_dst[i][1]
    A_pt = np.linalg.pinv(A)
    B = pts_dst.reshape((8,1))
    Projection_Matrix = A_pt.dot(B)
    Projection_Matrix = np.append(Projection_Matrix, [1])
    Projection_Matrix = Projection_Matrix.reshape((3,3))
    print("Projection_Matrix");print(Projection_Matrix)
    return Projection_Matrix
```

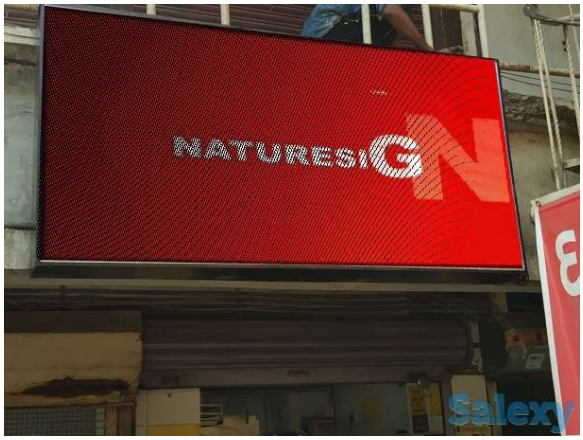
```
Projection_Matrix
[[ 6.05811363e+00  1.21887919e+00 -1.25758572e+03]
 [ 1.79587136e+00  3.54069048e+00 -8.09689585e+02]
 [ 4.84638694e-03  1.83599666e-03  1.00000000e+00]]
```

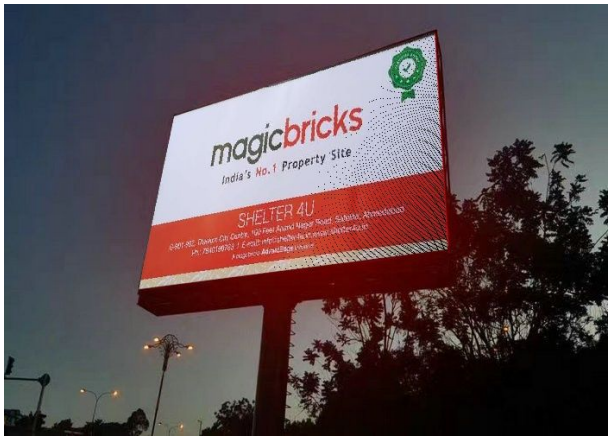
d. Use both backward and forward warping to switch them. Example 2 gives some illustration. (4 images)





Backward Warping





Forward Warping

e. Discuss the difference between forward and backward warping based on your results

   i. Forward Warping : pretty good results, but there are several pixels with wrong color, it may be caused by the transformation of integer

```
def Forward_Warpping(img, M, dsize):
    mtr = to_mtx(img)
    R,C = dsize
    dst = np.zeros((R,C,mtr.shape[2]))
    for i in range(mtr.shape[0]):
        for j in range(mtr.shape[1]):
            res = np.dot(M, [i,j,1])
            i2,j2,_ = (res / res[2] + 0.5).astype(int)
            if i2 >= 0 and i2 < R:
                if j2 >= 0 and j2 < C:
                    dst[i2,j2] = mtr[i,j]
    return to_img(dst)
```

   ii. Backward Warping:I have tried several way to implement , but it did not work very well. Can TA provide solution for reference.

```
def Backward_Warpping(img, M, dsize):
    mtr = (img)
    C,R = dsize
    dst = np.zeros((R,C,mtr.shape[2]))
#     print("M");print(M)
    invM = np.linalg.inv(M)
#     print("invM");print(invM)
    for i in range(R):
        for j in range(C):
            res = np.dot(invM, [i,j,1])
            i2,j2,_ = (res / res[2] + 0.5).astype(int)
            if i2 >= 0 and i2 < mtr.shape[0]:
                if j2 >= 0 and j2 < mtr.shape[1]:
                    dst[i,j] = mtr[i2,j2]
    return dst.astype(np.uint8)
```