Computer Vision Homwork2 Report

111062569 葉政賢

1. In this problem, you will implement both the linear least-squares version of the eight-point algorithm and its normalized version to estimate the fundamental matrices. You will implement the methods and complete the following:

(a) Implement the linear least-squares eight-point algorithm and report the returned fundamental matrix. Remember to enforce the rank-two constraint for the fundamental matrix via singular value decomposition.

Theory

Funfamenta; matrix 定義為 $\mathbf{F} = \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix}$，能夠使得兩幅圖片的點滿足：

$$\mathbf{x}^{\mathrm{T}}\mathbf{F}\mathbf{x'} = 0$$

假設 $\mathbf{x}=(u,v,1)^{\mathsf{T}}$， $\mathbf{x}'=(u',v',1)^{\mathsf{T}}$，則會使得 $(uu', uv', u, vu', vv', v, u', v', 1)\begin{pmatrix} F_{11} \\ F_{12} \\ F_{13} \\ F_{21} \\ F_{22} \\ F_{23} \\ F_{31} \\ F_{32} \\ F_{33} \end{pmatrix} = 0$ 。此時，將 F_3^3 設為 1，並滿足 $|\mathcal{F}|^2 = 1$ 的 constraint，則

$$\begin{pmatrix} u_1u_1' & u_1v_1' & u_1 & v_1u_1' & v_1v_1' & v_1 & u_1' & v_1' \\ u_2u_2' & u_2v_2' & u_2 & v_2u_2' & v_2v_2' & v_2 & u_2' & v_2' \\ u_3u_3' & u_3v_3' & u_3 & v_3u_3' & v_3v_3' & v_3 & u_3' & v_3' \\ u_4u_4' & u_4v_4' & u_4 & v_4u_4' & v_4v_4' & v_4 & u_4' & v_4' \\ u_5u_5' & u_5v_5' & u_5 & v_5u_5' & v_5v_5' & v_5 & u_5' & v_5' \\ u_6u_6' & u_6v_6' & u_6 & v_6u_6' & v_6v_6' & v_6 & u_6' & v_6' \\ u_7u_7' & u_7v_7' & u_7 & v_7u_7' & v_7v_7' & v_7 & u_7' & v_7' \\ u_8u_8' & u_8v_8' & u_8 & v_8u_8' & v_8v_8' & v_8 & u_8' & v_8' \end{pmatrix} \begin{pmatrix} F_{11} \\ F_{12} \\ F_{13} \\ F_{21} \\ F_{22} \\ F_{23} \\ F_{31} \\ F_{32} \end{pmatrix} = - \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Implement

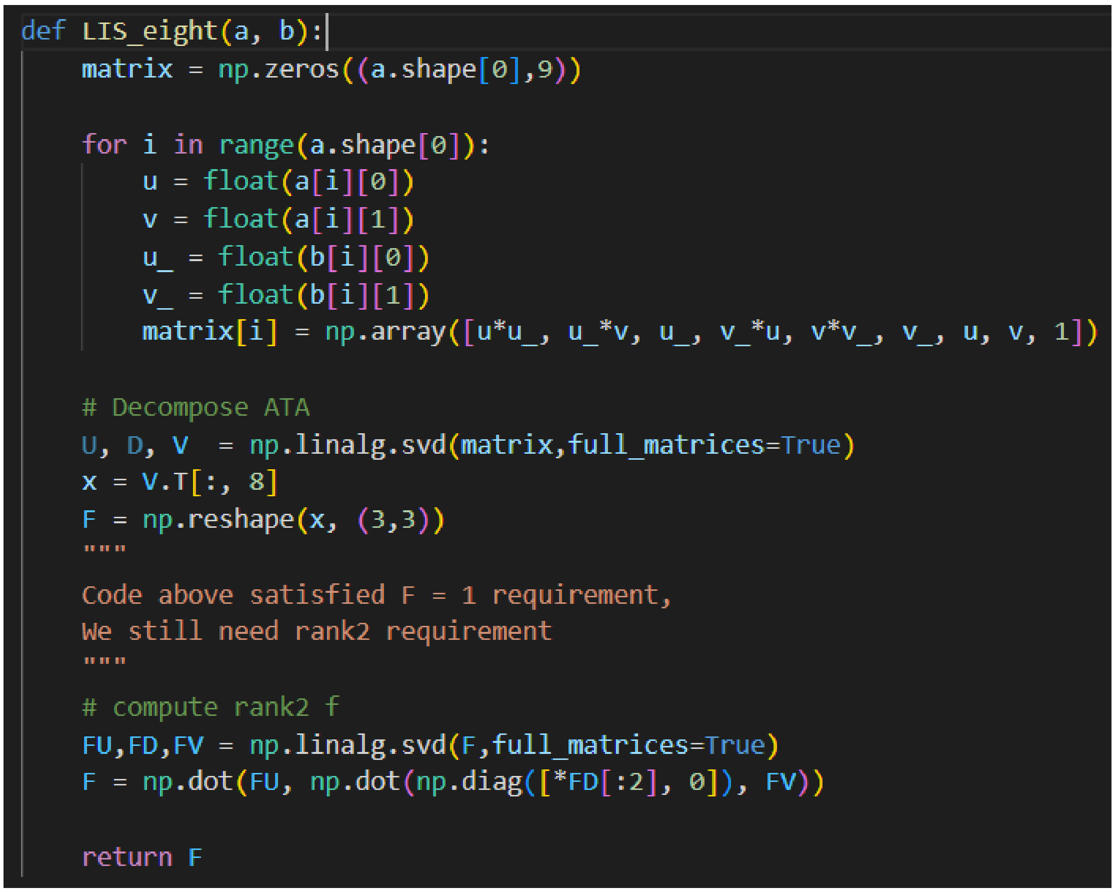
首先求解A^TA分解求得參數解，滿足$|\mathcal{F}|^2=1$。

然而，上述求解並不能滿足 rank2 的條件，因此在通過一次 SVD，使得F = U$\sum$ V^T，將 $\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix}$ 改成 $\Sigma' = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$，最後F = U$\sum'$ V^T即為所求。

-- Pseudo code --

```
>. M = $u_1u_1'$  $u_1v_1'$  $u_1$  $v_1u_1'$  $v_1v_1'$  $v_1$  $u_1'$  $v_1'$ ,1
>. U,D,V = SVD(M)
>. F = reshape(V(:9), 3,3)
>. FU, FD, FV = SVD(F)
>. FD(3,3) = 0
>. F = FU * FD * FV
```

-- Code Implement --



```python
def LIS_eight(a, b):
    matrix = np.zeros((a.shape[0],9))

    for i in range(a.shape[0]):
        u = float(a[i][0])
        v = float(a[i][1])
        u_ = float(b[i][0])
        v_ = float(b[i][1])
        matrix[i] = np.array([u*u_, u_*v, u_, v_*u, v*v_, v_, u, v, 1])

    # Decompose ATA
    U, D, V  = np.linalg.svd(matrix,full_matrices=True)
    x = V.T[:, 8]
    F = np.reshape(x, (3,3))
    """
    Code above satisfied F = 1 requirement,
    We still need rank2 requirement
    """
    # compute rank2 f
    FU,FD,FV = np.linalg.svd(F,full_matrices=True)
    F = np.dot(FU, np.dot(np.diag([*FD[:2], 0]), FV))

    return F
```

(b) Implement the normalized eight-point algorithm and report the returned fundamental matrix. Remember to enforce the rank-two constraint for the fundamental matrix via singular value decomposition.

Theory

I. Center the image data at the origin, and scale it so the mean squared distance between the origin and the data points is 2 pixels

$$q_i = T p_i \quad , \quad q_i' = T' p_i'$$

II. Use the eight-point algorithm to compute F from the points q_i and q_i' .

III. Enforce the rank-2 constraint

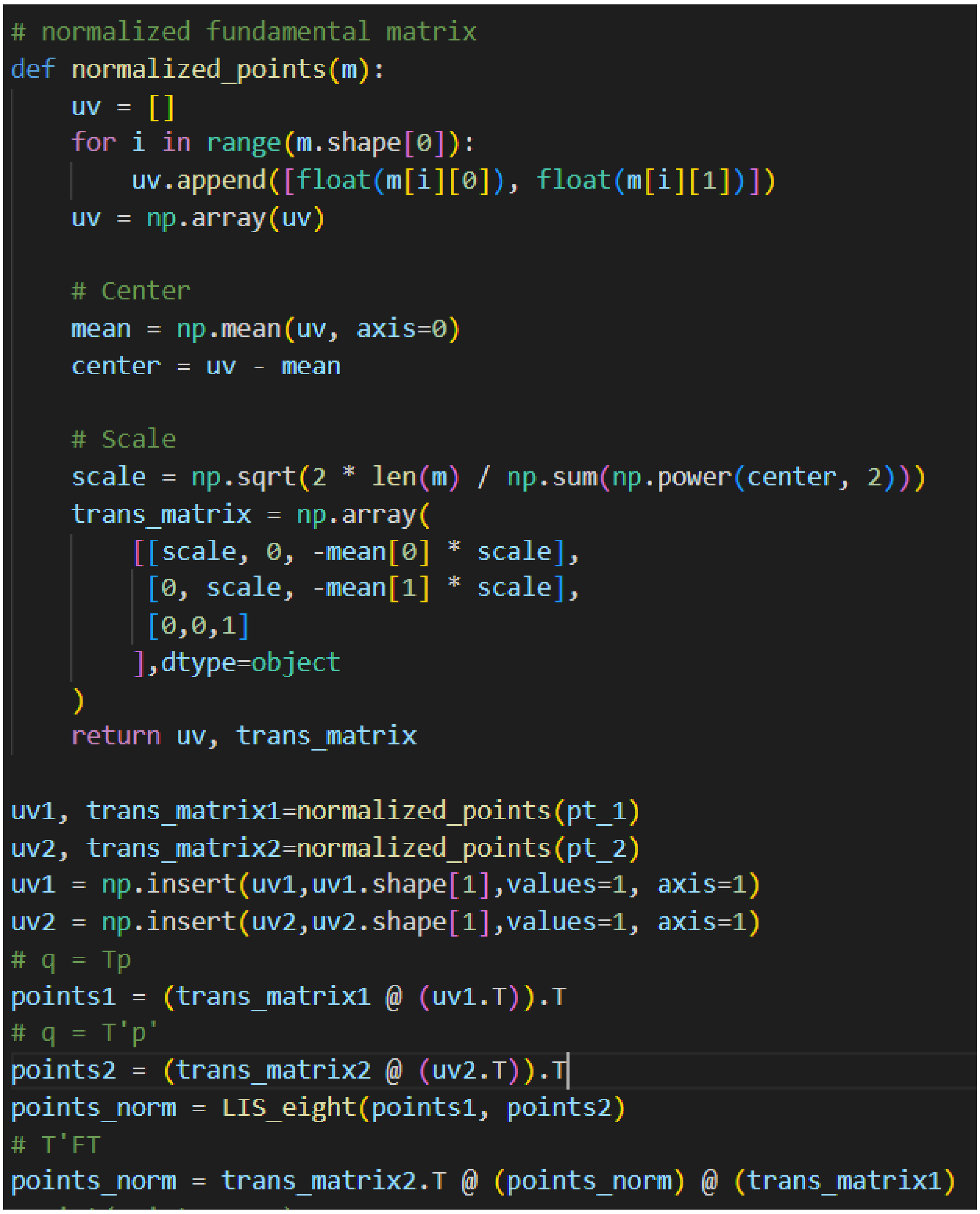
IV. Output $T^T F T'$

Implement

-- Pseudo --

```
>. center = uv – mean(uv)
>. # To satisfy the criteria that the average distance of a point p from the origin is equal to √ 2
>. Distance = $\sqrt[2]{\frac{2*points.numbers}{\sum center^2}}$
>. Transform matrix = $\begin{pmatrix} s & 0 & -st_x \\ 0 & s & -st_y \\ 0 & 0 & 1 \end{pmatrix}$
>. LTS_eight()
>. $T^T F T'$
```

-- Code Implement --



```python
# normalized fundamental matrix
def normalized_points(m):
    uv = []
    for i in range(m.shape[0]):
        uv.append([float(m[i][0]), float(m[i][1])])
    uv = np.array(uv)

    # Center
    mean = np.mean(uv, axis=0)
    center = uv - mean

    # Scale
    scale = np.sqrt(2 * len(m) / np.sum(np.power(center, 2)))
    trans_matrix = np.array(
        [[scale, 0, -mean[0] * scale],
         [0, scale, -mean[1] * scale],
         [0,0,1]
        ],dtype=object
    )
    return uv, trans_matrix

uv1, trans_matrix1=normalized_points(pt_1)
uv2, trans_matrix2=normalized_points(pt_2)
uv1 = np.insert(uv1,uv1.shape[1],values=1, axis=1)
uv2 = np.insert(uv2,uv2.shape[1],values=1, axis=1)
# q = Tp
points1 = (trans_matrix1 @ (uv1.T)).T
# q = T'p'
points2 = (trans_matrix2 @ (uv2.T)).T
points_norm = LIS_eight(points1, points2)
# T'FT
points_norm = trans_matrix2.T @ (points_norm) @ (trans_matrix1)
```

3. Plot the epipolar lines for the given point correspondences determined by the fundamental matrices computed from (a) and (b). Determine the accuracy of the fundamental matrices by computing the average distance between the feature points and their corresponding epipolar lines.

Theory

畫出 epipolar lines，

將 fundamental matrix * points 會得到 epipolar lines 係數。

對於未 normalized 的 fundamental matrix 所畫出的線會超出圖片本身。原則上先固定 x 為 [0, 圖片寬度]去求 y 值，再畫線。若 x = 0 時，y 會超出圖片，則改變以 y = 0 或圖片高度去求 x 值畫線。
計算距離方面，
將各點帶入 Ax + By + C 的平面中，再去與 $\sqrt[2]{A^2 + B^2}$ 相除得到與 epipolar lines 的距離。最後將距離除以總 points 得到平均距離。

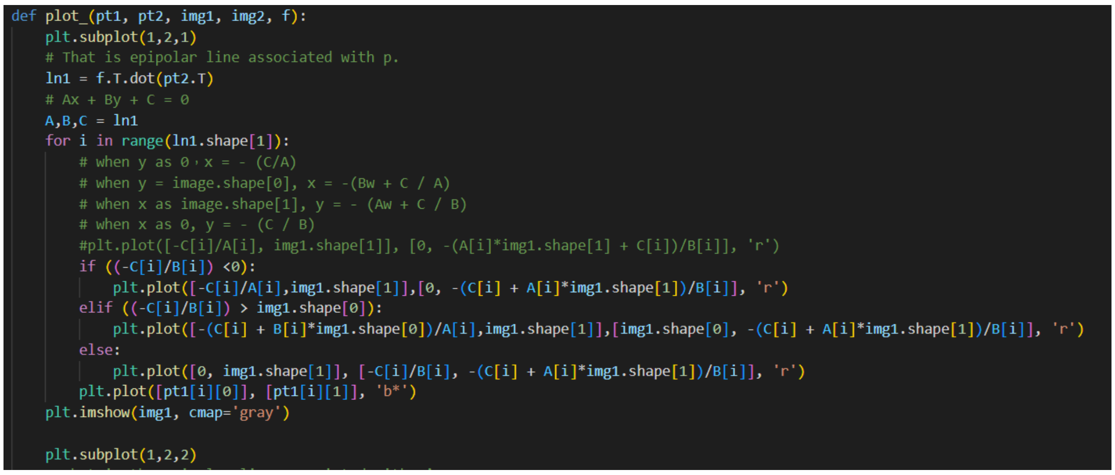
Implement

-- Pseudo --

```
>. # plot
>. ln = f.T.dot(pt2.T)
>. A,B,C = ln
>. # when y as 0，x = - (C/A)
   # when y = height, x = -(Bw + C / A)
   # when x as width, y = - (Aw + C / B)
   # when x as 0, y = - (C / B)

>. If (-C / B <0):
>.     Plot((-C/A, width), (0, -(C + A*width) / B))
>. elif(-C / B > height):
>.     Plot((-C/A, width), (height, -(C + A*width) / B))
>. else:
>.     Plot((0,width), (-C/B, -(C + A*width) / B)


>. # plot normalized
>. ln = f.T.dot(pt2.T)
>. A,B,C = ln
>. # when x as width, y = - (Aw + C / B)
   # when x as 0, y = - (C / B)
>. Plot((0,width), (-C/B, -(C + A*width) / B)


>. # Calculate Distance
>. ln = f.T.dot(pt2.T)
>. dist = abc(Ax + By + C) / sqrt(A**2 + B**2)
>. dist / points.number
```

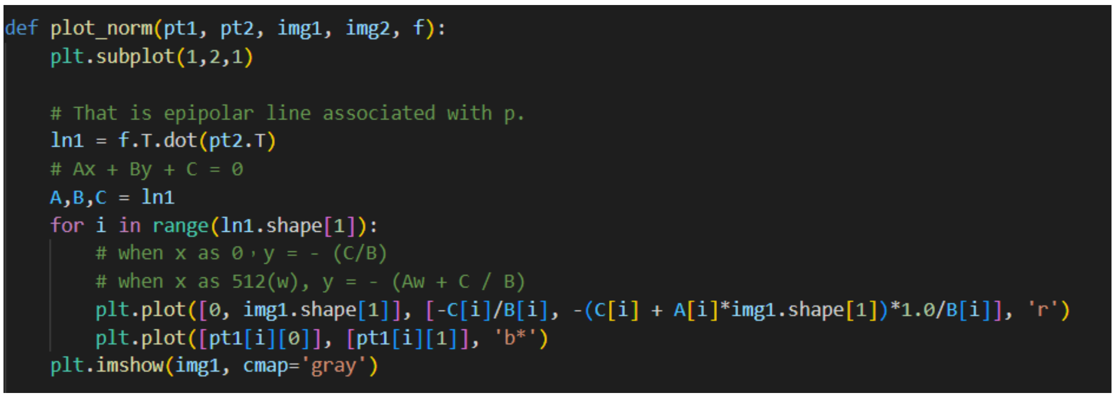
-- Code Implement --

Plot



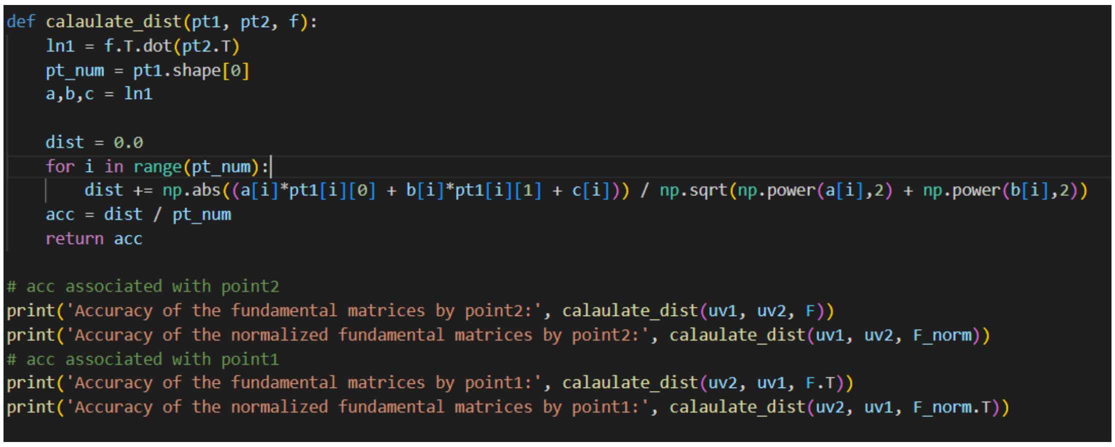
```
def plot_(pt1, pt2, img1, img2, f):
    plt.subplot(1,2,1)
    # That is epipolar line associated with p.
    ln1 = f.T.dot(pt2.T)
    # Ax + By + C = 0
    A,B,C = ln1
    for i in range(ln1.shape[1]):
        # when y as 0，x = - (C/A)
        # when y = image.shape[0], x = -(Bw + C / A)
        # when x as image.shape[1], y = - (Aw + C / B)
        # when x as 0, y = - (C / B)
        #plt.plot([-C[i]/A[i], img1.shape[1]], [0, -(A[i]*img1.shape[1] + C[i])/B[i]], 'r')
        if ((-C[i]/B[i]) <0):
            plt.plot([-C[i]/A[i],img1.shape[1]],[0, -(C[i] + A[i]*img1.shape[1])/B[i]], 'r')
        elif ((-C[i]/B[i]) > img1.shape[0]):
            plt.plot([-(C[i] + B[i]*img1.shape[0])/A[i],img1.shape[1]],[img1.shape[0], -(C[i] + A[i]*img1.shape[1])/B[i]], 'r')
        else:
            plt.plot([0, img1.shape[1]], [-C[i]/B[i], -(C[i] + A[i]*img1.shape[1])/B[i]], 'r')
        plt.plot([pt1[i][0]], [pt1[i][1]], 'b*')
    plt.imshow(img1, cmap='gray')

    plt.subplot(1,2,2)
```

Plot Normalized



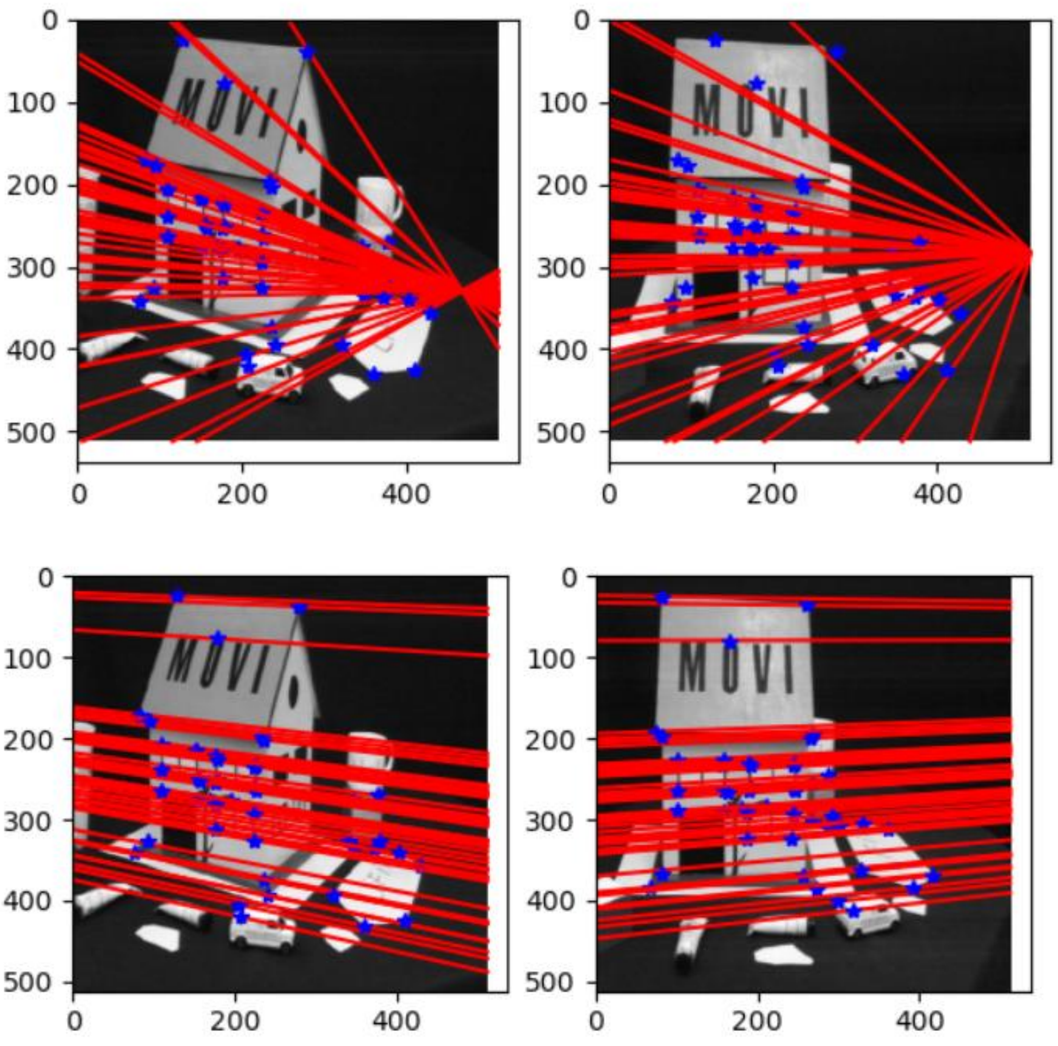
```
def plot_norm(pt1, pt2, img1, img2, f):
    plt.subplot(1,2,1)

    # That is epipolar line associated with p.
    ln1 = f.T.dot(pt2.T)
    # Ax + By + C = 0
    A,B,C = ln1
    for i in range(ln1.shape[1]):
        # when x as 0，y = - (C/B)
        # when x as 512(w), y = - (Aw + C / B)
        plt.plot([0, img1.shape[1]], [-C[i]/B[i], -(C[i] + A[i]*img1.shape[1])*1.0/B[i]], 'r')
        plt.plot([pt1[i][0]], [pt1[i][1]], 'b*')
    plt.imshow(img1, cmap='gray')
```

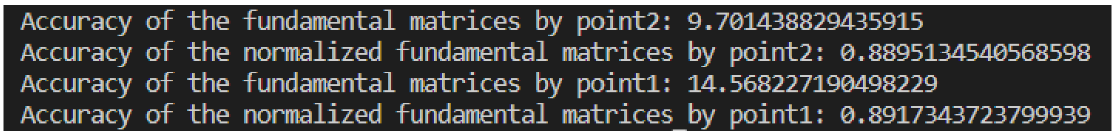
Calculate Distance



```
def calaulate_dist(pt1, pt2, f):
    ln1 = f.T.dot(pt2.T)
    pt_num = pt1.shape[0]
    a,b,c = ln1

    dist = 0.0
    for i in range(pt_num):
        dist += np.abs((a[i]*pt1[i][0] + b[i]*pt1[i][1] + c[i])) / np.sqrt(np.power(a[i],2) + np.power(b[i],2))
    acc = dist / pt_num
    return acc

# acc associated with point2
print('Accuracy of the fundamental matrices by point2:', calaulate_dist(uv1, uv2, F))
print('Accuracy of the normalized fundamental matrices by point2:', calaulate_dist(uv1, uv2, F_norm))
# acc associated with point1
print('Accuracy of the fundamental matrices by point1:', calaulate_dist(uv2, uv1, F.T))
print('Accuracy of the normalized fundamental matrices by point1:', calaulate_dist(uv2, uv1, F_norm.T))
```

-- Result --





```
Accuracy of the fundamental matrices by point2: 9.701438829435915
Accuracy of the normalized fundamental matrices by point2: 0.8895134540568598
Accuracy of the fundamental matrices by point1: 14.568227190498229
Accuracy of the normalized fundamental matrices by point1: 0.8917343723799939
```

2. You need to determine a homograpgy transformation for plan-to-plane transformation. The homography transformation is determined by a set of point correspondences between the source image and the target image.

(a) Implement a function that estimates the homography matrix H that maps a set of interest points to a new set of interest points.

Theory

兩個座標座標的轉換矩陣為

$[U \quad V \quad W] = [x \quad y \quad 1] \begin{bmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{bmatrix}$，但是將其化為方程組的話，會陷沒有常數項難以求解。因此我們將 t_{33}d 改寫為 1。
方程組將變為：

$$\begin{cases} U = t_{11} * x + t_{21} * y + t_{31} \\ V = t_{12} * x + t_{22} * y + t_{32} \\ W = t_{13} * x + t_{23} * y + 1 \end{cases}$$

整理可得：

$$\begin{cases} x * t_{11} + y * t_{21} + 1 * t_{31} + 0 * t_{12} + 0 * t_{22} + 0 * t_{32} - ux * t_{13} - uy * t_{23} = u \\ \\ 0 * t_{11} + 0 * t_{21} + 0 * t_{31} + x * t_{12} + y * t_{22} + 1 * t_{32} - vx * t_{13} - vy * t_{23} = v \end{cases}$$

因此，寫成線性方程組將為如下：

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -u_1x_1 & -u_1y_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -v_1x_1 & -v_1y_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -u_2x_2 & -u_2y_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -v_2x_2 & -v_2y_2 \\ \dots & & & & & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -u_nx_n & -u_ny_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -v_nx_n & -v_ny_n \end{bmatrix} \begin{bmatrix} t_{11} \\ t_{21} \\ t_{31} \\ t_{12} \\ t_{22} \\ t_{32} \\ t_{13} \\ t_{23} \end{bmatrix} = \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ \dots \\ u_n \\ v_n \end{bmatrix}$$

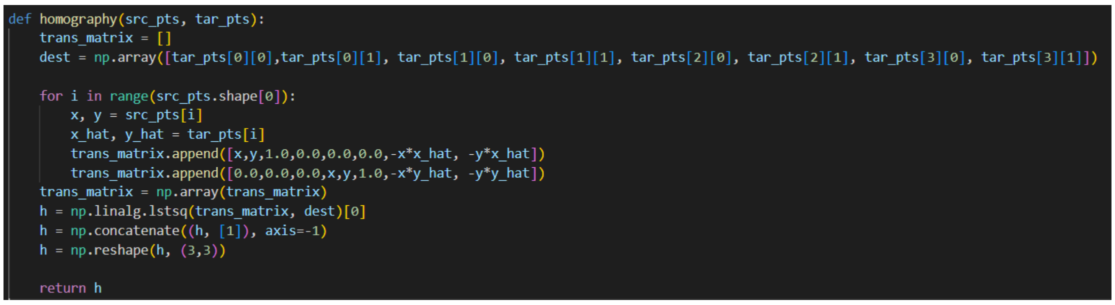
因下一題為將台達館轉至正面，因此此時我直接將台達館正面區域作為 interest points，轉至 512 X 512 的圖片，也就是 new set interest points。
台達館正面座標依順時針依序為： [425,330],[400,800],[900,1000],[880,0]
新座標為：[0,0],[0,512],[512,512],[512,0]

Implement

-- Pseudo --

```
>. M = $\begin{matrix} x_n & y_n & 1 & 0 & 0 & 0 & -u_nx_n & -u_ny_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -v_nx_n & -v_ny_n \end{matrix}$
>. H = Least Square(M, destination_points)
>. H = reshape(H, 3,3)
```

-- Code Implement --



```python
def homography(src_pts, tar_pts):
    trans_matrix = []
    dest = np.array([tar_pts[0][0],tar_pts[0][1], tar_pts[1][0], tar_pts[1][1], tar_pts[2][0], tar_pts[2][1], tar_pts[3][0], tar_pts[3][1]])

    for i in range(src_pts.shape[0]):
        x, y = src_pts[i]
        x_hat, y_hat = tar_pts[i]
        trans_matrix.append([x,y,1.0,0.0,0.0,0.0,-x*x_hat, -y*x_hat])
        trans_matrix.append([0.0,0.0,0.0,x,y,1.0,-x*y_hat, -y*y_hat])
    trans_matrix = np.array(trans_matrix)
    h = np.linalg.lstsq(trans_matrix, dest)[0]
    h = np.concatenate((h, [1]), axis=-1)
    h = np.reshape(h, (3,3))

    return h
```

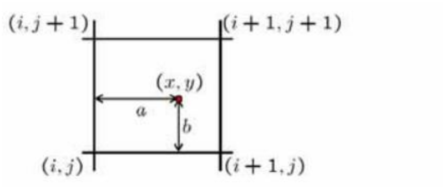
(b) Specify a set of point correspondences for the source image of the Delta building and the target one. Compute the 3X3 homography matrix to rectify the front building of the Delta building image. The rectification is to make the new image plane parallel to the front building as best as possible. Please select four corresponding straight lines to compute the homograph matrix.

Theory

Homography matrix 於上一題已提及，之後進行 backward warping and bilinear interpolation。

Backward warping 與 forward mapping 不同，反而是從 destination 座標去反求 source 座標位置，再將其複製 pixel 過來 destination 上。
因此，需先 destination points 乘上 Homography matrix 反矩陣求得 source points。
然而透過反矩陣求得的座標為浮點數，此時利用 bilinear interpolation 去得到鄰近 4 的點的 pixel 值再依比例綜合。

具體如下：
先求得上下兩邊 x 軸的比例，再依 y 軸比例分配各 pixel 值。



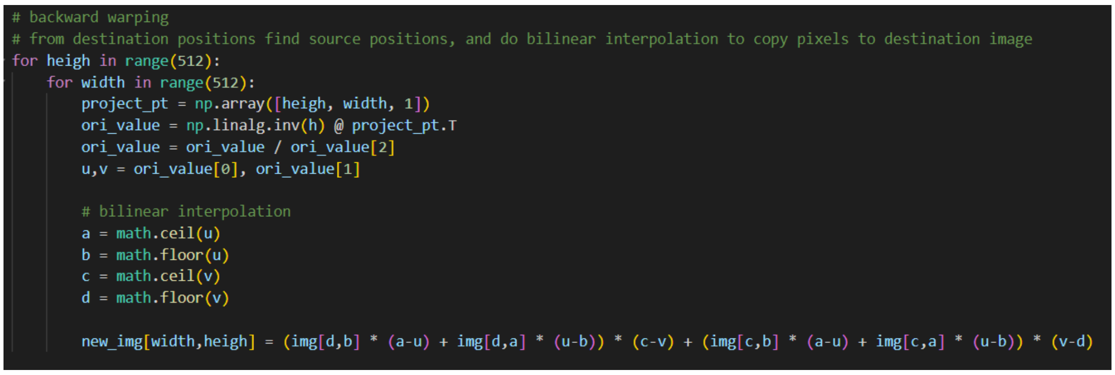
$$\begin{aligned} f(x,y) = \; & (1-a)(1-b) \quad f[i,j] \\ & +a(1-b) \quad f[i+1,j] \\ & +ab \quad f[i+1,j+1] \\ & +(1-a)b \quad f[i,j+1] \end{aligned}$$

Implement

-- Pseudo –

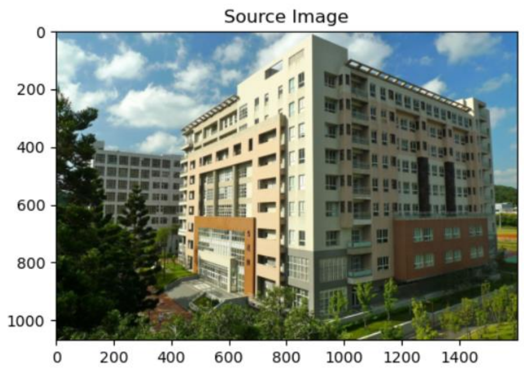
```
>. For (x,y):    u,v = $h^{-1}(x, y)$
>. # bilinear interpolation
>. a = ceil(u)
>. b = floor(u)
>. c = ceil(v)
>. d = floor(v)
>. dst[x,y] = [src[a,d] * (u-b) + sec[b,d] * (a-u)] * (c-v)    +    /
>. [src[a,c] * (u-b) + sec[b,c] * (a-u)] * (v-d)
```

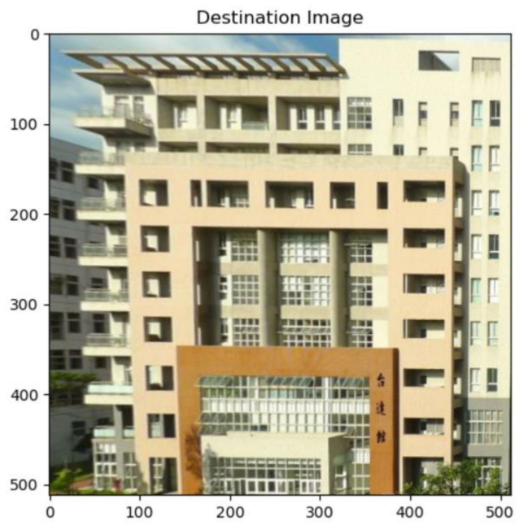
-- Code Implement –

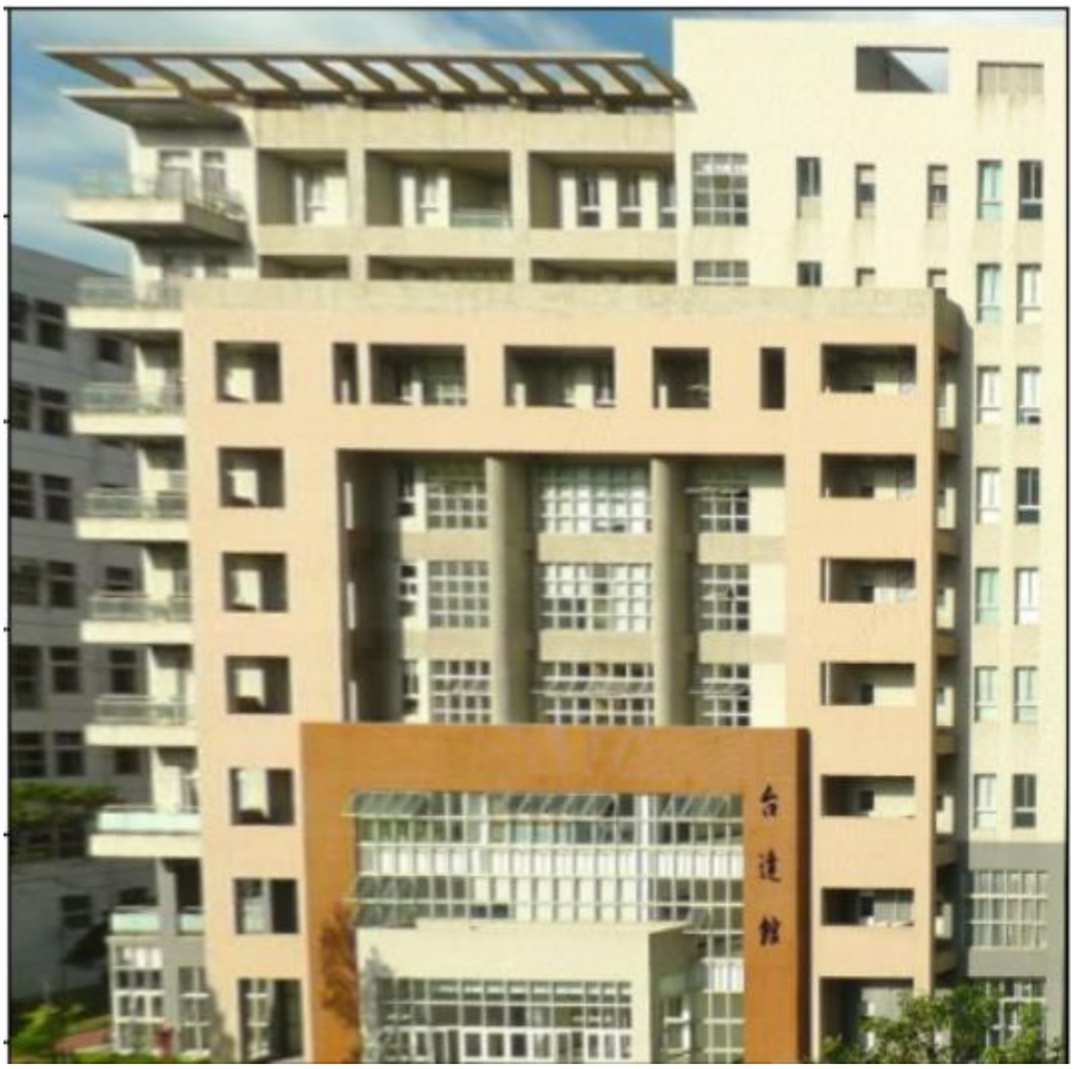


```python
# backward warping
# from destination positions find source positions, and do bilinear interpolation to copy pixels to destination image
for heigh in range(512):
    for width in range(512):
        project_pt = np.array([heigh, width, 1])
        ori_value = np.linalg.inv(h) @ project_pt.T
        ori_value = ori_value / ori_value[2]
        u,v = ori_value[0], ori_value[1]

        # bilinear interpolation
        a = math.ceil(u)
        b = math.floor(u)
        c = math.ceil(v)
        d = math.floor(v)

        new_img[width,heigh] = (img[d,b] * (a-u) + img[d,a] * (u-b)) * (c-v) + (img[c,b] * (a-u) + img[c,a] * (u-b)) * (v-d)
```

-- Result --



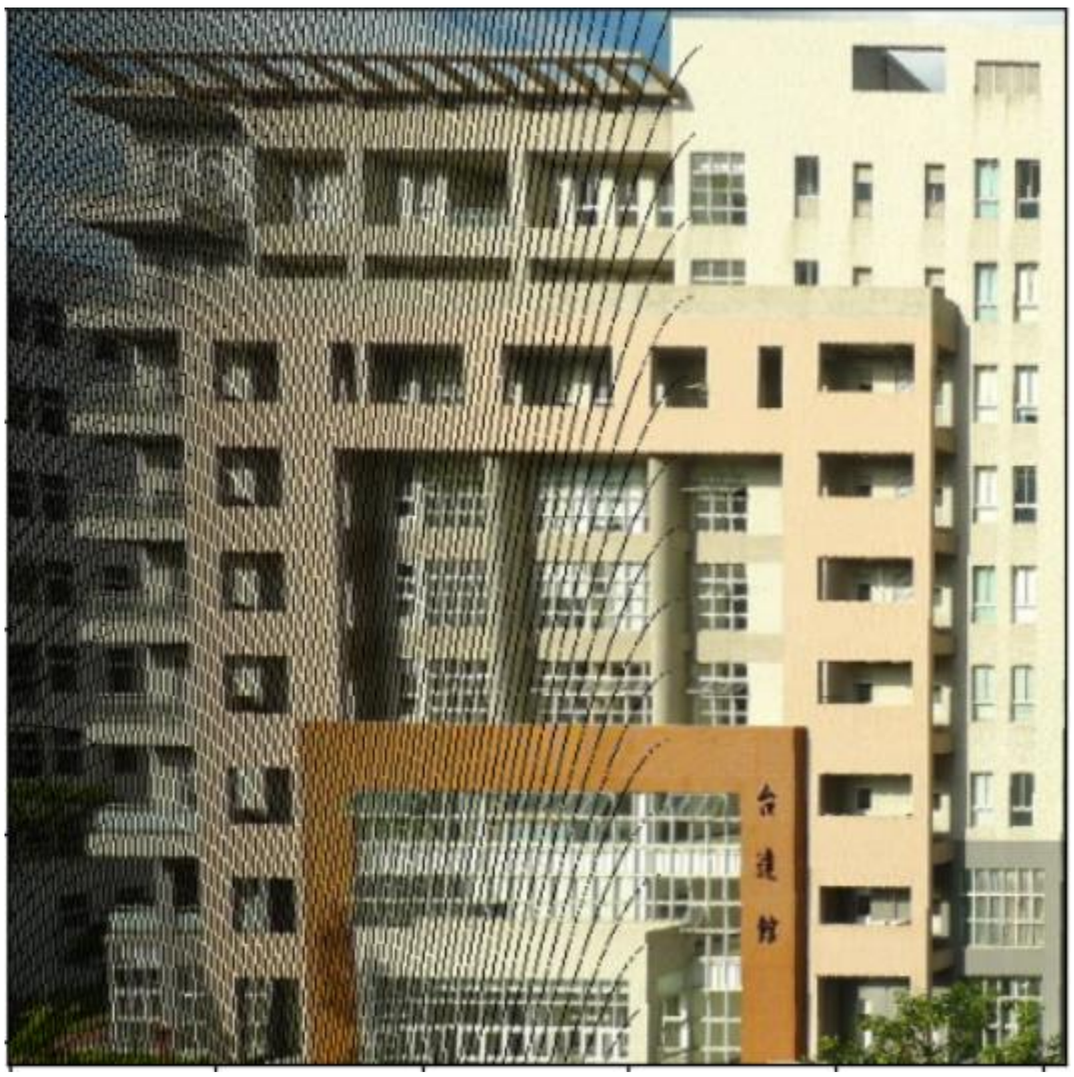




原圖越傾斜的部分轉換過來會越模糊。

Another Experiment

實驗使用 forward mapping 與 取 floor 的投射:


台達館