Computer Vision Assignment #4

0516234 黃聖祺 0516223 林恭白 0510532 楊上萱

A. Introduction

In this assignment, we are going to implement SfM(Structure from Motion). We have two images with different perspective on the same object. By finding the matching point and calculate the fundamental matrix. Estimate 3D positions after recovering the canonical camera projection.

B. Implementation Procedure

1. First of all, we use the SIFT algorithm as HW3 to find out keypoints, and pair the points to get the good matches.

```
kp_0, des_0 = find_img_keypoint(img_0)
kp_1, des_1 = find_img_keypoint(img_1)

good_matches, good_matches_for_img_show = find_good_matches(des_0, des_1)
```

   a. The matched points found by `knnmatch` ,then are determined by a threshold that if it is a good match by calculating the distance of them .

```
21      matches = bf.knnMatch(des_0, des_1, k=GOOD_MATCH_K)
```

   b. In our case we set k at 2

   c. We use 0.8 as good ratio to detect significantly better match.

```
24      for m, n in matches:
25          if m.distance < GOOD_DISTANCE_RATIO * n.distance:
26              good_matches_for_img_show.append([m])
27              good_matches.append(m)
```

   d. Then, choose top 25% of matched points

```
good_matches = sorted(good_matches, key=lambda x: x.distance)
num_good_matches = int(len(good_matches) * 0.25)
good_matches = good_matches[:num_good_matches]
return good_matches, good_matches_for_img_show
```

2. Then, to estimate the fundamental matrix we use RANSAC method to repeating select points and solve them.

```
for i in range(1000):
    idx = np.random.randint(0, len(normal_p0), 60)
```

```
f = compute_fundamental_normalized(match_points_0[idx], match_points_1[idx], shape)
```

   a. First we randomly select 60 points from the matched points list , then calculate the fundamental matrix by these random points.

b. To calculate the matrix we create the x matrix as the format below,

1. Solve a system of homogeneous linear equations
   a. Write down the system of equations

$$X^{\top}FX' = 0$$

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

$$\mathbf{Af} = \begin{bmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_nx_n & x'_ny_n & x'_n & y'_nx_n & y'_ny_n & y'_n & x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix} = \mathbf{0}$$

```
for i in range(n):
    A[i] = [x1[0,i]*x2[0,i], x1[0,i]*x2[1,i], x1[0,i]*x2[2,i],
            x1[1,i]*x2[0,i], x1[1,i]*x2[1,i], x1[1,i]*x2[2,i],
            x1[2,i]*x2[0,i], x1[2,i]*x2[1,i], x1[2,i]*x2[2,i] ]
```

c. Then we solve the linear equations by using SVD from Af=0,also resolve def(F)=0

1. Solve a system of homogeneous linear equations
   a. Write down the system of equations
   b. Solve **f** from **Af=0** using SVD

Matlab:

```
[U, S, V] = svd(A);
f = V(:, end);
F = reshape(f, [3 3])';
```

3. Resolve ***det(F)=0*** constraint using SVD

Matlab:

```
[U, S, V] = svd(F);
S(3,3) = 0;
F = U*S*V';
```

```
for i in range(n):
    A[i] = [x1[0,i]*x2[0,i], x1[0,i]*x2[1,i], x1[0,i]*x2[2,i],
            x1[1,i]*x2[0,i], x1[1,i]*x2[1,i], x1[1,i]*x2[2,i],
            x1[2,i]*x2[0,i], x1[2,i]*x2[1,i], x1[2,i]*x2[2,i] ]
```

d. Then get the distance by estimating the points by calculating |x'Fx| , the threshold is 0.01 in this case.

```
    extend_p0 = np.array([p0[0], p0[1], 1])
    extend_p1 = np.array([p1[0], p1[1], 1])
    disparity = abs((extend_p1.T @ f) @ extend_p0)
    # return np.linalg.norm(error)
    return disparity
```

e. Pick the Fundamental matrix with most inliners.

```
if len(inliers) > len(max_inliers):
    print("inliner length: {}".format(len(inliers)))
    max_inliers = inliers
    # max_inlier_h = h
    max_inlier_f = f
```

3. After getting the best Fundamental as we show the matching points, ans try to draw the epipolar lines

```
good_matches, good_matches_for_img_show = find_good_matches(des_0, des_1)
img = cv2.drawMatchesKnn(img_0, kp_0, img_1, kp_1, good_matches_for_img_show, None, flags=2)
cv2_img_show(img)
```

4. We get 4 possible solution of essential Matrix from Fundamental matrix

```
205  def compute_essential_candidate(f, K):
206      E = K.T @ f @ K
207      U, S, V = np.linalg.svd(E)
208      S = np.diag(S)
209
210      m = (S[1, 1] + S[2, 2]) / 2
211      E = U @ np.array([[m, 0, 0], [0, m, 0], [0, 0, 0]]) @ V.T
212
213      U, S, V = np.linalg.svd(E)
214      S = np.diag(S)
215      W = np.array([[0, -1, 0], [1, 0, 0], [0, 0, 1]])
216
217      P1 = K @ np.array([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0]])
218
219      # P2 chooses
220      P2_1 = np.zeros((3, 4))
221      P2_1[:, :-1] = U @ W @ V.T
222      P2_1[:, -1] = U[2]
223
224      P2_2 = np.zeros((3, 4))
225      P2_2[:, :-1] = U @ W @ V.T
226      P2_2[:, -1] = (-1)*U[2]
227
228      P2_3 = np.zeros((3, 4))
229      P2_3[:, :-1] = U @ W.T @ V.T
230      P2_3[:, -1] = U[2]
231
232      P2_4 = np.zeros((3, 4))
233      P2_4[:, :-1] = U @ W.T @ V.T
234      P2_4[:, -1] = (-1)*U[2]
235
236      P2_chooses = np.array([P2_1, P2_2, P2_3, P2_4])
237
238      return P1, P2_chooses
```

5. Calculate how many points are in front of camera with each essential matrix, and choose the one that has most points in front of camera as our best solution.

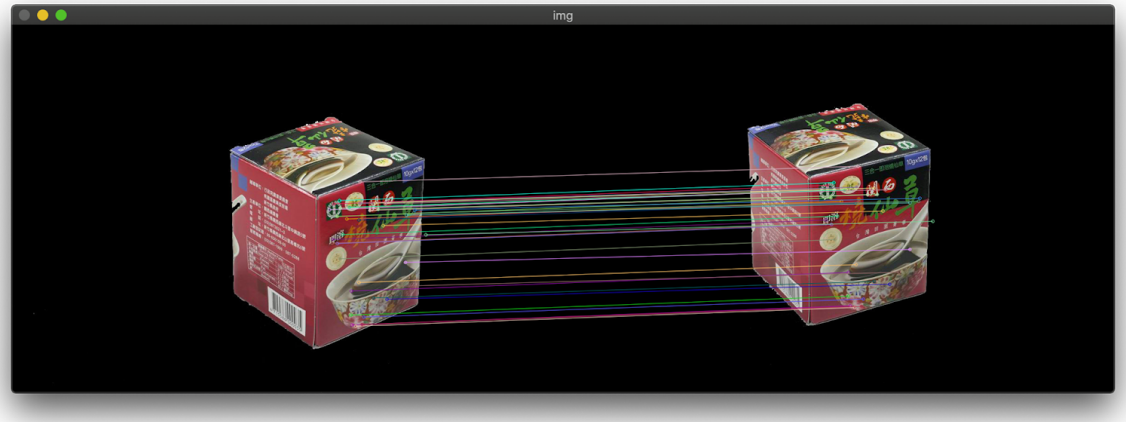
```python
def find_best_solution(x1, x2, P1, P2_chooses):
    ### add 1 for homogeneous ###
    x1_extend = np.ones((x1.shape[0], 3))
    x1_extend[:, :-1] = x1
    x2_extend = np.ones((x1.shape[0], 3))
    x2_extend[:, :-1] = x2

    ### find best solution between each P2_chooses ###
    cnt_of_front_points = np.zeros(4)
    for i, P2 in enumerate(P2_chooses):
        R = P2[:, :-1]
        t = P2[:,-1].reshape(3,1)
        ### Create matrix A ###

        for point_idx in range(x1_extend.shape[0]): # for each [u,v] pair
            p1 = P1[0, :]
            p2 = P1[1, :]
            p3 = P1[2, :]
            p1p = P2[0, :]
            p2p = P2[1, :]
            p3p = P2[2, :]

            u = x1[point_idx, 0]
            v = x1[point_idx, 1]
            up = x2[point_idx, 0]
            vp = x2[point_idx, 1]
            A = np.array([u  * p3.T - p1.T,
                          v  * p3.T - p2.T,
                          up * p3p.T - p1p.T,
                          vp * p3p.T - p2p.T])

            # U, S, VH = np.linalg.svd(A)
            A = np.array(A)
            A = A.reshape((4, 4))

            ### calculate X ###
            U, S, VH = np.linalg.svd(A)
            X = VH[-1]
            X = X / X[-1]
            X = X[:-1].reshape((3, 1))

            ### Count in-front-of-camera point ###
            C = (-1) * R.T @ t
            view_direction = (R[2, :].T).reshape(3,1)
            if ((X-C).T @ view_direction) > 0:
                # in front!
#                 print('inner product: ', (X-C).T @ view_direction)
                cnt_of_front_points[i] += 1
    print('cnt: ', cnt_of_front_points)

    max_p2_idx = np.argmax(cnt_of_front_points)
    return P2_chooses[max_p2_idx]
```

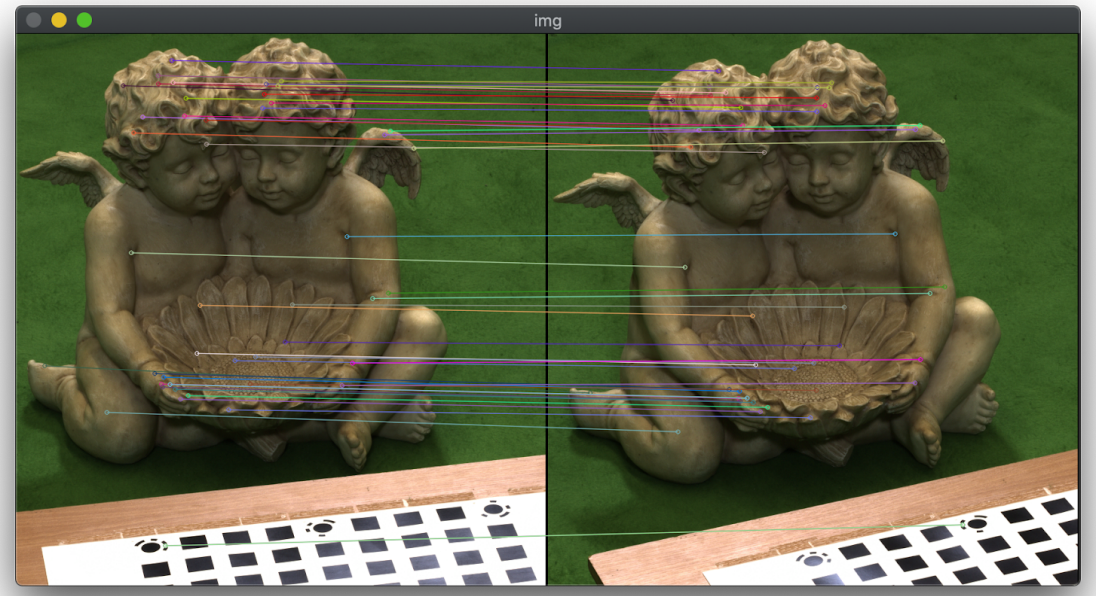
6. Do the triangulation to get 3D points.

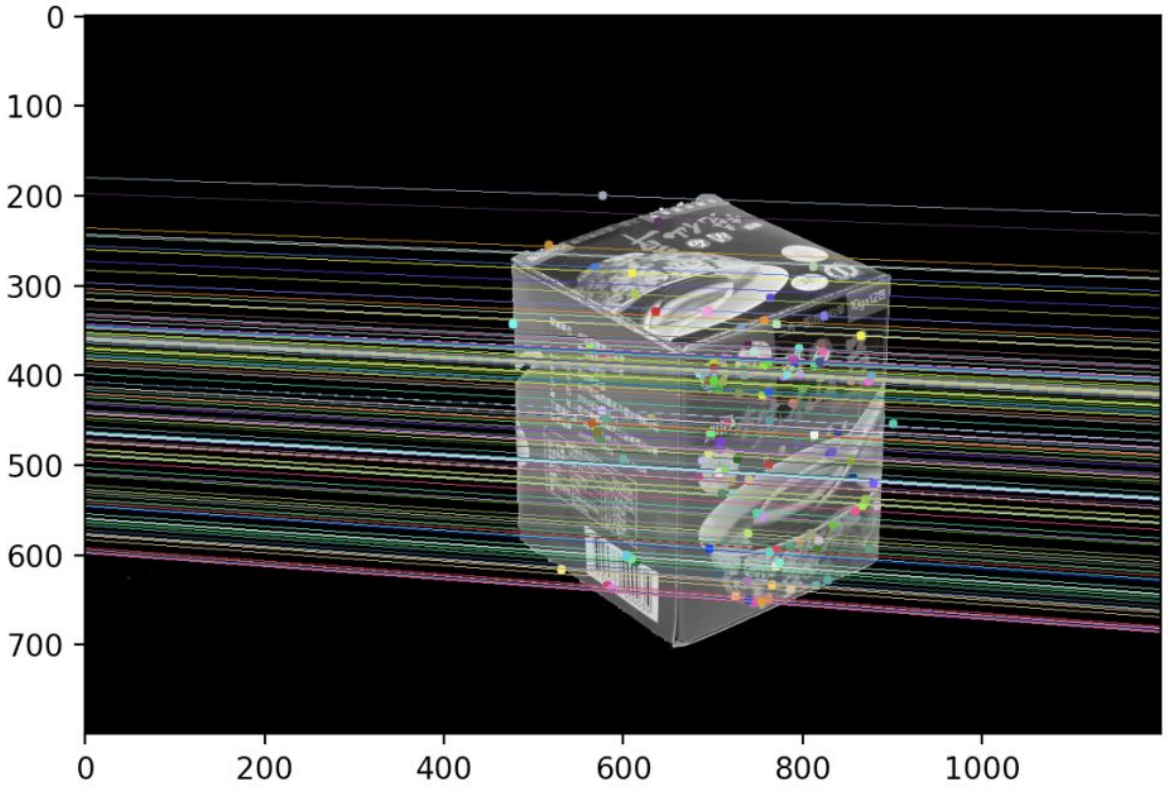
```python
def Triangulation_wesuck(x1,x2,P1,P2):
    X = []

    # print(u_1.shape)
    P2_1 = P2[0].reshape((P2[0].shape[0], 1))
    P2_2 = P2[1].reshape((P2[1].shape[0], 1))
    P2_3 = P2[2].reshape((P2[2].shape[0], 1))
    P1_1 = P1[0].reshape((P1[0].shape[0], 1))
    P1_2 = P1[1].reshape((P1[1].shape[0], 1))
    P1_3 = P1[2].reshape((P1[2].shape[0], 1))
    # print(P_1.shape)
    # print(u_1 @ P_0.T)
    for i in range(len(x1)):
        u_1, v_1 = x1[i, 0], x1[i, 1]
        u_2, v_2 = x2[i, 0], x2[i, 1]
        A = [u_1 * P1_3 - P1_1,
             v_1 * P1_3 - P1_2,
             u_2 * P2_3 - P2_1,
             v_2 * P2_3 - P2_2]
        A = np.array(A)
        A = A.reshape((4, 4))
        U, S, VH = np.linalg.svd(A)
        X1 = VH[-1]
        X1 = X1 / X1[-1]
        X1 = X1[:-1].reshape(3, 1)
        X.append(X1)

    return np.array(X)
```

C. Experimental Result

1. **Epipolar and matching points**







2. **Triangulation and matlab**

D. Discussion

1. **Appropriate amount of interest points?**
   Originally, we set the ratio of good distance ratio to be 0.3, which made the number of good matching points to be too little. So we tried a lot and find out that the result seemed to be good when the ratio was 0.8 and select the top 25% points. After that, we can get appropriate pairs of matching points.

2. **Appropriate geometric distance when doing RANSAC?**
   During the RANSAC part, we calculate the geometric distance $|x'Fx|$ and use a threshold d to points which are closer. After a lot of trials, we finally pick d to be 0.01.

3. **The randomness when picking points for RANSAC to compute fundamental matrix?**
   Due to the randomness of points for calculating fundamental matrix, it is possible that the picked points are not good enough so that the result of fundamental matrix and epipolar lines will be quite weird. Therefore, we need to try a few times to make sure the picked points are appropriate.

4. **The epipolar lines seemed to be correct, but the 3D-reconstructed structure becomes wrong?**
   The result of fundamental matrix and epipolar lines seemed to be correct. But the final result of 3D model was distorted. However, we had no idea which part was wrong.

E. Conclusion

1. Tuning of parameters (e.g. good_matching_ratio, gemotric_distance threshold, etc.) is required to reach a good enough result.
2. The randomness when picking points for RANSAC to calculate fundamental matrix will somehow affect the result.
3. We failed at reconstruct 3D model.

F. Work Assignment Plan

1. 黃聖祺-Coding
2. 林恭白-Coding, Report
3. 楊上萱-Coding