Computer Vision: from Recognition to Geometry
HW3

Name: ___楊樂多___ Department: __網媒所___ Student ID: _R06944038_

Part 1: Estimating Homography

Implement solution 1 for estimating homography

Input images

Target surface

Result

function solve_homography(u, v)

```python
def solve_homography(u, v):
    N = u.shape[0]
    if v.shape[0] is not N:
        print('u and v should have the same size')
        return None
    if N < 4:
        print('At least 4 points should be given')
    A = np.zeros((2*N, 8))
    b = np.zeros((2*N, 1))
    A[0:A.shape[0]:2,2] = 1
    A[1:A.shape[0]:2,5] = 1
    for n in range(N):
        r = n * 2
        A[r,0] = u[n][0]
        A[r,1] = u[n][1]
        A[r,6] = -1*(u[n][0]*v[n][0])
        A[r,7] = -1*(u[n][1]*v[n][0])
        r = r+1
        A[r,3] = u[n][0]
        A[r,4] = u[n][1]
        A[r,6] = -1*(u[n][0]*v[n][1])
        A[r,7] = -1*(u[n][1]*v[n][1])
    for n in range(N):
        r = n * 2
        b[r,0] = v[n][0]
        r = r + 1
        b[r,0] = v[n][1]

    X = np.linalg.solve(A, b)
    H = np.zeros((3, 3))
    H[2,2] = 1
    for x in range(len(X)) :
        r = x // 3
        c = x % 3
        H[r,c] = X[x]
    return H
```

Estimate the 3*3 homography matrix H by solution 1

Solution 1 (cont.)

- Solve linear system

$$
\begin{array}{l c c c}
 & 2N \times 8 & 8 \times 1 & 2N \times 1 \\
\begin{array}{l} \textbf{Point 1} \\ \\ \textbf{Point 2} \\ \\ \textbf{Point 3} \\ \\ \textbf{Point 4} \\ \\ \end{array} &
\begin{bmatrix}
u_{x,1} & u_{y,1} & 1 & 0 & 0 & 0 & -u_{x,1}v_{x,1} & -u_{y,1}v_{x,1} \\
0 & 0 & 0 & u_{x,1} & u_{y,1} & 1 & -u_{x,1}v_{y,1} & -u_{y,1}v_{y,1} \\
u_{x,2} & u_{y,2} & 1 & 0 & 0 & 0 & -u_{x,2}v_{x,2} & -u_{y,2}v_{x,2} \\
0 & 0 & 0 & u_{x,2} & u_{y,2} & 1 & -u_{x,2}v_{y,2} & -u_{y,2}v_{y,2} \\
u_{x,3} & u_{y,3} & 1 & 0 & 0 & 0 & -u_{x,3}v_{x,3} & -u_{y,3}v_{x,3} \\
0 & 0 & 0 & u_{x,3} & u_{y,3} & 1 & -u_{x,3}v_{y,3} & -u_{y,3}v_{y,3} \\
u_{x,4} & u_{y,4} & 1 & 0 & 0 & 0 & -u_{x,4}v_{x,4} & -u_{y,4}v_{x,4} \\
0 & 0 & 0 & u_{x,4} & u_{y,4} & 1 & -u_{x,4}v_{y,4} & -u_{y,4}v_{y,4}
\end{bmatrix} &
\begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = &
\begin{bmatrix} v_{x,1} \\ v_{y,1} \\ v_{x,2} \\ v_{y,2} \\ v_{x,3} \\ v_{y,3} \\ v_{x,4} \\ v_{y,4} \end{bmatrix}
\end{array}
$$

given the original corners of the input images and the corresponding position of the images in target surface

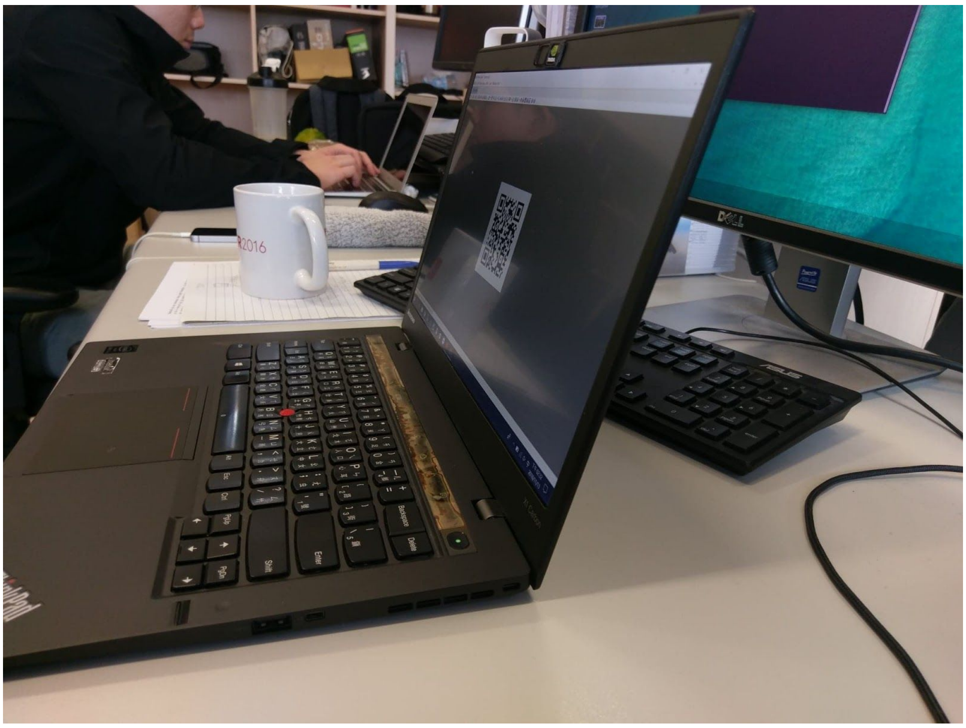
1.Build matrix A,vector b according to the solution 1 formula in slide p.7

2.solve the linear system with numpy linalg.solve function

3.return matrix H which represents the homography matrix

Part 2: Unwrap the Screen

Input image



Position of the QR code in input image

```
# order : 左上,左下,右上,右下
tl,bl,tr,br = [1038,365],[980,553],[1106,393],[1041,594]
```

solved with function solve_homography(u, v) mentioned above in part 1

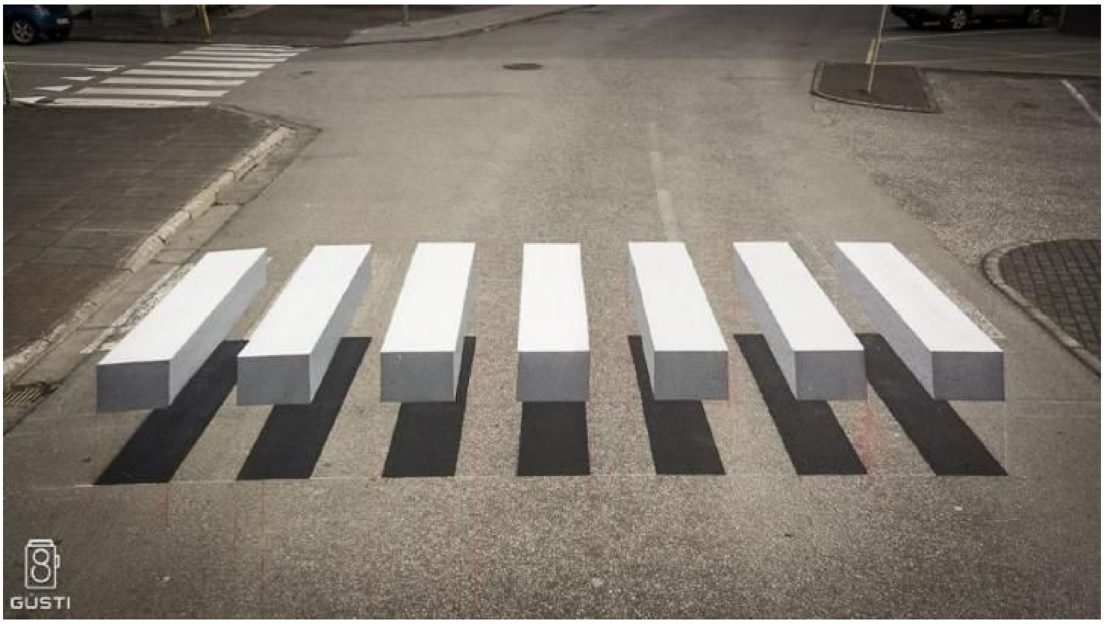
Extracted QR code

decoded link : http://media.ee.ntu.edu.tw/courses/cv/18F/

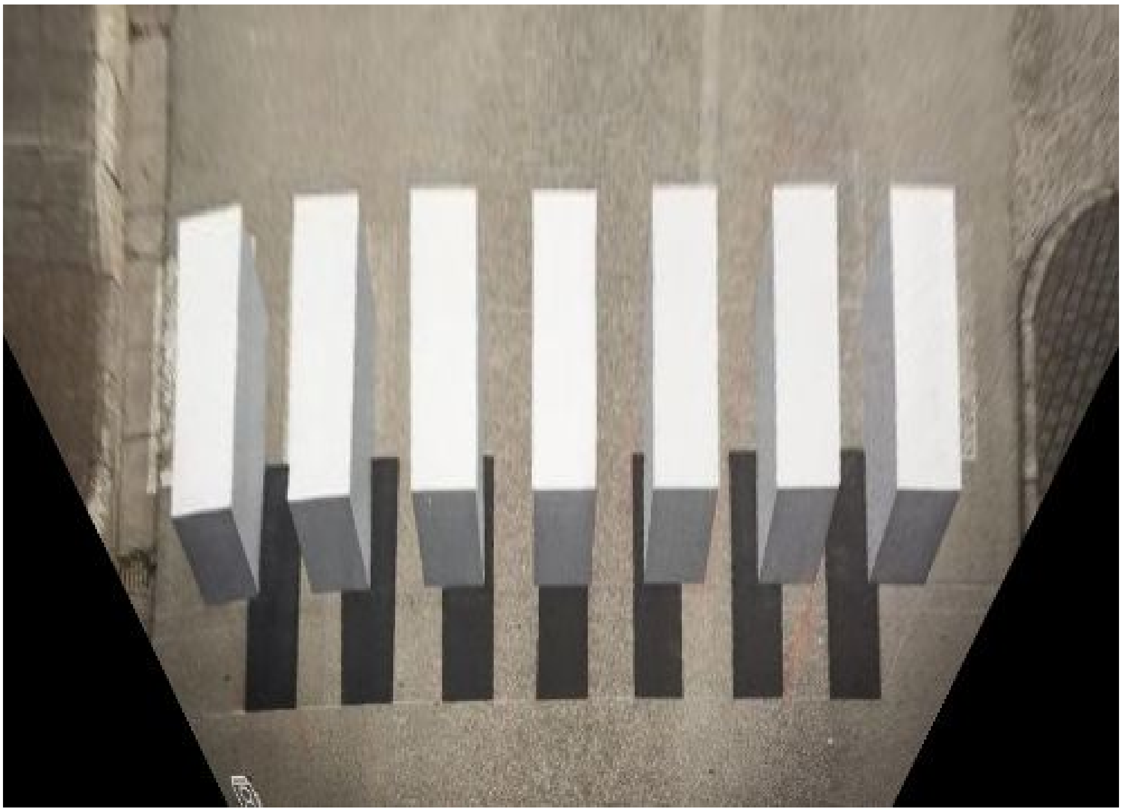
Part 3: Unwarp the 3D Illusion

solved with opencv projective transform function getPerspectiveTransform given the corners of several combination of the bars

Input image

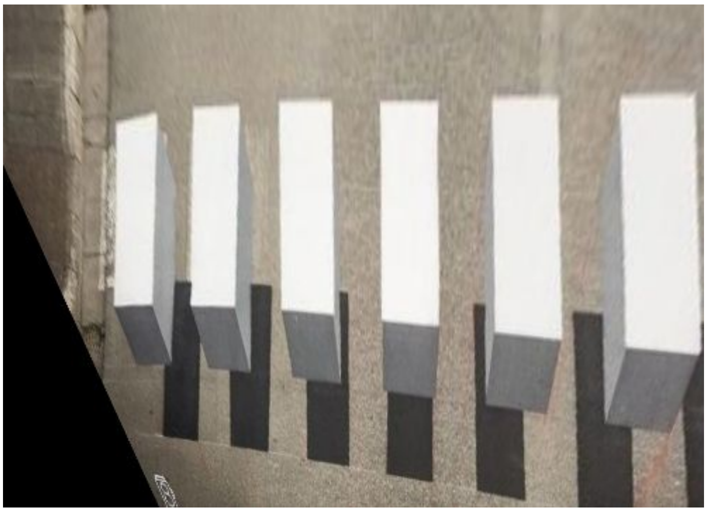
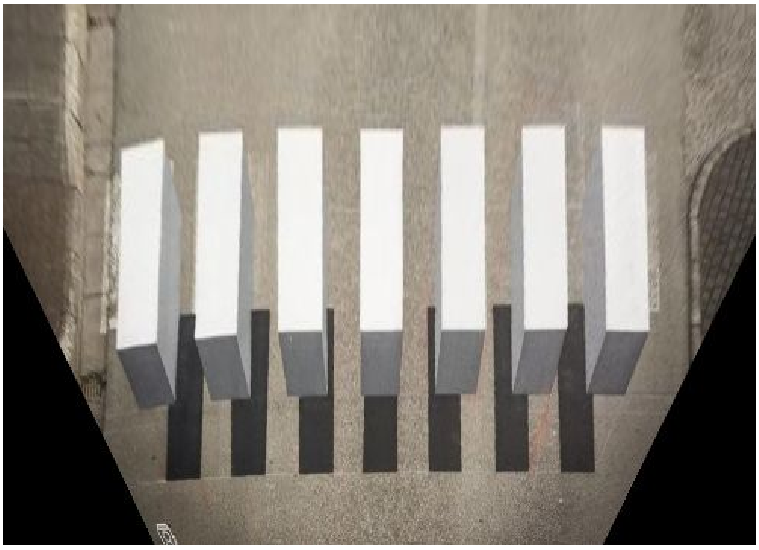


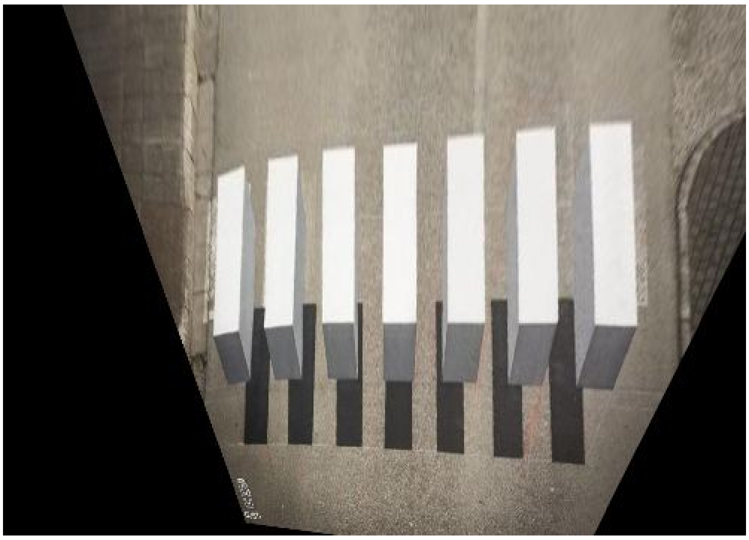
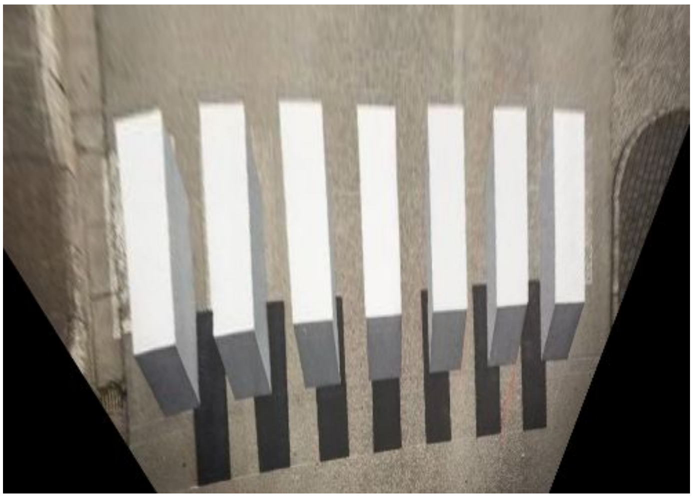
Result



Can you get the parallel bars from the top view? If not, why? Discuss in your report.
轉換到top-view時，期待每條bar彼此間互相平行，但由於每條bar在原圖中傾斜的方向、角度不同，使得在轉換之後並無法讓每條bar之間都完美的平行(i.e.當某些bars平行時，必會存在某些bars無法完美維持平行)。

實作上利用opencv getPerspectiveTransform(取得轉換之matrix)及warpPerspective function(將圖片依據轉換之matrix做變形/投影)來完成，並且嘗試以下幾種不同對應點組合(實驗後發現以中間的bar作為對應點下的轉換有最好之結果)：

對應點組合	轉換結果
以第一條bar之corners作為對應點 (第一條bar傾斜之角度與多數bar不同，因此以此為對應點轉換後效果較差)	
以中間的bar之corners作為對應點 (中間的bar傾斜之角度與多數bar較為相似，因此以此為對應點轉換後效果較其他設計更佳)	

以最後一條bar之corners作為對應點	
以第一、中間、最後條bars之corners分別作為對應點後平均	
以包含所有bars之範圍的corners作為對應點	

各條bar之對應點

```
#first
tl,bl,tr,br = [136,162],[64,270],[174,162],[109,265]
org_corners1  = np.array([tl,bl,tr,br],np.float32)
ref_corners1 = np.array([[80, 80],[96, 254],[107,80],[119,252]],np.float32)
#middle
tl,bl,tr,br = [345,157],[340,260],[381,157],[386,260]
org_corners2  = np.array([tl,bl,tr,br],np.float32)
ref_corners2 = np.array([[237, 80],[237, 254],[265,80],[262,254]],np.float32)
#last
tl,bl,tr,br = [552,157],[616,260],[588,158],[661,260]
org_corners3  = np.array([tl,bl,tr,br],np.float32)
ref_corners3 = np.array([[395, 80],[385, 254],[422,80],[409,255]],np.float32)
```