Assignment 3

ID: R06945011, NAME: Shih-Jou Chung(鍾詩柔), Department: BEBI

1. part 1

   code:

```
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/2 < 4:
        print('At least 4 points should be given')
    A = np.array([[u[0],u[1],1,0,0,0,-u[0]*v[0],-u[1]*v[0],-v[0]],
                  [0,0,0,u[0],u[1],1,-u[0]*v[1],-u[1]*v[1],-v[1]],
                  [u[2],u[3],1,0,0,0,-u[2]*v[2],-u[3]*v[2],-v[2]],
                  [0,0,0,u[2],u[3],1,-u[2]*v[3],-u[3]*v[3],-v[3]],
                  [u[4],u[5],1,0,0,0,-u[4]*v[4],-u[5]*v[4],-v[4]],
                  [0,0,0,u[4],u[5],1,-u[4]*v[5],-u[5]*v[5],-v[5]],
                  [u[6],u[7],1,0,0,0,-u[6]*v[6],-u[7]*v[6],-v[6]],
                  [0,0,0,u[6],u[7],1,-u[6]*v[7],-u[7]*v[7],-v[7]]]).astype(float)
    a = np.dot(A.T, A)
    eigValue, eigVect= np.linalg.eig(a)
    value,index= find_nearest( eigValue, 0 )
    H = np.reshape(eigVect[:,index],(3,3))
    return H
```

利用 Recap of Homography 原理，如式(1)，U 為原本圖像 N 點矩陣，H 為轉至過去的 Vector，V 為後來轉製成的圖像 N 點矩陣，U 和 N 為已知，必須尋找 H 矩陣。經過課本上的換算之後 U 矩陣轉換成 A 矩陣，而 V 矩陣則為 0，如式(2)。

A 矩陣中必須找到四個點以上，這邊使用四個點即可，所以為 8*9 矩陣，如式(3)，因 H 矩陣必須為 orthogonal，所以得先將 A 矩陣 Orthogonal 之後得到 EigenVector，EigenVector 需取 EigenValue 最接近零的 Vector，得到後就為 H(9*1)，如式(4)。

$$U*H = V \quad (1)$$

$$A*H = 0 \quad (2)$$

$$A = \begin{bmatrix} u_{x1} & u_{y1} & 1 & 0 & 0 & 0 & -u_{x1} * v_{x1} & -u_{y1} * v_{x1} & -v_{x1} \\ 0 & 0 & 0 & u_{x1} & u_{y1} & 1 & -u_{x1} * v_{y1} & -u_{y1} * v_{y1} & -v_{y1} \\ u_{x2} & u_{y2} & 1 & 0 & 0 & 0 & -u_{x2} * v_{x2} & -u_{y2} * v_{x2} & -v_{x2} \\ 0 & 0 & 0 & u_{x2} & u_{y2} & 1 & -u_{x2} * v_{y2} & -u_{y2} * v_{y2} & -v_{y2} \\ u_{x3} & u_{y3} & 1 & 0 & 0 & 0 & -u_{x3} * v_{x3} & -u_{y3} * v_{x3} & -v_{x3} \\ 0 & 0 & 0 & u_{x3} & u_{y3} & 1 & -u_{x3} * v_{y3} & -u_{y3} * v_{y3} & -v_{y3} \\ u_{x4} & u_{y4} & 1 & 0 & 0 & 0 & -u_{x4} * v_{x4} & -u_{y4} * v_{x4} & -v_{x4} \\ 0 & 0 & 0 & u_{x4} & u_{y4} & 1 & -u_{x4} * v_{y4} & -u_{y4} * v_{y4} & -v_{y4} \end{bmatrix} \quad (3)$$

$$H = \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} \quad (4)$$

將得到的 H 轉成 3*3 矩陣後，將原本圖像的 U 座標經過 H 轉換後轉成 V 座標後，得到的 V 座標的顏色值貼到 canvas 圖上。

共有五張圖，第一張圖抓 width 和 height 之後 U 點分別為[0,0], [width,0], [0,height], [width,height];V 點為要標到的點 corners1，在利用 A 矩陣找出各別 H 矩陣 Vector，後將原本本的座標值貼到 canvas 圖上，其他皆以此類推。

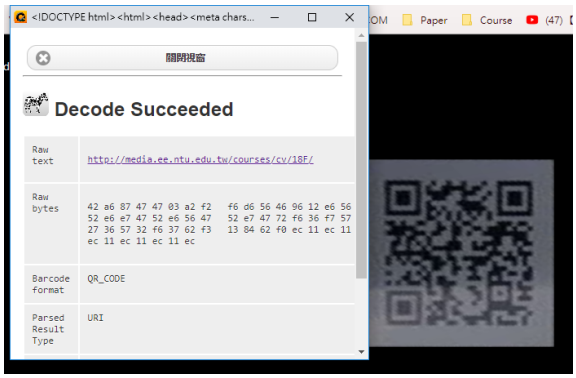
output:

2. part 2 :

U 座標為自己設定長、寬(canvas)，V 在助教給予的圖像找到後設為 corners (img)，之後利用 U 找到 A 矩陣後再找到 H 矩陣，將得到的 V 座標值貼到 U 矩陣上得到 QR code，然而會產生邊界尖銳現象，所以利用 neighbor filter 模糊化得到下面可以利用電腦 QR code 掃描的圖像。

Output:

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



3. part 3:

我將在功課講義上的 top 道路影像擷取下來後取點代表 U(圖一中四顆藍點)，原本給予的 input 代表 V(圖二中四顆橘點)，之後運算 H 矩陣並擴及到旁邊的陰影，得到以下結果。影像大致符合從上往下看的影像，但是因為是從原本的影像轉過來，而原本的影像在四邊就有彎曲，導致轉過來的影像並沒有非常直線，然而下面的陰影處能完整的互相平行。

如果每條斑馬線都各別取四點並投射到需投射的點，則能各別平行，且得到不錯的結果，但本次只使用四點得到的影像就會導致結果如圖三。

Original input:

圖一

圖二

轉至的結果並不能整個還原像原本的圖像旁邊的街道一般，是因為旁邊的 pixel 位置在利用 H 矩陣後會跑出原本的圖像 pixel。

Output: