NCTU 2019 CV HW4:
Structure from Motion (SfM)

Hsin-Yu Chen, Yuan-Syun Ye

May 2019

1 Introduction

In HW4, we need to compute the camera parameters and the 3D points coordinates by corresponding points in two or more images. First, we calculate feature points by SIFT implemented in HW3. Next, we use RANSAC to find the fundamental matrix of the least outliner by 8-point Algorithm, then we can draw the epipolar lines. Finally, we compute four possible camera matrix then pick the right one to project 2D points back to 3D by Linear Triangulation and Inner product.

2 Implementation

2.1 Feature matching by SIFT features

Like what we have done in HW3. We use SIFT to find the feature points, and use the brute-force matcher method and ratio test to find the corresponding points of the two images. The code is Figure 1. The result is Figure 2.

```
def get_feature_points(img1, img2):
    sift = cv2.xfeatures2d.SIFT_create()

    (kp1, des1) = sift.detectAndCompute(image1, None)
    (kp2, des2) = sift.detectAndCompute(image2, None)

    matches = hw3.brute_force_matcher(des1, des2, BF_MACTHER_DISTANCE)
    matched_pt_order = hw3.sort_matched_points(matches)
    imgpts1, imgpts2 = hw3.get_matched_points(matched_pt_order, kp1, kp2)

    return imgpts1, imgpts2

imgpts1, imgpts2 = get_feature_points(image1, image2)
matched_feature_image = hw3.show_matched_image(image1, image2, imgpts1, imgpts2, draw_line=False, circle_size=10)
fig = plt.figure(figsize=(15,10))
plt.imshow(matched_feature_image), plt.axis('off'), plt.show()
```

Figure 1 using SIFT feature to match feature

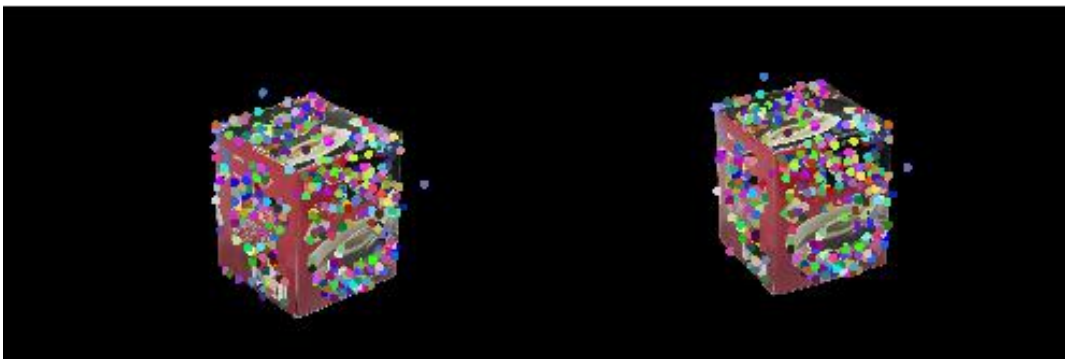


Figure 2 SIFT Result

2.2 Estimate Fundamental matrix and Essential matrix by RANSAC

2.2.1 Normalize image coordinates

Before computing Fundamental matrix, we need to Normalize image coordinates first in order to improve the stability of the calculation. The code is in Fig 3

Let points in both images' mean = 0 a standard deviation ~= (1,1,1)

X and X' are pair of corresponding points in two images

And we need to compute N_x and N_x'

$$\hat{x} = N_x x \quad \text{and} \quad \hat{x}' = N_x' x'$$

We set normalized matrix N_x as:

$$N_x = \begin{pmatrix} s_x & 0 & -s_x\overline{x} \\ 0 & s_y & -s_y\overline{y} \\ 0 & 0 & 1 \end{pmatrix}.$$

Where $\overline{x}$, $\overline{y}$ is the average of the original points, σ is variances of the original points

$$S_x = \sqrt{2/\sigma_x^2} \text{ and } S_y = \sqrt{2/\sigma_y^2}.$$

```python
def get_normalization_matrix(img_shape):
    '''
        get the normalization matrix
    '''
    T = np.array([[2/img_shape[1], 0, -1],
                  [0, 2/img_shape[0], -1],
                  [0, 0, 1]])
    return T

def normalization(imgpts1, imgpts2, img1, img2):
    '''
        ref: lecture P.54
    '''
    # t1: image1 normalization matrix, t2: image2 normalization matrix
    t1 = get_normalization_matrix(img1.shape)
    t2 = get_normalization_matrix(img2.shape)

    # to homography coordinate
    homopts1 = np.array([ [each[0], each[1], 1.0] for each in imgpts1])
    homopts2 = np.array([ [each[0], each[1], 1.0] for each in imgpts2])

    num_of_point = len(imgpts1)
    for i in range(num_of_point):

        # the Homogeneous coefficient should be one
        p2h = np.matmul(t1, homopts1[i])
        homopts1[i] = p2h/p2h[-1]

        p2h1 = np.matmul(t2, homopts2[i])
        homopts2[i] = p2h1/p2h1[-1]

    normalpts1 = np.delete(homopts1, -1, axis=1)
    normalpts2 = np.delete(homopts2, -1, axis=1)

    return normalpts1, normalpts2, t1, t2
```

Figure 3 normalization process

2.2.2 Compute Fundamental matrix by 8 points Algorithm and De-normalize it

X and X' are pair of corresponding points in two images.

The fundamental matrix is defined by the equation:

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

That is, while

X = (x, y, 1) and X' = (x, y, 1)

$$\mathbf{F} = \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix}$$

The equation corresponding to a pair of points (x, y, 1) and (x, y, 1) is:

$$x'xf_{11} + x'yf_{12} + x'f_{13} + y'xf_{21} + y'yf_{22} + y'f_{23} + xf_{31} + yf_{32} + f_{33} = 0.$$

Then change equation to vector inner product form:

$$(x'x, x'y, x', y'x, y'y, y', x, y, 1)\,\mathbf{f} = 0.$$

It can also be written as the follow form:

$$\mathbf{Af} = \begin{bmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ x'_nx_n & x'_ny_n & x'_n & y'_nx_n & y'_ny_n & y'_n & x_n & y_n & 1 \end{bmatrix} \mathbf{f} = \mathbf{0}.$$

Then, we compute Fundamental matrix by **8-point Algorithm**:

Linear solution: solve f from Af = 0 using SVD

Constraint enforcement: Resolve det(F)=0 constraint using SVD

De-normalize by normalized matrix N_x

$$F = N_x'^{\mathsf{T}} F N_x$$

The code is in Figure 4

```python
def denormalize_fundamental_mat(normalmat1, normalmat2, normalize_fundamental):
    '''
        ref: Multiple View Geometry in Computer Vision - Algorithm 11.1
    '''
    F = np.matmul(np.matmul(normalmat2.T, normalize_fundamental), normalmat1)
    F = F/F[-1,-1]
    return F

def get_fundamental(normalpts1, normalpts2):
    '''
        ref: Multiple View Geometry in Computer Vision - Chapter 11.1, lecture P.50
    '''

    A = np.zeros((len(normalpts1), 9), dtype=np.float64)
    for i in range(len(normalpts1)):
        x1, y1 = normalpts1[i][0], normalpts1[i][1]
        x2, y2 = normalpts2[i][0], normalpts2[i][1]
        A[i] = np.array([x2*x1, x2*y1, x2, y2*x1, y2*y1, y2, x1, y1, 1])

    # resolve det(f) = 0
    u, s, v = np.linalg.svd(A)
    F = v[-1]
    F = F.reshape(3, 3)
    u, s, v = np.linalg.svd(F)
    s = np.array([[s[0], 0 ,0],
                  [0, s[1], 0],
                  [0 , 0, 0]])
    F = np.matmul(np.matmul(u, s), v)

    return F
```

Figure 4 de-normalize fundamental matrix and compute fundamental matrix process

2.2.3 Compute Essential matrix

The defining equation for the essential matrix is:

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

Where

$$\hat{\mathbf{x}} = \mathrm{K}^{-1}\mathbf{x}.$$

So it can be written as:

$$\mathbf{x}'^{\mathrm{T}}\mathrm{K}'^{-\mathrm{T}}\mathrm{E}\mathrm{K}^{-1}\mathbf{x} = 0.$$

And

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

That is, relationship between the fundamental and essential matrices is:

$$\mathrm{E} = \mathrm{K}'^{\mathrm{T}}\mathrm{F}\mathrm{K}.$$

And we have already Estimate Fundamental matrix by RANSAC so we can compute essential matrices. The code is in Figure 5.

```python
def get_essential_mat(K1, K2, F):
    '''
        ref: Multiple View Geometry 9.12
    '''
    if(K1[-1,-1] != 1):
        K1 = K1 / K1[-1,-1]
    if(K2[-1,-1] != 1):
        K2 = K2 / K2[-1,-1]

    E = np.dot( K2.T , np.dot(F,K1))

    return E
```

Figure 5 essential matrix process

2.2.4 RANSAC

RANSAC is a way of randomly sampling feature points and uses the sampled feature points to calculate fundamental matrix, while the more inliner number of fundamental matrix is the best solution. Here we choose 8 for the number of random samples. The Sampson distance is used to compute geometric error for each fundamental matrix, which is defined as:

$$\sum_i \frac{(\mathbf{x}_i'^{\mathsf{T}}\mathbf{F}\mathbf{x}_i)^2}{(\mathbf{F}\mathbf{x}_i)_1^2 + (\mathbf{F}\mathbf{x}_i)_2^2 + (\mathbf{F}^{\mathsf{T}}\mathbf{x}_i')_1^2 + (\mathbf{F}^{\mathsf{T}}\mathbf{x}_i')_2^2}.$$

We determine the inliner threshold to 0.000005, and to repeat 2500 times. The code is in Figure 6 and Figure 7.

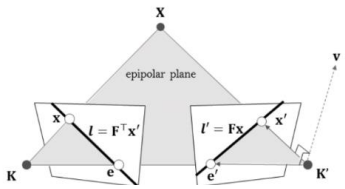
```python
def get_geometric_error(testpts1, testpts2, fundamentalmat, inliner_threshold):
    '''
        ref: Multiple View Geometry 11.4.3
    '''
    error = 0
    inliner_number = 0
    inlinerpt_indexs = np.zeros((len(testpts1),1), dtype=np.int)


    # transform test points to homography coordinate
    x1 = np.array([ [each[0], each[1], 1.0] for each in testpts1]).T
    x2 = np.array([ [each[0], each[1], 1.0] for each in testpts2]).T


    fx = np.dot(fundamentalmat.T, x1)
    ftx = np.dot(fundamentalmat, x2)

    d = np.power(fx[0], 2) + np.power(fx[1], 2) + np.power(ftx[0], 2) + np.power(ftx[1], 2)
    m = np.diag(np.dot(np.dot(x2.T, fundamentalmat), x1))
    m = np.power(m, 2)
    sampson = m/d

    for i in range(sampson.shape[0]):
        if(sampson[i] <= inliner_threshold):
            error += sampson[i]
            inlinerpt_indexs[inliner_number] = i
            inliner_number += 1

    return error, inliner_number, inlinerpt_indexs[:inliner_number, :]
```

Figure 6 compute inliner threshold by Sampson distance

```python
def find_fundamental_by_RANSAC(imgpts1, imgpts2, img1, img2, inliner_threshold, ransac_iteration = 2000, random_seed = None):
    '''
        ref: Multiple View Geometry 11.6
    '''

    best_fundamental = np.zeros((3,3))
    best_inlinernum = -1

    print("Key Point Number\n", len(imgpts1))
    ransac_sample_number = 8

    # normalization the key points
    normalpts1, normalpts2, nomalmat1, normalmat2 = normalization(imgpts1, imgpts2, img1, img2)
    best_error = 0
    best_inlinerpt_index = []

    for i in range(ransac_iteration):

        sampts1, sampts2 = sample_points(normalpts1, normalpts2, ransac_sample_number, random_seed)
        unnormalized_f = get_fundamental(sampts1, sampts2)
        error, inlinernum, inlinerpt_indexs = get_geometric_error(normalpts1, normalpts2, unnormalized_f, inliner_threshold)

        if(inlinernum > best_inlinernum):
            best_error = error
            best_fundamental = unnormalized_f
            best_inlinernum = inlinernum
            best_inlinerpt_index = inlinerpt_indexs

    # homographs coefficient and denormalize the fundamental matrix
    best_fundamental = denormalize_fundamental_mat(nomalmat1, normalmat2, best_fundamental)
    best_essential = get_essential_mat(intrinsic_matrix1, intrinsic_matrix2, best_fundamental)
    best_inlinerpts1, best_inlinerpts2 = get_inliner_points(imgpts1, imgpts2, best_inlinerpt_index)

    print("RANSC Error\n", best_error)
    print("Inliner Number\n", best_inlinernum)

    return best_fundamental, best_essential, best_inlinerpts1, best_inlinerpts2
```

Figure 7 using RANSAC to find the most inliner number of fundamental matrix

2.3 Draw epipolar lines



The function of epipolar lines is defined as:

$$\mathbf{l} = \mathbf{F}^{\top}\mathbf{x}' \text{ and } \mathbf{l}' = \mathbf{F}\mathbf{x}$$

Now, we have the equation of epipolar lines of both image, then we substituting two points into equations and draw the line on the image. The code is in Figure 8., and result is in Figure 9.

```python
def draw_epilines(img1, img2, lines, pts1, pts2, colors):
    '''
        ref: https://docs.opencv.org/3.4.4/da/de9/tutorial_py_epipolar_geometry.html
        x0, y0 = (0, -b/c)
        x1, y1 = (w, -(aw+c)/b)
    '''
    imgA = np.copy(img1)
    imgB = np.copy(img2)
    h, w, _ = img1.shape

    i=0
    for r,pt1, pt2 in zip(lines, pts1, pts2):
        x0, y0 = (0, int(-r[2]/r[1]))
        x1, y1 = (w, int(-(r[0]*w+r[2])/r[1]))
        imgA = cv2.line(imgA, (x0, y0), (x1, y1), colors[i], 5)
        imgA = cv2.circle(imgA, (int(pt1[0]), int(pt1[1])), 15, (255, 0, 0), -1)
        imgB = cv2.circle(imgB, (int(pt2[0]), int(pt2[1])), 15, (255, 0, 0), -1)
        i += 1
    return imgA, imgB
```

Figure 8 draw epipolar lines process

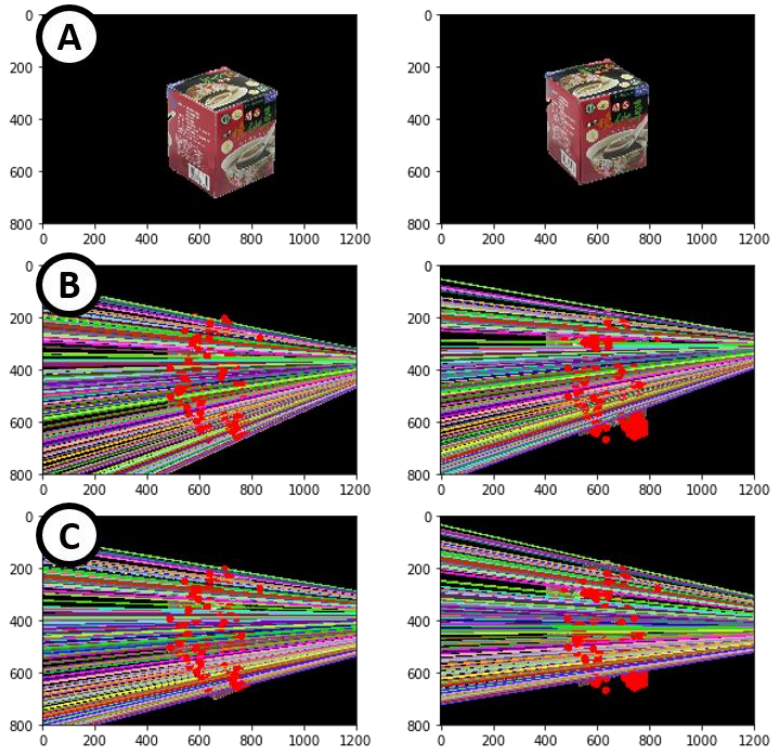


Figure 9 Epipolar lines result, draw with (a) original image, (b) our implement, and (c) Open CV

2.4 Estimate four possible camera matrix

Essential matrix E can present by ***R*** and ***t***:

$$\mathbf{E} = \mathbf{K}'^{\top}\mathbf{F}\mathbf{K} = [\mathbf{t}]_{\times}\mathbf{R}$$

Due to the lack of rank caused by the outer product operation and the orthogonality of the rotation matrix, the singular value decomposition of the essential matrix ***E*** has a special structure, which is:

$$\mathbf{E} = \mathbf{U}\Sigma\mathbf{V}^{\top} = \mathbf{U}\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 0\end{bmatrix}\mathbf{V}^{\top}$$

First, we defined:

$$\mathbf{Z} = \begin{bmatrix}0 & 1 & 0\\ -1 & 0 & 0\\ 0 & 0 & 0\end{bmatrix} \quad \mathbf{W} = \begin{bmatrix}0 & -1 & 0\\ 1 & 0 & 0\\ 0 & 0 & 1\end{bmatrix}$$

Where

$$\mathbf{ZW} = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 0\end{bmatrix} = \Sigma$$

Then we can write Essential matrix ***E*** as:

$$\mathbf{E} = \mathbf{UZWV}^{\top}$$

U is orthogonal matrix ($U^\top U=UU^\top=I$), so we get the equation:

$$\mathbf{E} = \mathbf{U}\mathbf{Z}\mathbf{U}^\top\mathbf{U}\mathbf{W}\mathbf{V}^\top$$

And define ***S*** and ***R'*** as:

$$\mathbf{S} = \mathbf{U}\mathbf{Z}\mathbf{U}^\top \text{ and } \mathbf{R}' = \mathbf{U}\mathbf{W}\mathbf{V}^\top$$

Now, we can write equation as: ***E = SR'***, which is similar with $\boldsymbol{E = [t]_x R}$, we want to check whether they are same:

$$\begin{aligned}\mathbf{S} = \mathbf{U}\mathbf{Z}\mathbf{U}^\top &= \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} u_{11} & u_{21} & u_{31} \\ u_{12} & u_{22} & u_{32} \\ u_{13} & u_{23} & u_{33} \end{bmatrix} \\ &= \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{bmatrix} \begin{bmatrix} u_{12} & u_{22} & u_{32} \\ -u_{11} & -u_{21} & -u_{31} \\ 0 & 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & u_{11}u_{22} - u_{12}u_{21} & u_{11}u_{32} - u_{12}u_{31} \\ u_{12}u_{21} - u_{11}u_{22} & 0 & u_{21}u_{32} - u_{22}u_{31} \\ u_{12}u_{31} - u_{11}u_{32} & u_{22}u_{31} - u_{21}u_{32} & 0 \end{bmatrix}\end{aligned}$$

U is orthogonality, where $u_{*3}=u_{*1}\times u_{*2}$ (u_{*i} is ith column of U). S can be written as:

$$\mathbf{S} = \begin{bmatrix} 0 & u_{33} & -u_{23} \\ -u_{33} & 0 & u_{13} \\ u_{23} & -u_{13} & 0 \end{bmatrix}$$

The outer product of the vector can be described by a 3 × 3 matrix, defined as:

$$[\mathbf{u}]_\times = \begin{bmatrix} 0 & -u_3 & u_2 \\ u_3 & 0 & -u_1 \\ -u_2 & u_1 & 0 \end{bmatrix}$$

And ***R'*** is an orthogonal matrix because:

$$\mathbf{R}'\mathbf{R}'^\top = \mathbf{U}\mathbf{W}\mathbf{V}^\top\mathbf{V}\mathbf{W}^\top\mathbf{U}^\top = \mathbf{I} = \mathbf{V}\mathbf{W}^\top\mathbf{U}^\top\mathbf{U}\mathbf{W}\mathbf{V}^\top = \mathbf{R}'^\top\mathbf{R}'$$

So

$$\boldsymbol{[t]_x = S} \text{ and } \boldsymbol{R = R'}$$

According to the relationship between the outer product operation and the antisymmetric matrix, we can get ***t***:

$$\boldsymbol{t = (s_{32}, s_{13}, s_{21})^T = (u_{31}, u_{32}, u_{33})^T}$$

Because $\boldsymbol{-E = -[t]_x R = [-t]_x R}$ is also an essential matrix equivalent to ***E***, there are two possible solutions of ***t***, where:

$$\boldsymbol{t_1 = (s_{32}, s_{13}, s_{21})^T = (u_{31}, u_{32}, u_{33})^T} \text{ and } \boldsymbol{t_2 = -t_1}$$

Also, Σcan be write as another way:

$$-\mathbf{Z}\mathbf{W}^\top = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \Sigma$$

So Essential matrix ***E*** can also be defined as:

$$\boldsymbol{E = -UZU^T UW^T V^T}$$

And get another set of solutions:

$$S = -UZU^T \text{ and } R' = UW^TV^T$$

That is, we get two possible solutions of R:

$$R_1 = UWV^T \text{ and } R_2 = UW^TV^T$$

So the four sets of possible solutions ***(R1, t1), (R1, t2), (R2, t1) and (R2, t2)*** are generated. That is, for a given essential matrix $E = U\sum V^T$, and first camera matrix $P = [I \mid 0]$, there are four possible choices for the second camera matrix P , namely:

$$P = [UWV^T \mid +u_3] \text{ or } [UWV^T \mid -u_3] \text{ or } [UW^TV^T \mid +u_3] \text{ or } [UW^TV^T \mid -u_3].$$

The code is in Figure 10.

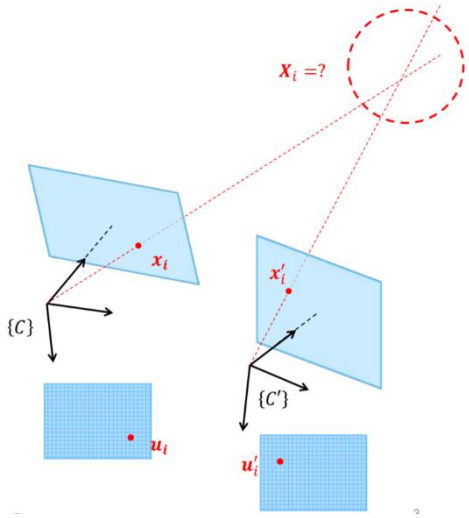
```
def get_external(e, index=None):
    '''
        get external parameter
        ref: homework lecture page 3
    '''
    w = np.array([[0.0, -1.0, 0.0],[1.0, 0.0, 0.0],[0.0, 0.0, 1.0]])
    z = np.array([[0.0, 1.0, 0.0],[-1.0, 0.0, 0.0],[0.0, 0.0, 0.0]])

    U, S, V = np.linalg.svd(e)
    m = (S[0] + S[1])/2
    S[0] = m
    S[1] = m
    S[2] = 0

    T1 = U[:,2]
    T2 = -U[:,2]
    R1 = np.matmul(np.matmul(U, w.T), V)
    R2 = np.matmul(np.matmul(U, w), V)

    if(np.linalg.det(R1)<0):
        R1 = -1*R1
    if(np.linalg.det(R2)<0):
        R2 = -1*R2

    if(index is None):
        return T1, T2, R1, R2
    elif(index == 0):
        return T1, R1
    elif(index == 1):
        return T1, R2
    elif(index == 2):
        return T2, R1
    elif(index == 3):
        return T2, R2
```

Figure 10 estimate four camera matrix

2.5 Project 2D points back to 3D

2.5.1 Triangulation

Now we have camera matrices P, P' and 2D correspondences $\boldsymbol{u}_i \leftrightarrow \boldsymbol{u}_i'$, and we want to reconstructed 3D point $\boldsymbol{X}_i$.



In the method of Linear Triangulation, we get two equations from each perspective camera model. And we need to solve this system of equations with SVD to reconstructed 3D point X_i.

$$\tilde{u}_i = P\tilde{X}_i \qquad \tilde{u}'_i = P'\tilde{X}_i$$

$$\Downarrow \qquad \Downarrow$$

$$\tilde{u}_i \times P\tilde{X}_i = 0 \qquad \tilde{u}'_i \times P'\tilde{X}_i = 0$$

$$\begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} \times \begin{bmatrix} p^{1T} \\ p^{2T} \\ p^{3T} \end{bmatrix} \tilde{X}_i = 0 \qquad \begin{bmatrix} u'_i \\ v'_i \\ 1 \end{bmatrix} \times \begin{bmatrix} p'^{1T} \\ p'^{2T} \\ p'^{3T} \end{bmatrix} \tilde{X}_i = 0$$

$$\begin{bmatrix} v_i p^{3T} - p^{2T} \\ p^{1T} - u_i p^{3T} \\ u_i p^{2T} - v_i p^{1T} \end{bmatrix} \tilde{X}_i = 0 \qquad \begin{bmatrix} v'_i p'^{3T} - p'^{2T} \\ p'^{1T} - u'_i p'^{3T} \\ u'_i p'^{2T} - v'_i p'^{1T} \end{bmatrix} \tilde{X}_i = 0$$

$$\Updownarrow \qquad \Updownarrow$$

$$\begin{bmatrix} v_i p^{3T} - p^{2T} \\ u_i p^{3T} - p^{1T} \end{bmatrix} \tilde{X}_i = 0 \qquad \begin{bmatrix} v'_i p'^{3T} - p'^{2T} \\ u'_i p'^{3T} - p'^{1T} \end{bmatrix} \tilde{X}_i = 0$$

Combining these equations:

$$\begin{bmatrix} v_i p^{3T} - p^{2T} \\ u_i p^{3T} - p^{1T} \\ v'_i p'^{3T} - p'^{2T} \\ u'_i p'^{3T} - p'^{1T} \end{bmatrix} \tilde{X}_i = 0$$

That is,

$$\begin{bmatrix} v_i p_{31} - p_{21} & v_i p_{32} - p_{22} & v_i p_{33} - p_{23} & v_i p_{34} - p_{24} \\ u_i p_{31} - p_{11} & u_i p_{32} - p_{12} & u_i p_{33} - p_{13} & u_i p_{34} - p_{14} \\ v'_i p'_{31} - p'_{21} & v'_i p'_{32} - p'_{22} & v'_i p'_{33} - p'_{23} & v'_i p'_{34} - p'_{24} \\ u'_i p'_{31} - p'_{11} & u'_i p'_{32} - p'_{12} & u'_i p'_{33} - p'_{13} & u'_i p'_{34} - p'_{14} \end{bmatrix} \tilde{X}_i = 0$$

Then we defined the matrix as A and solve this system of equations by SVD.

$$A\tilde{X}_i = 0$$

The code is in Figure 11 linear triangulation process

```python
def linear_LS_Triangulation(x1, p1, x2, p2):
    '''
        ref: 1995 Triangulation ch5.1 Linear Triangulation
        ref: Multiple View Geometry 12.2
        https://perception.inrialpes.fr/Publications/1997/HS97/HartleySturm-cviu97.pdf
    '''

    A = np.array([x1[0]*p1[2,:] - p1[0,:],
                  x1[1]*p1[2,:] - p1[1,:],
                  x2[0]*p2[2,:] - p2[0,:],
                  x2[1]*p2[2,:] - p2[1,:]])

    U, S, V = np.linalg.svd(A)
    X = V[-1]/V[-1,3]
    return X
```

Figure 11 linear triangulation process

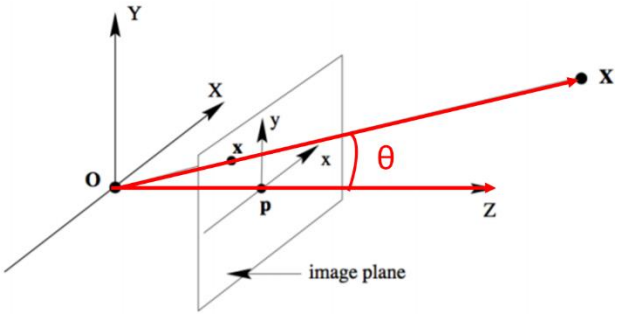
2.5.2 Pick the solution with most of 3D points in front of cameras

We want to pick the solution with most of 3D points in front of cameras. The method we check whether a 3D points ***X*** is in front of cameras is calculate Inner product of View Direction and vector from Camera Center to 3D points ***X***.

Because Inner product of two vector ***U*** and ***V*** is:

$$\mathbf{U} \cdot \mathbf{V} = \| \mathbf{U} \| \ \| \mathbf{V} \| \cos\theta$$

And cos θ is positive while θ is between 90 to -90 degrees. That is, if the value of Inner product is positive then this point X is in front of the cameras.



First, we have ***R*** and ***t***, which can turn points in world coordinate system into camera coordinate system by the function:

$$\begin{bmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \end{bmatrix} = R \begin{bmatrix} X_{world} \\ Y_{world} \\ Z_{world} \end{bmatrix} + t$$

And we need to turn points in camera coordinate system into world coordinate system, so change the function into:

$$\begin{bmatrix} X_{world} \\ Y_{world} \\ Z_{world} \end{bmatrix} = R^{-1} \left(\begin{bmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \end{bmatrix} - t \right) = R^{\top} \begin{bmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \end{bmatrix} - R^{\top} t$$

Then we can compute Camera Center and View Direction in world coordinate system.

Camera Center ***C***:

$$\begin{bmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \Leftrightarrow \mathbf{C} = \begin{bmatrix} X_{world} \\ Y_{world} \\ Z_{world} \end{bmatrix} = R^{\top} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} - R^{\top}t = -R^{\top}t$$

View Direction

$$\underbrace{\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}}_{\text{camera coordinate system}} \Leftrightarrow \underbrace{\left(R^{\top} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} - R^{\top}t \right)}_{\text{world coordinate system}} - (\mathbf{C}) = \left(R(3,:)^{\top} - R^{\top}t \right) - \left(-R^{\top}t \right) = R(3,:)^{\top}$$

Next, calculate the Inner product and check whether it is positive.

$$(X - \mathbf{C}) \cdot R(3,:)^{\top} > 0?$$

We have to calculate number of points in front of cameras for each solution ***(R1, t1), (R1, t2), (R2, t1) and (R2, t2)*** and pick the solution with most points.

The code is in Figure 12 check project points is in front camera.

```
58 def in_front_of_camera(R, T, pts,is_opencv=False):
59     num_of_points = 0
60     camera_center = -np.dot(np.transpose(R),T)
61     vide_direciton = np.transpose(R)[2,:]
62     for i in range(pts.shape[0]):
63         if is_opencv:
64             hp = pts[:,i]
65             hp = hp/hp[-1]
66             X_location = hp[:3] - camera_center
67         else:
68             X_location = pts[i] - camera_center
69         if np.dot(X_location,vide_direciton) > 0 :
70             num_of_points = num_of_points + 1
71     return num_of_points
```

Figure 12 check project points is in front camera

3 Experimental

3.1 TA Images

The TA provides two test materials, namely, Mesona, and Statue, which correspond to Figure 13, Figure 14, respectively. Because of we randomly method sample the feature, the reconstruction will result is unstable. For each input image, we set different random seed, to show the best result. The triangulation of Mesona image is used our fundamental about the parameter: using SIFT feature with 0.65 pixels for match, RANSAC iteration 2000 times, 0.0008 inliner thresholds. About the Statue Image, it could use the proved rotation and translation matrix, so we just set the RANSAC thresholds 0.000003 for pick up the inliner to triangulate them.

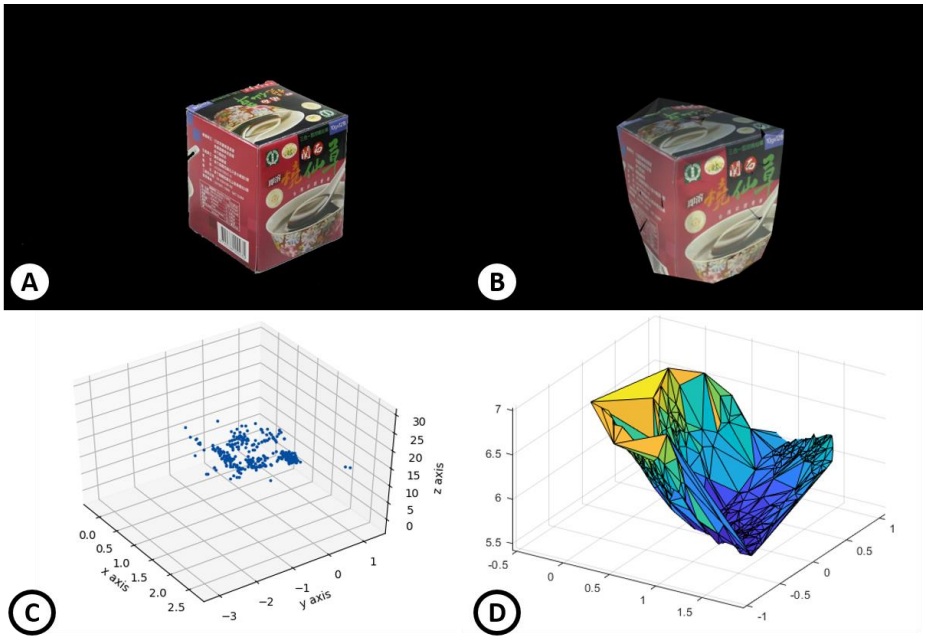


Figure 13 Reconstruction Mesona image, (a) is input image, (b) is the triangulation points, (c) is the mesh of cloud points, and (d) is textured by Unity3D.

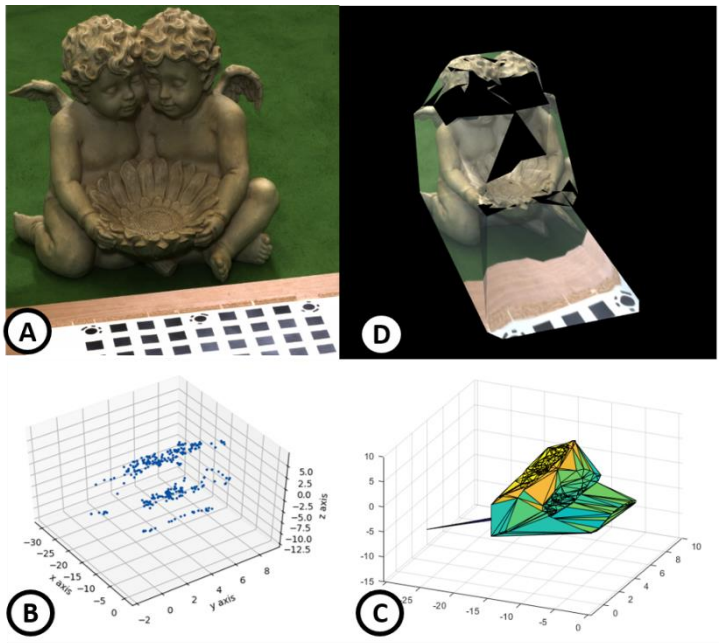


Figure 14 Reconstruction Statue image, (a) is input image, (b) is linear triangulation points, (c) is the mesh of the cloud points, and (d) is texture the mesh object by Unity3D

3.2 Our Images

Due to the camera-base reconstruction is dependent on the image feature, so we select a rich pattern box for our experiment target. The RANSAC iteration times and the inliner threshold is the same above mentioned, different is the feature match threshold, we set the 0.3. See the result in Figure 15, the feature points distribute on the box surface and beside the pattern of the box(Image B and D).

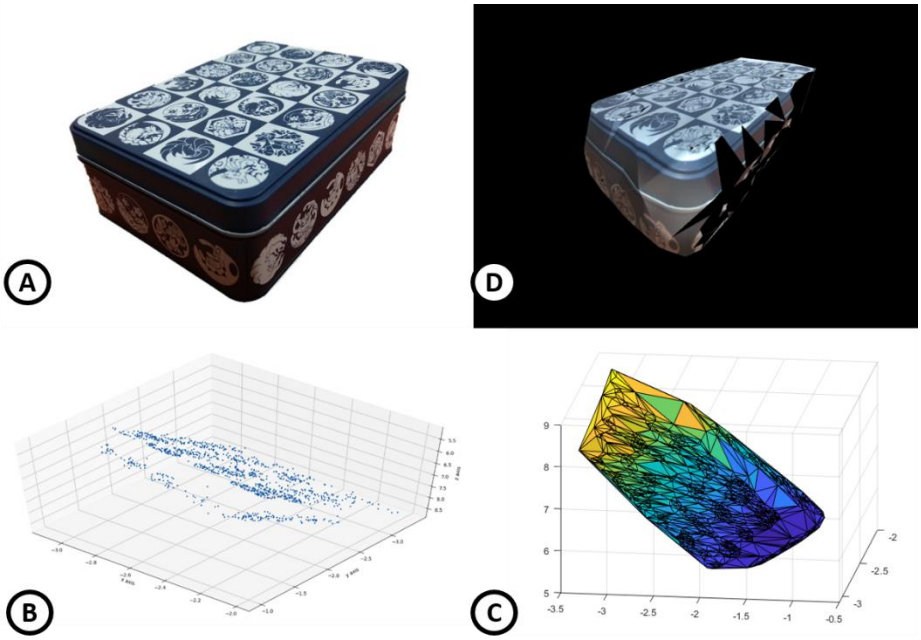


Figure 15 Reconstruction a patterned box image, (a) is input image, (b) is the triangulation points, (c) is the mesh of cloud points, and (d) is textured by Unity3D

3.3 Compare with Open CV

To check the results of our implementation, we compare the result with Open CV in Figure 9 by epipolar lines result and also show the Fundamental matrix estimated by each way. It can be found that our Fundamental matrix and the matrix Open CV computed are different, probably because our RANSAC did not completely select all the inliners. Comparing the calculated Fundamental matrix, ours is:

$$\mathrm{F} = \begin{pmatrix} 0.00000058 & -0.00006382 & 0.02252994 \\ 0.00007623 & -0.00000086 & -0.10828925 \\ -0.02433268 & 0.0920117 & 1. \end{pmatrix}$$

And Open CV is:

$$\mathrm{F} = \begin{pmatrix} 0.0.00000085 & 0.00001079 & -0.00566836 \\ -0.00001176 & 0.00000077 & 0.02057989 \\ 0.0035924 & -0.01913668 & 1. \end{pmatrix}$$

It can be seen that there is a difference between the two.

4 Discussion

There are two methods to estimate Fundamental matrix, 8-point algorithm and 7-point algorithm respectively, where 8-point algorithm is simplest method, just need to compute least squares solution of a set of linear equations using SVD, and that is what we did in the homework. Compare with 8-point algorithm, 7-point algorithm is faster (need fewer points) and could be more robust (fewer points), but there would have one or three real solutions, cause more case need to deal with, also the method is more complex.

Another thing to note is that intrinsic matrix K given for first image TA provide. The k_{33} is 0.001，we need to divide intrinsic matrix K by 0.001 to turn k_{33} to 1.Next, we would confuse about why the function TA given in pdf is: ***P = K [R, -RT]*** ,and finally we found out that ***t*** given for second image is the location of the camera-center in world coordinates, while ***R*** is the rotation matrix in camera coordinates.

In RANSAC, we can use the de-normalized fundamental matrix or normalized fundamental to check the inliner points number. wrong inliner number will cause RANSAC to return error fundamental matrix. Due to the normalizing and normalizing process will have calculation error, for reducing the error we count the inliner points used normalized fundamental matrix.

5 Conclusion

We implement Structure from Motion(SfM), used SIFT to find feature points, and used the Brute-Force method to pair feature points, then use 8-point algorithm to estimate fundamental matrix, find the best fundamental matrix by RANSAC and draw the epipolar lines. Then, we compute four possible camera matrix and do Linear Triangulation to reconstructed 3D respectively. Finally, we pick the solution with most of 3D points in front of cameras and project 2D points back to 3D.

6 Work Assignment Plan

This homework divided into two parts. Yuan-Syun Ye is responsible for the part of cording and checks this report. Hsin-Yu Chen is responsible for the writing of the report.

References

http://cvrs.whu.edu.cn/downloads/ebooks/Multiple%20View%20Geometry%20in%20Computer%20Vision%20(Second%20Edition).pdf

https://www.researchgate.net/publication/303522230_qiantanjichujuzhenbenzhijuzhenyuxiangjiyidong_Beginner%27s_Guide_to_Fundamental_Matrix_Essential_Matrix_and_Camera_Motion_Recovery?fbclid=IwAR3chaM6yPcNxyrD2vTfzR3tgcejHuzF-jZXXgi8_B0AFNWeLa7tdWzbnHg