# CVHW4

team members:0510134 王泓仁、0516006 賴奕善、0516218 軒轅照雯

## 1. Introduction

SfM is the process of reconstructing 3D structure from its projections into a series of images taken from different viewpoints. It is studied in the fields of computer vision and visual perception. In biological vision, SfM refers to the phenomenon by which humans (and other living creatures) can recover 3D structure from the projected 2D (retinal) motion field of a moving object or scene.

It commonly starts with feature extraction and matching, followed by geometric verification. The resulting scene graph serves as the foundation for the reconstruction stage, which seeds the model with a carefully selected two-view reconstruction, before incrementally registering new images, triangulating scene points, filtering outliers, and refining the reconstruction using bundle adjustment (BA).

## 2. Implementation procedure

### (1) find out correspondence across images using the same procedure as homework 3

- Interest points detection and feature description by SIFT.
- For every keypoint in image1, find the 2 closest keypoints in image2.
- Set ratio distance threshold to get the best matches.

```
#SIFT
sift = cv2.xfeatures2d.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
```

```
#feature matching: for every keypoint in image1, find 2 best matches
match_pt=np.zeros((des1.shape[0], 2)) #(indices of kp1, indices of kp2_1, indices of kp2_2)
match_dist=np.zeros((des1.shape[0], 2)) #(distance of kp1 and kp2_1, distance of kp1 and kp2_2)
for i in range(des1.shape[0]):
    min1=1000
    min2=1000
    for j in range(des2.shape[0]):
        d=dist(des1[i], des2[j])
        if d <= min1:
            min2=min1
            min1=d
            pt1=j
        elif d <= min2:
            min2=d
            pt2=j
    match_pt[i][0]=pt1
    match_pt[i][1]=pt2
    match_dist[i][0]=min1
    match_dist[i][1]=min2
```

```
#ratio distance: for the 2 best matches of each keypoint, if the ratio distance>0.45, discard
goodMatch=[]
coor1=[]
coor2=[]
int_coor1=[]
int_coor2=[]
for x in range(match_dist.shape[0]):
    if match_dist[x][0] < 0.45*match_dist[x][1]:
        goodMatch.append([x,int(match_pt[x][0])])

gM_num=len(goodMatch)
for m, n in goodMatch:
    coor1.append((kp1[m].pt[0],kp1[m].pt[1]))
    coor2.append((kp2[n].pt[0],kp2[n].pt[1]))
    int_coor1.append((int(kp1[m].pt[0]),int(kp1[m].pt[1])))
    int_coor2.append((int(kp2[n].pt[0]),int(kp2[n].pt[1])))
```

(2) Estimate fundamental matrix with normalized 8-point algorithm

- T1, T2 normalize the coordinates
- randomly choose 8 points
- solve f from Af=0 by SVD
- resolve det(F)=0 constraint using SVD
- use $|x'^{T}Fx|$ as distance and set threshold to determine inliers
- choose the largest number of inliers and recalculate the fundamental matrix with them

```
#estimate the fundamental matrix with 8-point algorithm
#homo coor
homo_coor1=np.zeros((3, gM_num))
homo_coor2=np.zeros((3, gM_num))
for x in range(gM_num):
    homo_coor1[0,x] = coor1[x][0]
    homo_coor1[1,x] = coor1[x][1]
    homo_coor1[2,x] = 1
    homo_coor2[0,x] = coor2[x][0]
    homo_coor2[1,x] = coor2[x][1]
    homo_coor2[2,x] = 1
#normalization
T1=np.array([[2/img1.shape[0], 0, -1],[0, 2/img1.shape[1], -1],[0, 0, 1]])
T2=np.array([[2/img2.shape[0], 0, -1],[0, 2/img2.shape[1], -1],[0, 0, 1]])
norm_coor1=T1@homo_coor1
norm_coor2=T2@homo_coor2
```

```
#RANSAC
max_inliers=0
best_inliers=[]
for i in range(100): #iteration
    A = []
    sample=np.random.randint(gM_num, size=8) #samples
    for j in sample:
        u, v = norm_coor1[0,j], norm_coor1[1,j]
        x, y = norm_coor2[0,j], norm_coor2[1,j]
        A.append([u*x, u*y, u, v*x, v*y, v, x, y, 1])
    A = np.asarray(A)
    u, s, vh = np.linalg.svd(A)
    f = (vh[-1,:] / vh[-1,-1]).reshape(3, 3)
    #constraint det(F)=0
    u, s, vh = np.linalg.svd(f)
    s[2]=0
    F=u@np.diag(s)@vh
    #|x'TFx|<threshold to determine inliers
    n=0
    inliers=[]
    for k in range(gM_num):
        if abs((norm_coor2[:, k].T@F)@norm_coor1[:,k])<1: #threshold
            n=n+1
            inliers.append(k)
    if n>max_inliers:
        best_inliers=inliers
        max_inliers=n
```

```
#recalculate fundamental matrix
B=[]
for b in best_inliers:
    u, v = norm_coor1[0,b], norm_coor1[1,b]
    x, y = norm_coor2[0,b], norm_coor2[1,b]
    B.append([u*x, u*y, u, v*x, v*y, v, x, y, 1])
B = np.asarray(B)
u, s, vh = np.linalg.svd(B)
f = (vh[-1,:] / vh[-1,-1]).reshape(3, 3)
u, s, vh = np.linalg.svd(f)
s[2]=0
F=u@np.diag(s)@vh
#denormalize
F=T2.T@F@T1
```

(3) Draw the interest points on image1 and the corresponding epipolar lines on image2

- epipolar line on image 2: m = Fx where m[0]=a, m[1]=b, m[2]=c for ax+by+c=0

```
#features & epipolar lines visualization
radius = 3
color_arr = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (255, 255, 0), (0, 255, 255), (255,
thickness = 1
pt_img=img_1.copy()
eline_img=img_2.copy()
for x in range(gM_num):
    color=color_arr[x%6]
    pt_img=cv2.circle(pt_img, int_coor1[x], radius, color, thickness)
    m=F@homo_coor1[:,x]
    y1=int((-m[0]*0-m[2])/m[1])
    y2=int((-m[0]*(img2.shape[1]-1)-m[2])/m[1])
    eline_img=cv2.line(eline_img, (0, y1),(img2.shape[1]-1, y2), color, thickness)
vis_img = np.concatenate((pt_img, eline_img), axis=1)
cv2.imwrite('feature_epipolar_visualization.jpg', vis_img)
```

(4) Get 4 possible solutions of the camera matrix

```
#Essential matrix
K = np.array([[1.4219,0.0005,0.5092],[0,1.4219,0.3802],[0,0,0.01]])
E = K.T @ F @ K
'''
K1 = np.array([[5426.566895,0.678017,330.096680],
               [0.000000,5423.133301,648.950012],
               [0.000000,0.000000,1.000000]])
K2 = np.array([[5426.566895,0.678017,387.430023],
               [0.000000,5423.133301,620.616699],
               [0.000000,0.000000,1.000000]])
E = K2.T @ F @ K1
'''
```

```
#Camera matrix
u, s, vh = np.linalg.svd(E)
m=1
s=np.array([m, m, 0])
E=u @ np.diag(s) @ vh
u, s, vh = np.linalg.svd(E)
W=np.array([[0, -1, 0], [1, 0, 0], [0, 0, 1]])

R1=np.identity(3)
c1=np.zeros(3).reshape(3,1)
R21=u @ W @ vh
R22=u @ W.T @ vh
c21=np.array(u[:, 2])
c21 = c21.reshape(3,1)
c22=np.array(-u[:, 2])
c22 = c22.reshape(3,1)

P_o = np.concatenate((K @ R1, K @ R1 @ -c1),axis=1)
P1 = np.concatenate((K @ R21, K @ R21 @ -c21),axis=1)
P2 = np.concatenate((K @ R21, K @ R21 @ -c22),axis=1)
P3 = np.concatenate((K @ R22, K @ R22 @ -c21),axis=1)
P4 = np.concatenate((K @ R22, K @ R22 @ -c22),axis=1)
'''
P_o = np.concatenate((K1 @ R1, K1 @ R1 @ -c1),axis=1)
P1 = np.concatenate((K2 @ R21, K2 @ R21 @ -c21),axis=1)
P2 = np.concatenate((K2 @ R21, K2 @ R21 @ -c22),axis=1)
P3 = np.concatenate((K2 @ R22, K2 @ R22 @ -c21),axis=1)
P4 = np.concatenate((K2 @ R22, K2 @ R22 @ -c22),axis=1)
'''
```

(5) Apply triangulation to get the 3D points

- choose the solution with most 3D points in front of both cameras

```
#Triangulation
def triangulation(c1,c2,P1,R1,T1,P2,R2,T2):
    X_3d = []
    X_2d = []
    n= 0
    for i in range(gM_num):
        A = np.array([c1[i][0] * P1[2,:] - P1[0,:],
                      c1[i][1] * P1[2,:] - P1[1,:],
                      c2[i][0] * P2[2,:] - P2[0,:],
                      c2[i][1] * P2[2,:] - P2[1,:]
                     ])
        Au, As, Avh = np.linalg.svd(A)
        #3d point
        p=Avh[-1,:3]/Avh[-1,-1]
        X_3d.append(p.tolist())
        X_2d.append(c1[i])
        #camera center
        C1=-R1.T@T1
        C2=-R2.T@T2
        #check if the 3d point is in front of both cameras
        if ((np.dot((p - C1.T),R1[2,:].T)) > 0) and ((np.dot((p - C2.T),R2[2,:].T)) > 0):
            n += 1
    return n,X_3d,X_2d
```

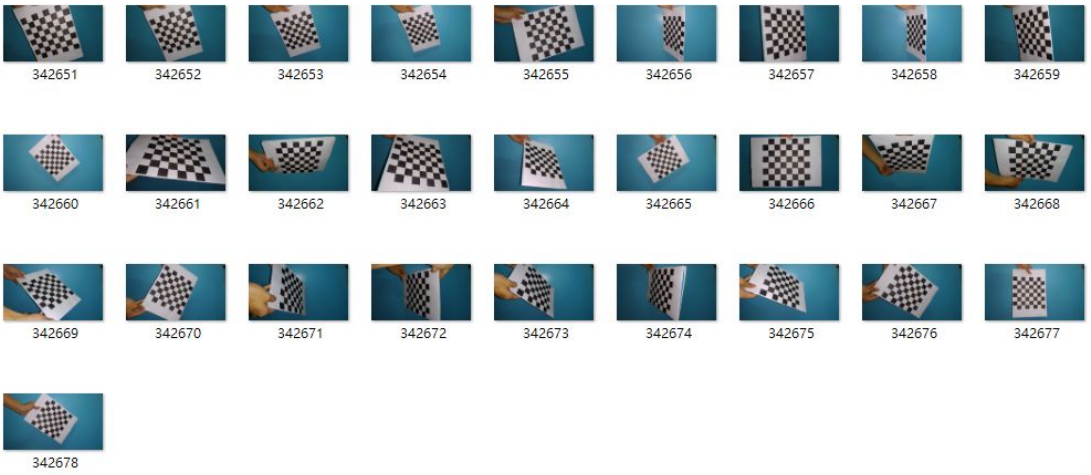
```
#choose the solution with most 3d points in front of both cameras
max_n = (-1,[])
n1 = triangulation(coor1,coor2,P_o,R1,c1, P1,R21,c21)
if n1[0] > max_n[0]:
    max_n = n1
n2 = triangulation(coor1,coor2,P_o,R1,c1, P2,R21,c22)
if n2[0] > max_n[0]:
    max_n = n2
n3 = triangulation(coor1,coor2,P_o,R1,c1, P3,R22,c21)
if n3[0] > max_n[0]:
    max_n = n3
n4 = triangulation(coor1,coor2,P_o,R1,c1, P4,R22,c22)
if n4[0] > max_n[0]:
    max_n = n4
```

(6) 3D reconstruction and texture mapping by matlab

```
#3d reconstruction & texture mapping
eng = matlab.engine.start_matlab()
m = P_o.tolist()
m = matlab.double(m)
threedp =  matlab.double(max_n[1])
twodp =  matlab.double(max_n[2])
eng.obj_main(threedp, twodp, m, 'Mesona1.JPG', 1 ,nargout=0)
#eng.obj_main(threedp, twodp, m, 'Statue1.bmp', 1 ,nargout=0)
```

(7) Take pictures of chessboard for camera calibration and pictures of speaker as our own experiment photos

- For calibration: fix camera position and focus, rotate and move the chessboard
- For own photos: fix object position, slightly rotate the camera, erase the background of the photo




(8) Use cv2 function to do camera calibration and find the intrinsic matrix K

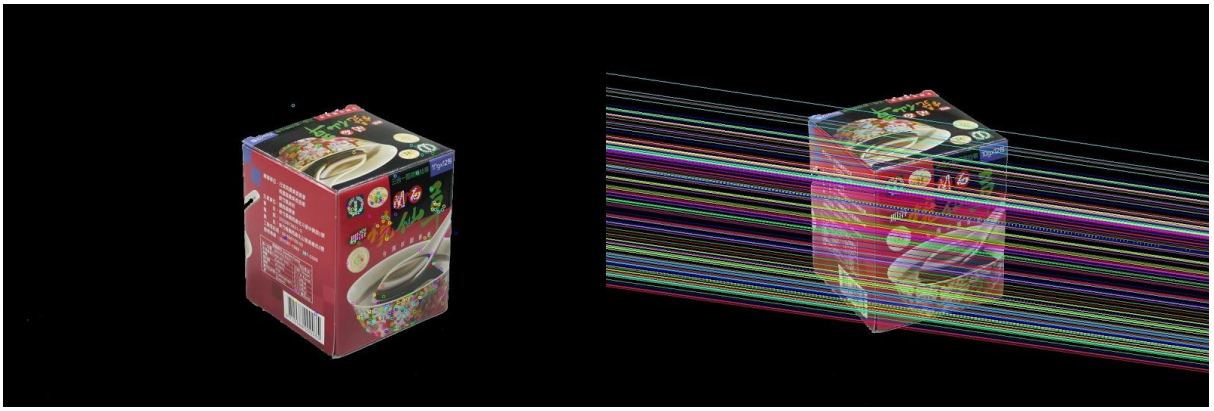
```
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, img_size,None,None)
own_camera_K=mtx
own_camera_K
```

```
array([[1.30090943e+03, 0.00000000e+00, 8.48833947e+02],
       [0.00000000e+00, 1.30903009e+03, 3.92923837e+02],
       [0.00000000e+00, 0.00000000e+00, 1.00000000e+00]])
```

(9) repeat step (1)~(7)

## 3. Experimental result

(1) data1



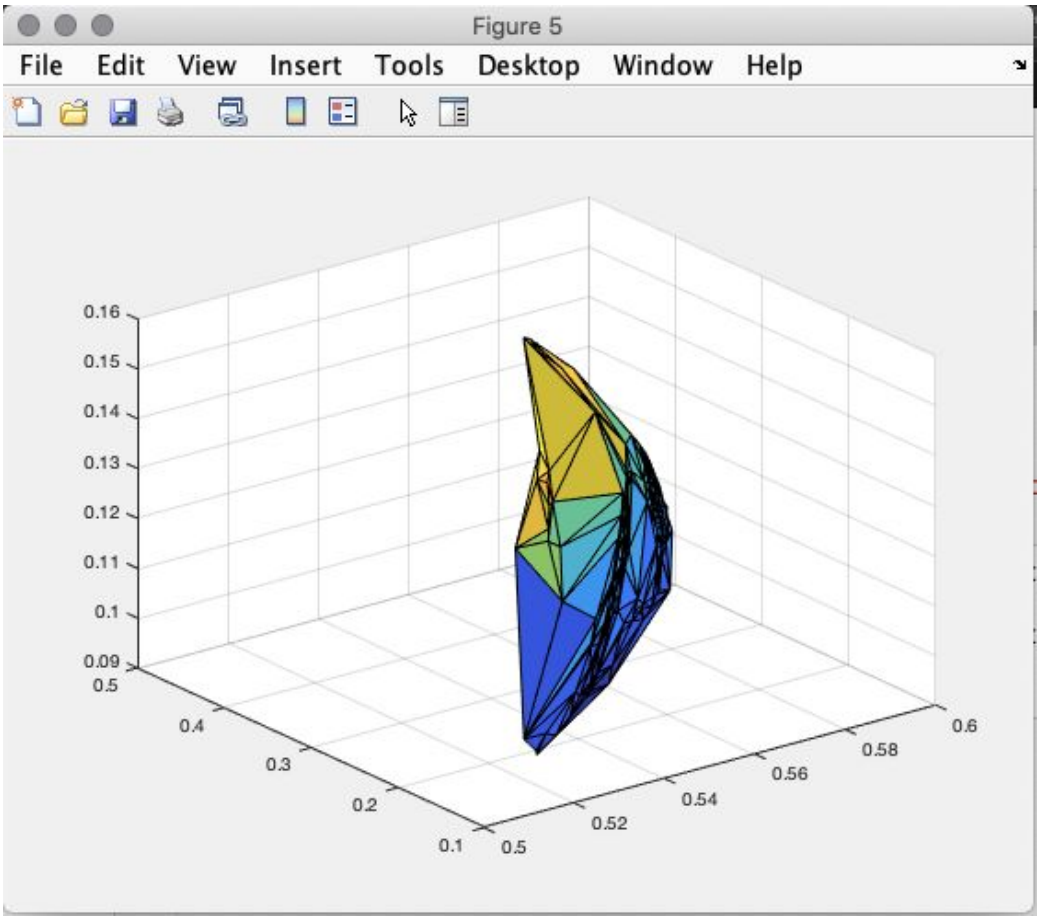


Figure 5
File Edit View Insert Tools Desktop Window Help
0.16
0.15
0.14
0.13
0.12
0.11
0.1
0.09
0.5
0.4
0.3
0.2
0.1
0.5
0.52
0.54
0.56
0.58
0.6

(2) data2

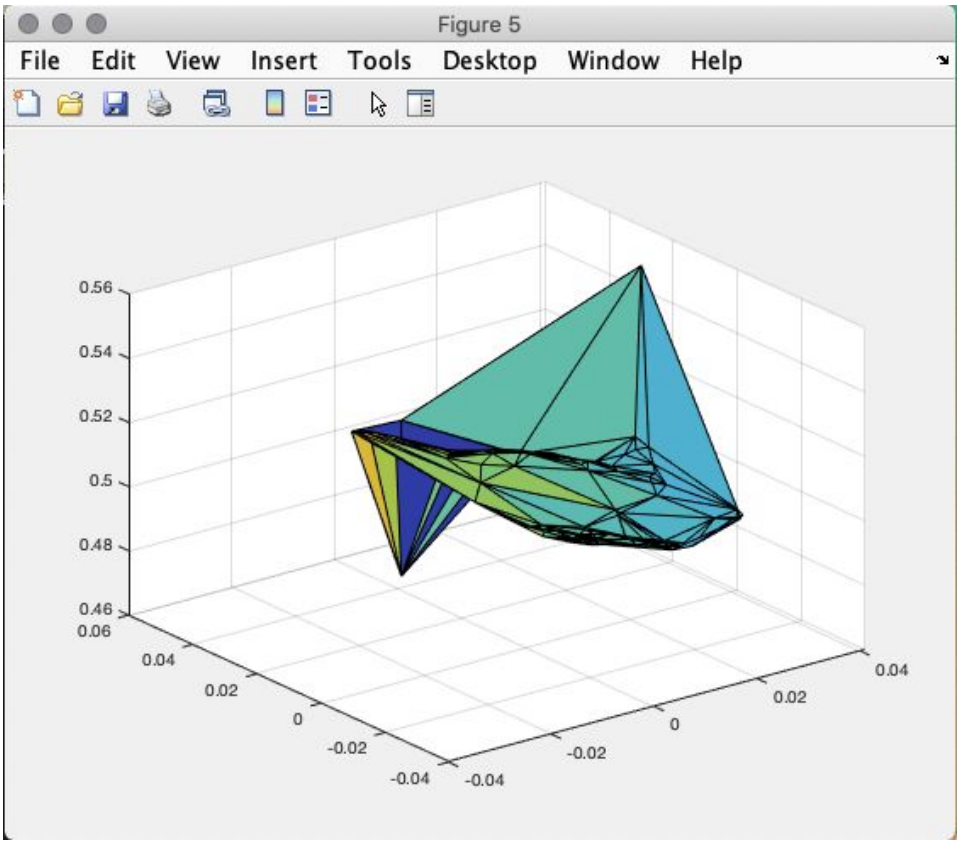


Figure 5
File Edit View Insert Tools Desktop Window Help
0.56
0.54
0.52
0.5
0.48
0.46
0.06
0.04
0.02
0
-0.02
-0.04
-0.04
-0.02
0
0.02
0.04

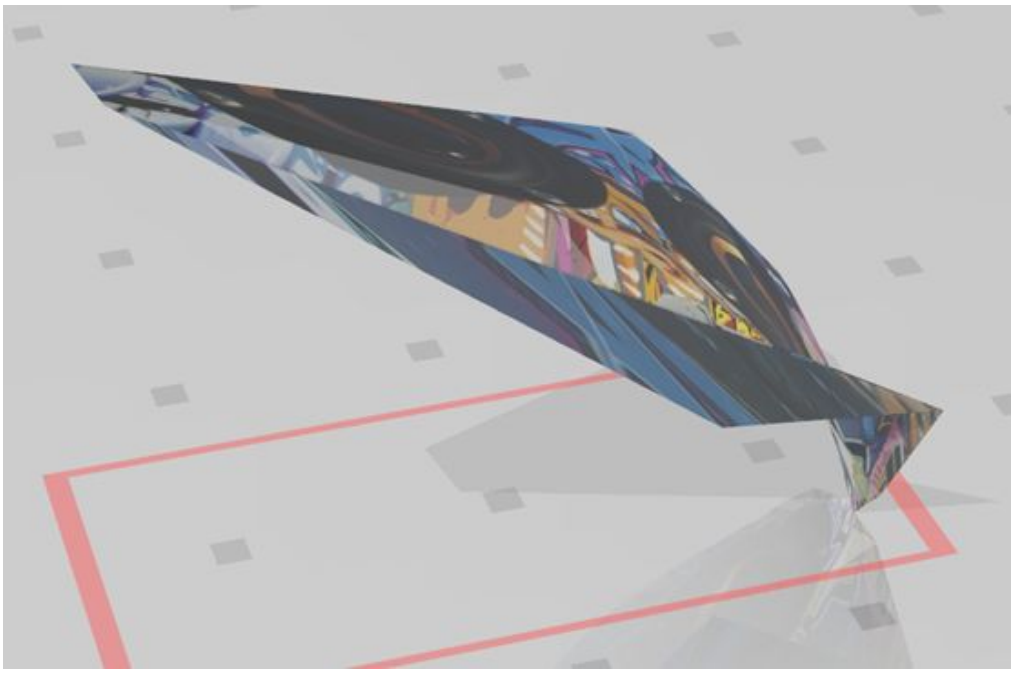
(3) own photos:

Size:1839x1384

Ransac iteration:100

Ratio distance:0.45

-many key points, radiate epipolar lines

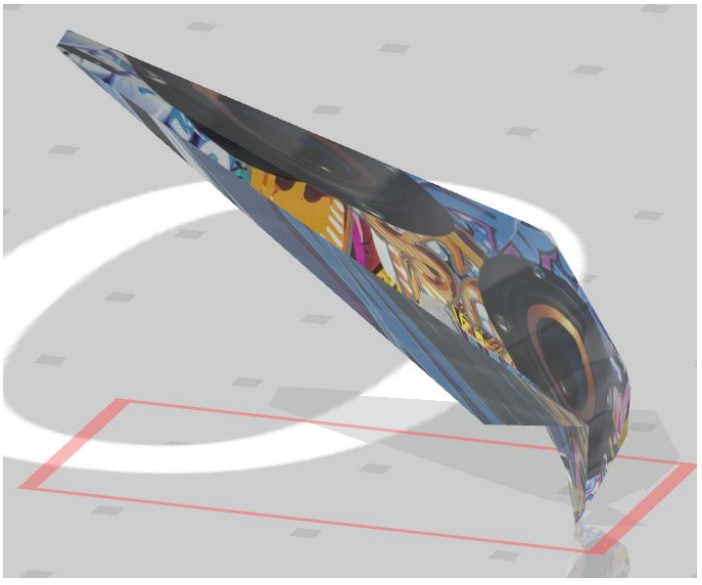


Size:400x300

Ransac iteration:200

Ratio distance:0.45

few key point, few epipolar line



## 4. Discussion

This homework was quite challenging that we met several problems:

### (1) Normalization

There are mainly 2 ways of normalizations that are widely used in the normalized 8-point algorithm we found, and this took us a long time to decide which to apply. At last, we use the one that was mentioned in class, since the results seem to be better.

### (2) The threshold for determining inliers

This threshold plays an important role in estimating the fundamental matrix and also directly affects the epipolar lines. We tried different values between 0 and 1, since the optimal value of $|x'^{T}Fx|$ equals to 0; however, “1” gave us the best and stable results.

### (3) Epipolar line representation

At first, we misunderstood the meaning of the elements in array m=Fx, we thought m[0]/m[2] was the slope and m[1]/m[2] was the y-intersection in the equation y=ax+b, so the epipolar lines appeared strangely. Finally, we found out that m[0]=a, m[1]=b, m[2]=c in the equation ax+by+c=0.

### (4) Camera matrix V.S. essential matrix

We were quite confused by camera matrix and essential matrix, but since our 2D correspondent points aren’t multiplied by $K^{-1}$, we used camera matrix as input of triangulation.

### (5) Using Matlab function in python

To use the Matlab engine in python, we need to set up the config in Matlab root. Besides, the data types between Matlab and python are different, therefore, we have to change ndarray to list and change the element to Matlab type.

### (6) About our own photo experiment

As we can see, the results of 3D reconstruction seem not better than those of given datas. Maybe it’s because the value of intrinsic matrix is

not precisely like the real one of the camera. Wrong K may cause the wrong calculation of essential matrix. Also, if we use the interpolation to downsize the photo for fewer calculation time, the lower resolution(getting blur or unclear) of the photo may affect the keypoint detection. Less keypoints affect the robustness of the fundamental matrix calculation after RANSAC.

## 5. Conclusion

The interpretation of structure from motion is examined from a computational point of view. The question addressed is how the three-dimensional structure and motion of objects can be inferred from the two-dimensional transformations of their projected images when no three-dimensional information is conveyed by the individual projections.

While the current state-of-the-art SfM algorithms can handle the diverse and complex distribution of images in large-scale Internet photo collections, it seems like they frequently fail to produce fully satisfactory results in terms of completeness and robustness.