HW4 report

Group 5

0516222 許芳瑀 0516209 呂淇安 0516326 陳廷達

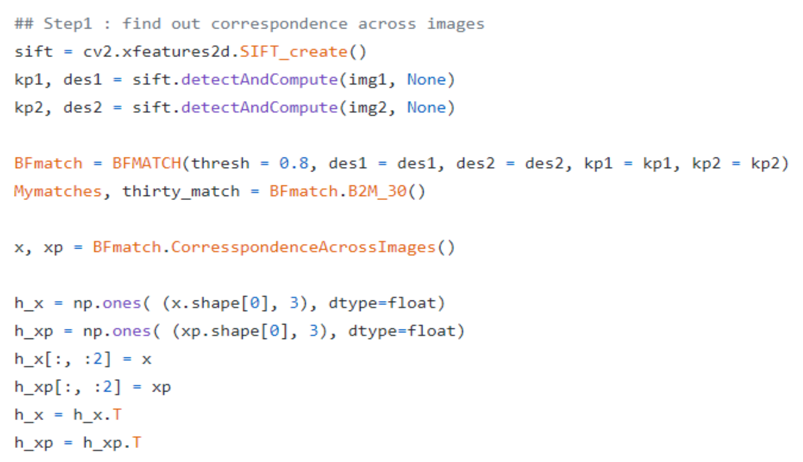
1. Introduction

The goal of this homework is using two pictures to reconstruct 3D model. In homework 3, we had learned the method of finding the same points in two pictures taken in different angles. In order to find fundamental matrix, we apply 8-points algorithm on previous points we found and ratio test. After getting fundamental matrix, we can calculate 4 possible essential matrices. The final step is examining 4 possible directions of camera, and then picking the best result with maximum number of points in front of camera.

2. Implementation procedure

(1) find out correspondence across images

This step is similar to hw3. The purpose is finding the corresponding points in 2 pictures. We use cv2 build-in function to reach the goal.



```
## Step1 : find out correspondence across images
sift = cv2.xfeatures2d.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)

BFmatch = BFMATCH(thresh = 0.8, des1 = des1, des2 = des2, kp1 = kp1, kp2 = kp2)
Mymatches, thirty_match = BFmatch.B2M_30()

x, xp = BFmatch.CorresspondenceAcrossImages()

h_x = np.ones( (x.shape[0], 3), dtype=float)
h_xp = np.ones( (xp.shape[0], 3), dtype=float)
h_x[:, :2] = x
h_xp[:, :2] = xp
h_x = h_x.T
h_xp = h_xp.T
```

(2) estimate the fundamental matrix across images (normalized 8 points)

By step 1, we get x and x’. First we nnormorlize the points.

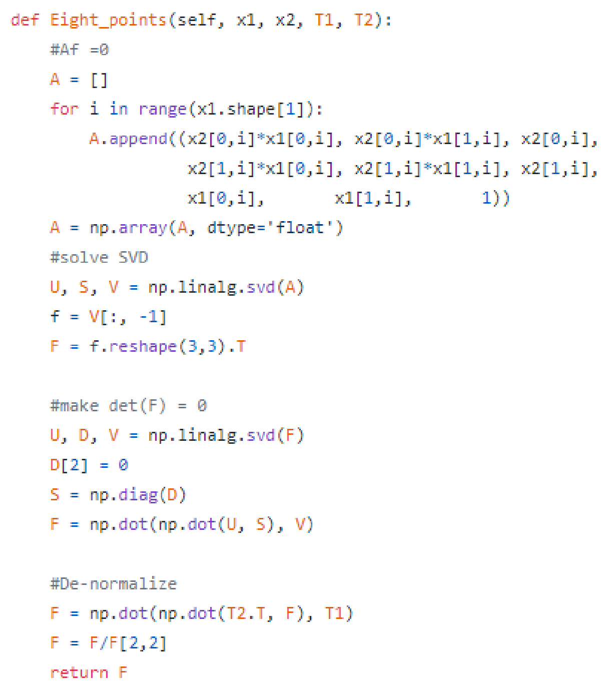
```
def normalize(points, imgsize):
    '''
    new_pt =
     [[x0, x1, x2 ...],
      [y0, y1, y2 ...],
      [1,  1,  1  ...]]
    '''
    T = np.array([[2/imgsize[1], 0, -1],
                  [0, 2/imgsize[0], -1],
                  [0, 0, 1]])
    new_pt = T.dot(np.array(points))
    return new_pt, T
```

And then apply the RANSAC with 8-points algorithm. The fundamental matrix is defined as x’ F x = 0. This can be written as

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

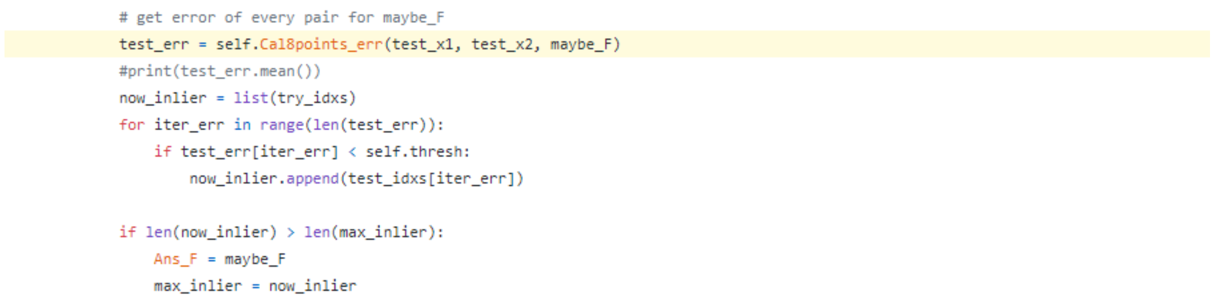
$$\mathbf{Af} = \begin{bmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_nx_n & x'_ny_n & x'_n & y'_nx_n & y'_ny_n & y'_n & x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\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

To solve the homogeneous linear equations, we use SVD



```
def Eight_points(self, x1, x2, T1, T2):
    #Af =0
    A = []
    for i in range(x1.shape[1]):
        A.append((x2[0,i]*x1[0,i], x2[0,i]*x1[1,i], x2[0,i],
                  x2[1,i]*x1[0,i], x2[1,i]*x1[1,i], x2[1,i],
                  x1[0,i],       x1[1,i],       1))
    A = np.array(A, dtype='float')
    #solve SVD
    U, S, V = np.linalg.svd(A)
    f = V[:, -1]
    F = f.reshape(3,3).T

    #make det(F) = 0
    U, D, V = np.linalg.svd(F)
    D[2] = 0
    S = np.diag(D)
    F = np.dot(np.dot(U, S), V)

    #De-normalize
    F = np.dot(np.dot(T2.T, F), T1)
    F = F/F[2,2]
    return F
```

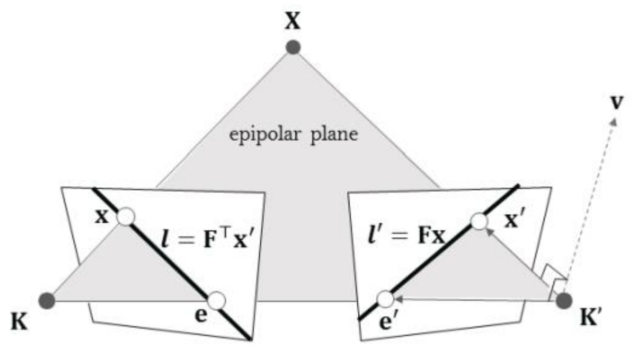
and calculate the error of F we found. And then keep the best result.



```
# get error of every pair for maybe_F
test_err = self.Cal8points_err(test_x1, test_x2, maybe_F)
#print(test_err.mean())
now_inlier = list(try_idxs)
for iter_err in range(len(test_err)):
    if test_err[iter_err] < self.thresh:
        now_inlier.append(test_idxs[iter_err])

if len(now_inlier) > len(max_inlier):
    Ans_F = maybe_F
    max_inlier = now_inlier
```

(3) draw the interest points on you found in step.1 in one image and the corresponding epipolar lines in another

By using fundamental matrix, the epipolar lines can be calculated as $l = F^T x'$ and $l' = Fx$. There are 3 parameters in l and l', which means ax + by + c = 0. We assign x0 and y0 first, and then calculate and x1 and y1. By the two points we can draw a line.



```
inliers_x = h_x[:, idx]
inliers_xp = h_xp[:, idx]

lines_on_img1 = np.dot(F.T, inliers_xp).T
lines_on_img2 = np.dot(F, inliers_x).T

#print(lines_on_img1)
#print(lines_on_img2)

draw(lines_on_img1, lines_on_img2, inliers_x, inliers_xp, img1, img2)
```

```
def draw(l, lp, x, xp, img1, img2):
    pic1 = cv2.cvtColor(img1, cv2.COLOR_GRAY2BGR)
    pic2 = cv2.cvtColor(img2, cv2.COLOR_GRAY2BGR)
    h, w = img1.shape

    for r, rp, pt_x, pt_xp in zip(l, lp, x.T, xp.T):
        color = tuple(np.random.randint(0, 255, 3).tolist())
        x0 = 0
        y0 = (-r[2]/r[1]).astype(np.int)
        x1 = w
        y1 = (-(r[2]+r[0]*w)/r[1]).astype(np.int)
        pic1 = cv2.line(pic1, (x0, y0), (x1, y1), color, 1)
        ''' 以下是同時畫右邊的線
        x0 = 0
        y0 = (-r[2]/r[1]).astype(np.int)
        x1 = w
        y1 = (-(r[2]+r[0]*w)/r[1]).astype(np.int)
        pic2 = cv2.line(pic2, (x0,y0), (x1,y1), color, 1)
        '''
        pic1 = cv2.circle(pic1, tuple((int(pt_x[0]), int(pt_x[1]))), 3, color, -1)
        pic2 = cv2.circle(pic2, tuple((int(pt_xp[0]), int(pt_xp[1]))), 3, color, -1)

    cv2.imwrite('./step3/left.png', np.concatenate((pic1, pic2), axis=1))
```

(4) get 4 possible solutions of essential matrix from fundamental matrix

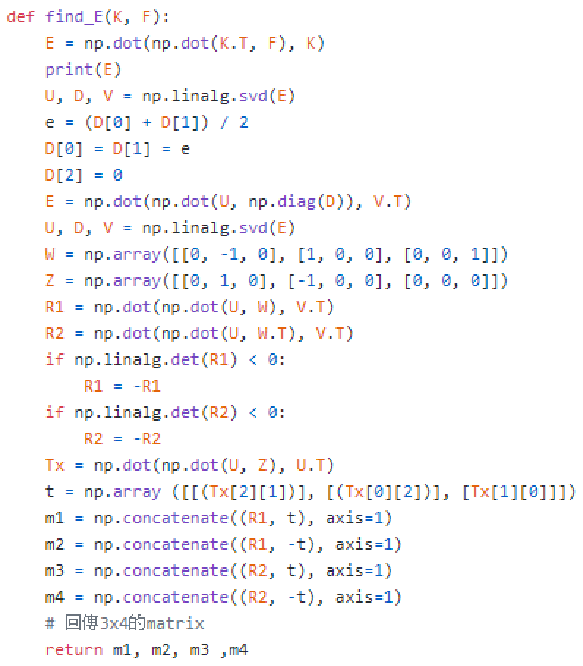
Assuming first camera matrix is P1 = [I | 0], there are four possible choices for the second camera matrix P2. First we use intrinsic matrix K to get $\mathbf{E} = \mathbf{K}'^{\top}\mathbf{F}\mathbf{K}$ and do SVD $E = \mathbf{U}\mathrm{diag}(1,1,0)\mathbf{V}^{\top}$.

The possible P2 will be

$$
\begin{aligned}
\mathbf{P}_2 &= [\mathbf{UWV}^\top \mid +\mathbf{u}_3] \\
\mathbf{P}_2 &= [\mathbf{UWV}^\top \mid -\mathbf{u}_3] \\
\mathbf{P}_2 &= [\mathbf{UW}^\top\mathbf{V}^\top \mid +\mathbf{u}_3] \\
\mathbf{P}_2 &= [\mathbf{UW}^\top\mathbf{V}^\top \mid -\mathbf{u}_3]
\end{aligned}
$$

W is

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



```python
def find_E(K, F):
    E = np.dot(np.dot(K.T, F), K)
    print(E)
    U, D, V = np.linalg.svd(E)
    e = (D[0] + D[1]) / 2
    D[0] = D[1] = e
    D[2] = 0
    E = np.dot(np.dot(U, np.diag(D)), V.T)
    U, D, V = np.linalg.svd(E)
    W = np.array([[0, -1, 0], [1, 0, 0], [0, 0, 1]])
    Z = np.array([[0, 1, 0], [-1, 0, 0], [0, 0, 0]])
    R1 = np.dot(np.dot(U, W), V.T)
    R2 = np.dot(np.dot(U, W.T), V.T)
    if np.linalg.det(R1) < 0:
        R1 = -R1
    if np.linalg.det(R2) < 0:
        R2 = -R2
    Tx = np.dot(np.dot(U, Z), U.T)
    t = np.array ([[(Tx[2][1])], [(Tx[0][2])], [Tx[1][0]]])
    m1 = np.concatenate((R1, t), axis=1)
    m2 = np.concatenate((R1, -t), axis=1)
    m3 = np.concatenate((R2, t), axis=1)
    m4 = np.concatenate((R2, -t), axis=1)
    # 回傳3x4的matrix
    return m1, m2, m3 ,m4
```

(5) apply triangulation to get 3D points & find out the most appropriate solution of essential matrix

With 4 possible camera matrix, we use triangulation to convert image points back to 3D coordinate.

$$x = \mathbf{P}X \quad x' = \mathbf{P}'X$$

$$\Downarrow$$

$$x = w\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \quad \mathbf{P} = \begin{bmatrix} \mathbf{p}_1^\top \\ \mathbf{p}_2^\top \\ \mathbf{p}_3^\top \end{bmatrix} \qquad x = w\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{P}X = \begin{bmatrix} \mathbf{p}_1^\top \\ \mathbf{p}_2^\top \\ \mathbf{p}_3^\top \end{bmatrix} X = \begin{bmatrix} \mathbf{p}_1^\top X \\ \mathbf{p}_2^\top X \\ \mathbf{p}_3^\top X \end{bmatrix}$$

$$x' = w\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} \quad \mathbf{P}' = \begin{bmatrix} \mathbf{p}'^\top_2 \\ \mathbf{p}'^\top_2 \\ \mathbf{p}'^\top_3 \end{bmatrix} \qquad w\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} u\mathbf{p}_3^\top X \\ v\mathbf{p}_3^\top X \\ \mathbf{p}_3^\top X \end{bmatrix} = \begin{bmatrix} \mathbf{p}_1^\top X \\ \mathbf{p}_2^\top X \\ \mathbf{p}_3^\top X \end{bmatrix}$$

$$\mathbf{A}X = 0 \quad A = \begin{bmatrix} u\mathbf{p}_3^\top - \mathbf{p}_1^\top \\ v\mathbf{p}_3^\top - \mathbf{p}_2^\top \\ u'\mathbf{p}'^\top_3 - \mathbf{p}'^\top_1 \\ v'\mathbf{p}'^\top_3 - \mathbf{p}'^\top_2 \end{bmatrix}$$

```python
def triangular(P2, x, xp):
    P1 = np.array([[1, 0, 0, 0],
                   [0, 1, 0, 0],
                   [0, 0, 1, 0]], dtype=float)

    p1 = P1[0, :]
    p2 = P1[1, :]
    p3 = P1[2, :]

    pp1 = P2[0, :]
    pp2 = P2[1, :]
    pp3 = P2[2, :]

    pointsX =[]
    for p, pp in zip(x, xp):
        u = p[0]
        v = p[1]
        up = pp[0]
        vp = pp[1]

        A = np.array([u*p3.T - p1.T,
                      v*p3.T - p2.T,
                      up*pp3.T - pp1.T,
                      vp*pp3.T - pp2.T])

        U, S, V = np.linalg.svd(A)
        X = V[:, -1]
        pointsX.append(X)
    pointsX = np.array(pointsX)

    for i in range(pointsX.shape[1]-1):
        pointsX[:,i] = pointsX[:,i] / pointsX[:,3]
    pointsX = pointsX[:,:3].T
    return(pointsX)
```

Next step is checking which solution will have maximum number of points in front of cameras.

The camera center is

- Camera Center

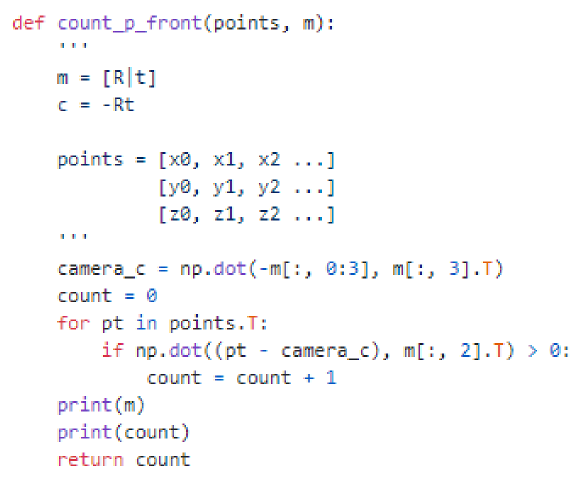
$$\begin{bmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \Leftrightarrow \mathbf{C} = \begin{bmatrix} X_{world} \\ Y_{world} \\ Z_{world} \end{bmatrix} = R^{\top} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} - R^{\top} t = -R^{\top} t$$

And the view direction is

- View Directopm

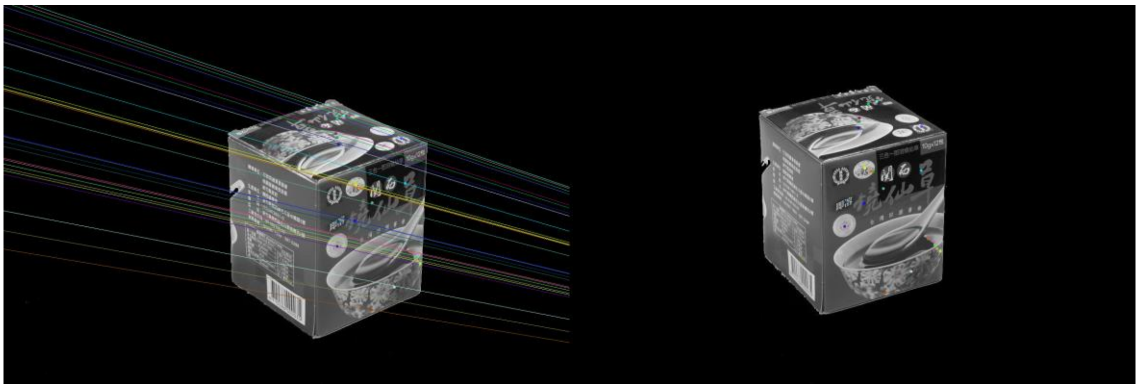
$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \Leftrightarrow \left(R^{\top} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} - R^{\top} t \right) - (\mathbf{C}) = \left(R(3,:)^{\top} - R^{\top} t \right) - \left(-R^{\top} t \right) = R(3,:)^{\top}$$

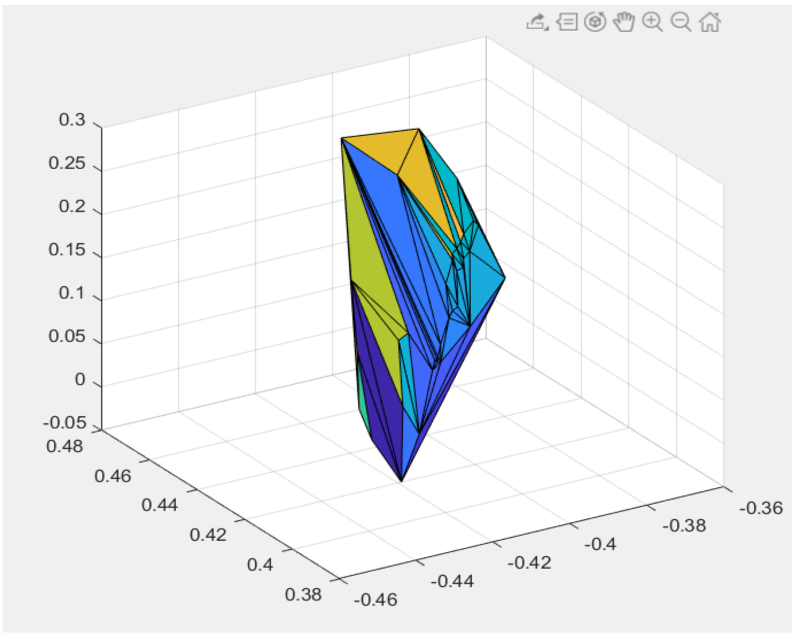
By performing this formula $(X - \mathbf{C}) \cdot R(3,:)^{\top} > 0$ we can know if the point is in front of camera. After doing the test of all points, we keep the best matrix to do next step.



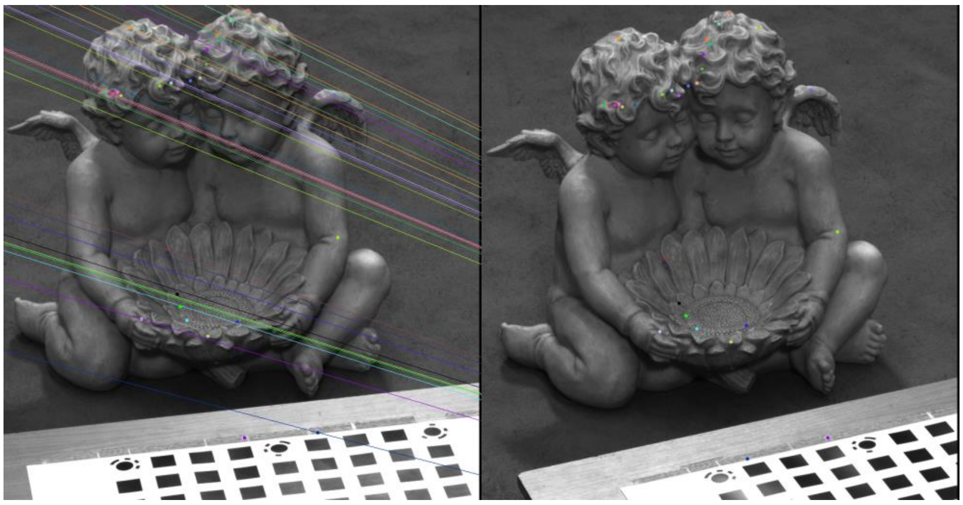
```
def count_p_front(points, m):
    '''
    m = [R|t]
    c = -Rt

    points = [x0, x1, x2 ...]
             [y0, y1, y2 ...]
             [z0, z1, z2 ...]
    '''
    camera_c = np.dot(-m[:, 0:3], m[:, 3].T)
    count = 0
    for pt in points.T:
        if np.dot((pt - camera_c), m[:, 2].T) > 0:
            count = count + 1
    print(m)
    print(count)
    return count
```

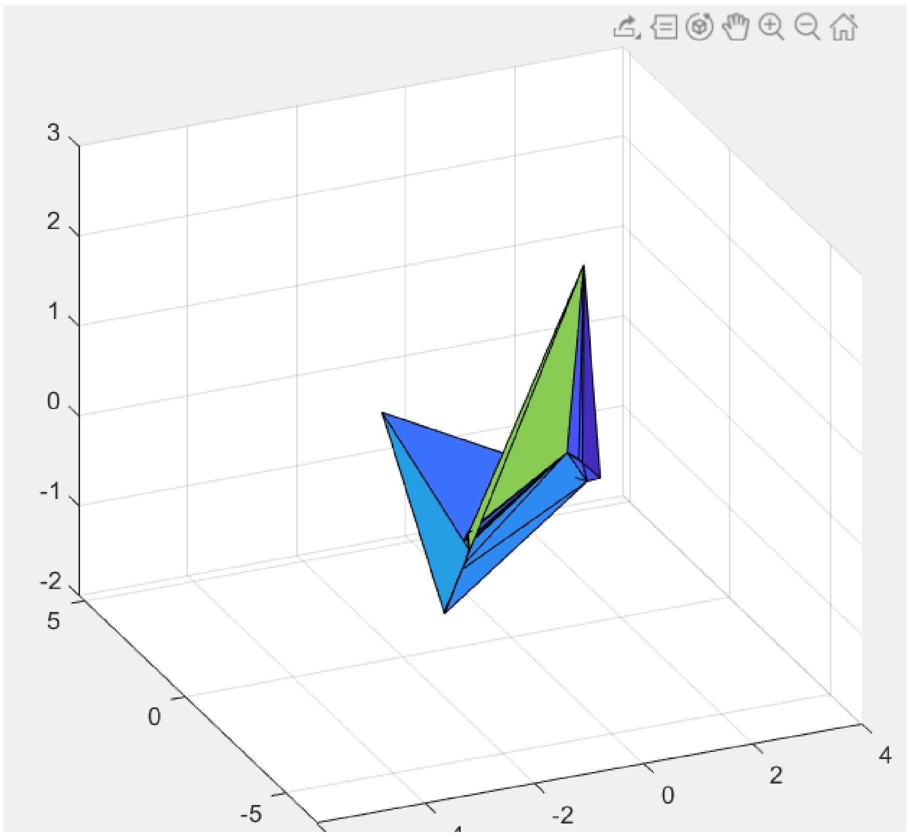
3. Experimental results (of course you should also try your own images)
    (1) Mesona



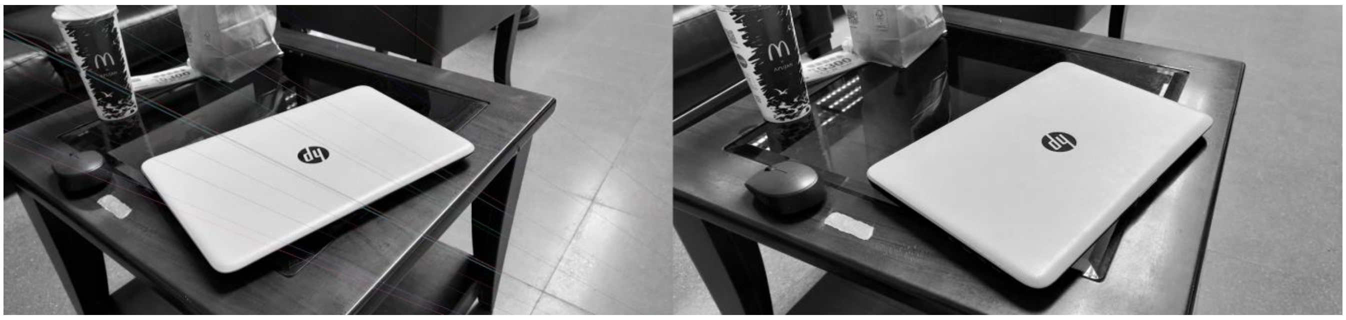


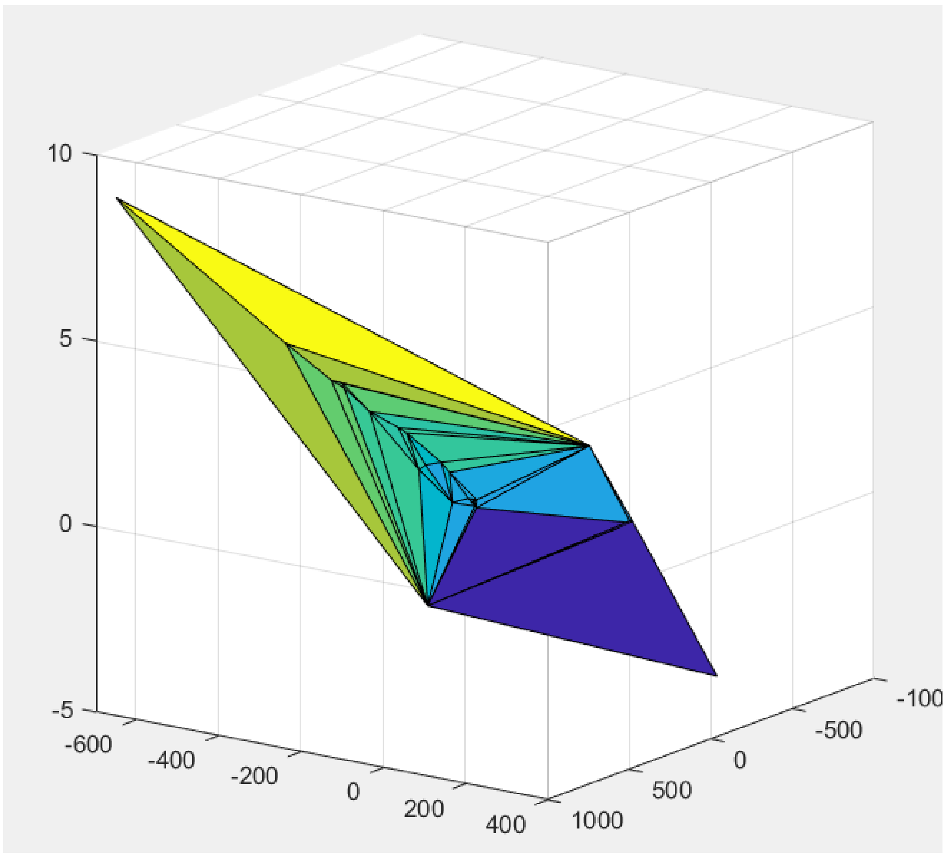
(2) Statue





(3) our picture





4. Discussion (what difficulties you have met? how you resolve them?)
    (1) When drawing the pictures (like step 3 and matlab), we didn’t know if our image is correct or not. So it is hard to debug or adjust our code.
    (2) There is an error in matlab code. We correct the file name in .mtl and the bug is fixed.
    (3) When doing matrix operations, some matrices need to be transposed first but some don’t need. The kind of problem confused us a little.
    (4) When calculating 8 points algorithm with RANSAC, we only take actually 8 points at first, but the outcome is not robust, so we increase the number of points we pick in one iteration.

5. Conclusion

    After finishing this homework, we know more about the translation of 2D and 3D coordinates. The theory about image transform can be used in many applications. Though there are a lot of packages about SfM we can use, the experience of this homework will help us a lot in the future.

6. Work assignment plan between team members.

    陳廷達:33%

    許芳瑀:33%

    呂淇安:33%