CV2024 HW4 SfM

Group10 - 313551073 顏琦恩, 313581027 葉彥谷, 313552001 黃梓濤

I. Introduction

Structure from Motion (SfM) is a technique used to reconstruct the 3D structure of a scene from a series of 2D images taken from different viewpoints. It simultaneously estimates the camera positions and orientations, as well as the three-dimensional positions of points in the scene. Our task is to implement the whole process with any camera calibration related functions, thus we utilized functions in OpenCV to implement the technique.

II. Implementation Procedure

Step 1. Finding Correspondence Across Images

In this section, we find the key points in images by SIFT method and match them with BFMatcher from OpenCV. The detailed code is shown in the following code block.

```
def extract_keypoint(img):
    sift = cv2.SIFT_create()
    keypoint, descriptors = sift.detectAndCompute(img, None)
    return keypoint, descriptors

def match_keypoint(descriptors1, descriptors2):
    bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)
    matches = bf.match(descriptors1, descriptors2)
    return sorted(matches, key=lambda x: x.distance)

## key points matching
key1, des1 = extract_keypoint(img1)
key2, des2 = extract_keypoint(img2)
matches = match_keypoint(des1, des2)
```

Step 2. Estimating the Fundamental Matrix

After finding the matching points between two images, we estimate the fundamental matrix by RANSAC and draw the epipolar lines. For the estimation, we further utilize the 8-point algorithm for experiment. We draw epipolar lines on both of the images. The detailed code is presented in the following code block.

```
def cal_fundamental_matrix(key1, key2, matches):
    pts1 = np.float32([key1[m.queryIdx].pt for m in matches])
    pts2 = np.float32([key2[m.trainIdx].pt for m in matches])
    F, mask = cv2.findFundamentalMat(pts1, pts2,
```

```
method=cv2.FM_RANSAC,ransacReprojThreshold=0.9, confidence=0.99) # mask is
selected by RANSAC
    # F, mask = cv2.findFundamentalMat(pts1, pts2, method=cv2.FM_8POINT)
    return F, mask, pts1, pts2

def draw_epilines(img, lines):
    r, c = img.shape[:2]
    img_color = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
    for line in lines:
        color = tuple(np.random.randint(0, 255, 3).tolist())
        x0, y0 = map(int, [0, -line[2] / line[1]])
        x1, y1 = map(int, [c, -(line[2] + line[0] * c) / line[1]])
        img_color = cv2.line(img_color, (x0, y0), (x1, y1), color, 1)
    return img_color

## estimate fundamental matrix
F, mask, pts1, pts2 = cal_fundamental_matrix(key1, key2, matches)

## calculate epipolar line
# epiline in left image
lines1 = cv2.computeCorrespondEpilines(pts2_inliers.reshape(-1, 1, 2), 2, F)
lines1 = lines1.reshape(-1, 3)
img1_epilines = draw_epilines(img1, lines1)
cv2.imwrite(f'{savedir}/epilines_image1.jpg', img1_epilines)

# epiline in right image
lines2 = cv2.computeCorrespondEpilines(pts1_inliers.reshape(-1, 1, 2), 1, F)
lines2 = lines2.reshape(-1, 3)
img2_epilines = draw_epilines(img2, lines2)
cv2.imwrite(f'{savedir}/epilines_image2.jpg', img2_epilines)
```

Step 3. Get possible solutions of Essential Matrix

The essential matrix is calculated using the formula $E = (K_2)^T F K_1$, where K_1 and K_2 are the intrinsic matrices of the cameras, and F is the fundamental matrix. The possible solutions of the essential matrix are then calculated through the Singular Value Decomposition (SVD) method; it extracts possible rotation matrices and translation vectors from E. The detailed code is shown in the following code block.

```
def E_to_pose(E):
    # eigenvalue correction : eigenvalue of E should have two and have same
value
    U, S, Vt = np.linalg.svd(E)
    m = (S[0] + S[1]) / 2
    E = U @ np.diag([m, m, 0]) @ Vt

    # calculate rotation matrix and translation(t) using svd
    U, _, Vt = np.linalg.svd(E)
    W = np.array([[0, -1, 0], [1, 0, 0], [0, 0, 1]]) # rotation matrix
```

```
    if np.linalg.det(U) < 0:
        U *= -1
    if np.linalg.det(Vt) < 0:
        Vt *= -1
    R1 = U @ W @ Vt
    R2 = U @ W.T @ Vt
    t = U[:, 2]
    return R1, R2, t

# calculate E
E = K2.T @ F @ K1
pts1_inliers = pts1[mask.ravel() == 1].reshape(-1, 2)
pts2_inliers = pts2[mask.ravel() == 1].reshape(-1, 2)

# Calculate R and t from E
R1, R2, t = E_to_pose(E)
```

Step 4. Finding the Most Appropriate Solution and Apply Triangulation

The function iterates over four possible poses derived from the rotation matrices and the translation vector, computing the projection matrices for both cameras. Using these projections, it triangulates 3D points from the input image correspondences. The 3D points are checked for validity by counting those with positive depth relative to the cameras. The pose with the highest number of valid points is selected as the best pose, and the corresponding 3D points are returned. The detailed code is shown in the following code block.

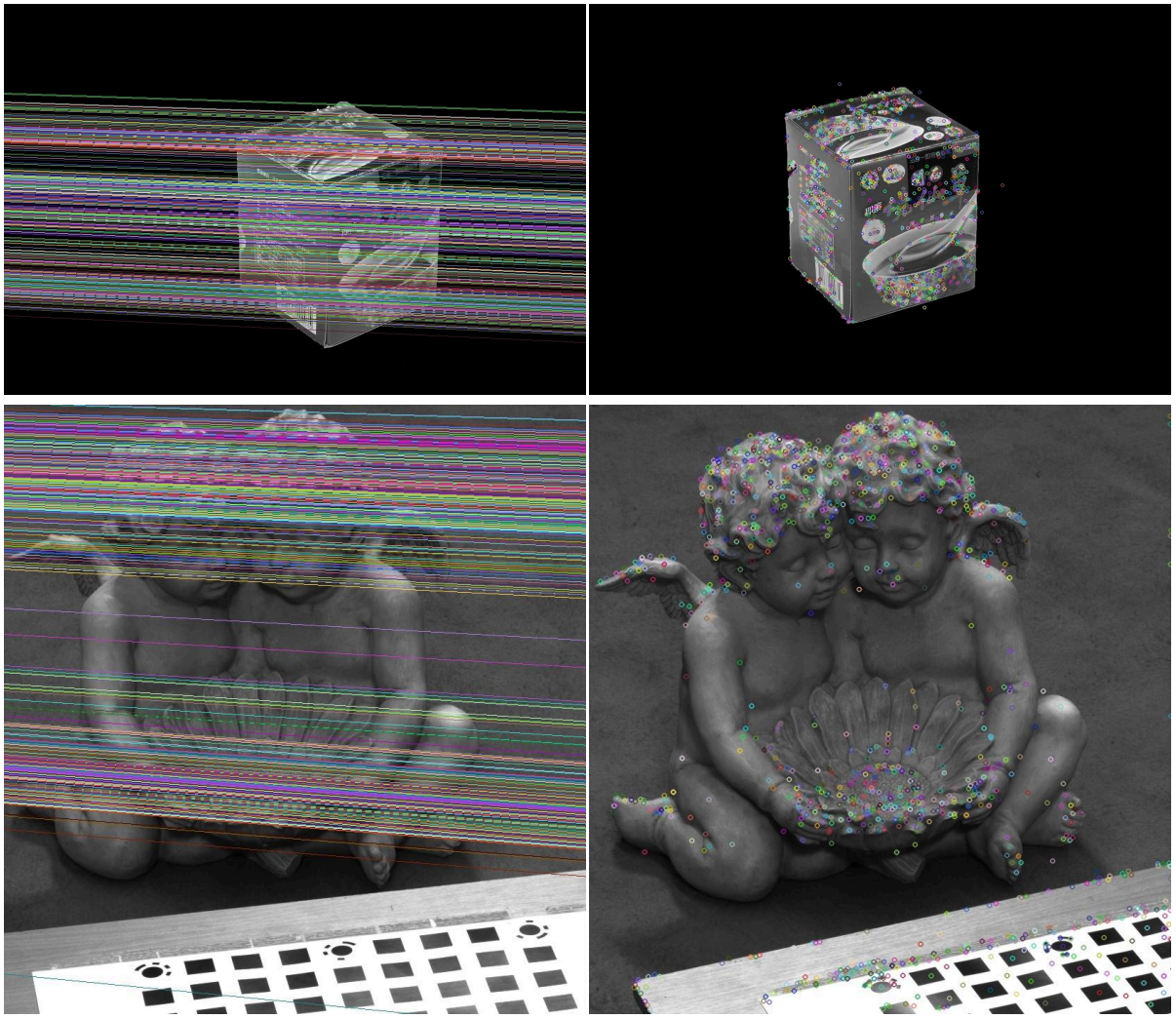
```
def find_best_rt_and_triangulate(R1, R2, t, pts1, pts2, K1, K2):
    possible_poses = [(R1, t), (R1, -t), (R2, t), (R2, -t)]
    best_pose = None
    max_points_in_front = 0
    best_points_3d = None

    for R, t in possible_poses:
        P1 = K1 @ np.hstack((np.eye(3), np.zeros((3, 1))))  # K1 [I|0]
        P2 = K2 @ np.hstack((R, t.reshape(3, 1)))           # K2 [R|t]
        pts1_h = cv2.convertPointsToHomogeneous(pts1).reshape(-1, 3).T  # 2D
point to homo
        pts2_h = cv2.convertPointsToHomogeneous(pts2).reshape(-1, 3).T  # 2D
point to homo

        tri_points_3D_h = cv2.triangulatePoints(P1, P2, pts1_h[:2], pts2_h[:2])
        tri_points_3D = tri_points_3D_h[:3] / tri_points_3D_h[3]

        points_in_front = np.sum(tri_points_3D[2] > 0)

        if points_in_front > max_points_in_front:
            max_points_in_front = points_in_front
            best_pose = (R, t)
```

```
            best_points_3d = tri_points_3D.T

    return best_pose, best_points_3d

# Triangulate
best_pose, points_3d = find_best_rt_and_triangulate(R1, R2, t, pts1_inliers,
pts2_inliers, K1, K2)
```

Experimental Results

We conducted the experiment with two different data setups, which are images from TA and images captured ourselves. The following figures present the feature matching and the final results of image stitching using data provided either by TA or by us. Note: The description of the output file is provided in Appendix I.

1. Results of TA's dataset



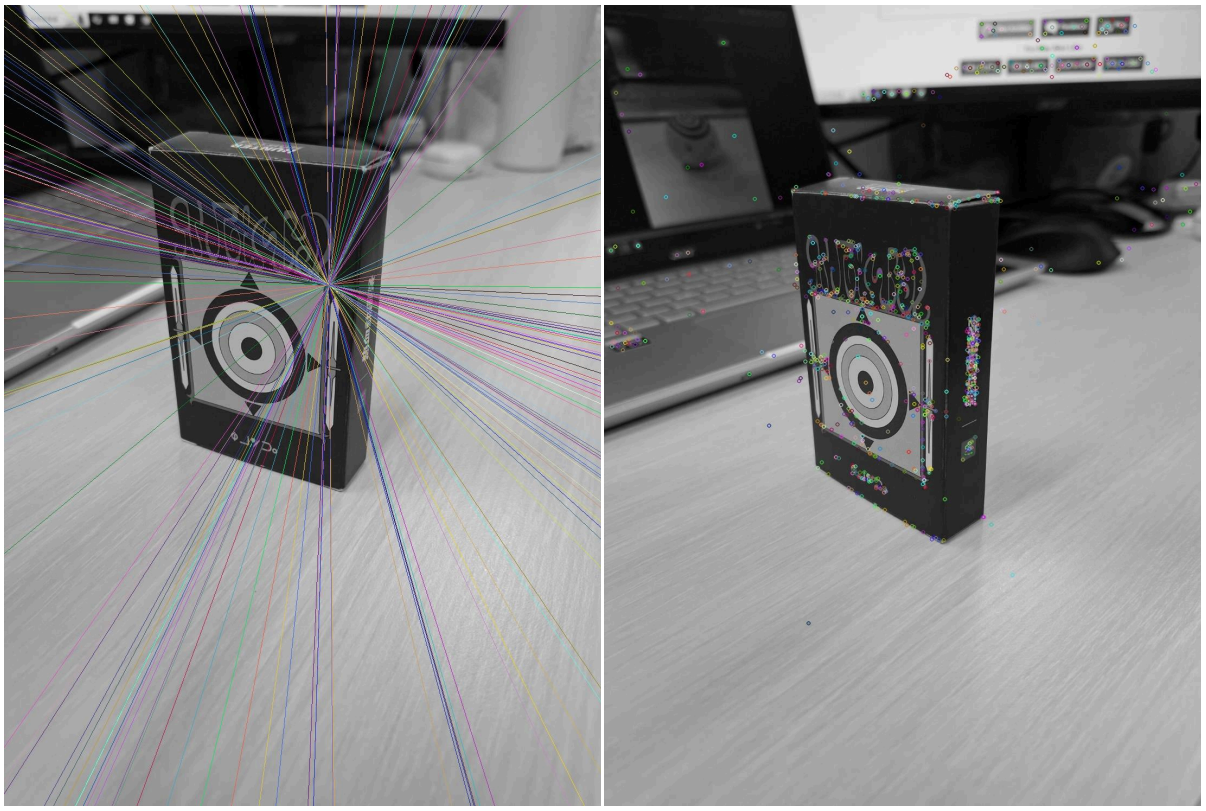
Epipolar lines Interest points

Figure. 1 Epipolar lines and interest points of images.

View 1 View 2 View 3

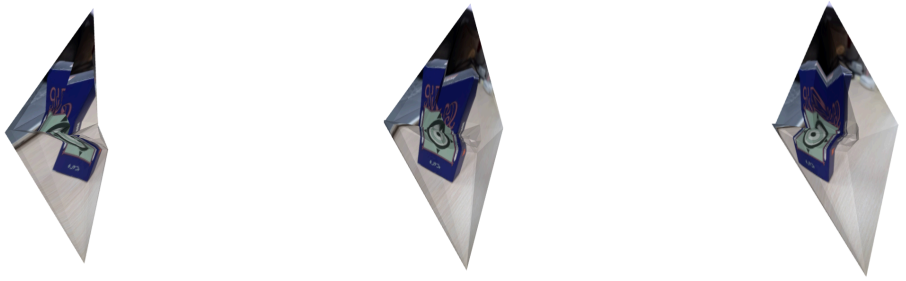
Figure. 2 Multiview of 3D meshes of images.

2. Results of our dataset



Epipolar lines Interest points

Figure. 3 Epipolar lines and interest points of images.



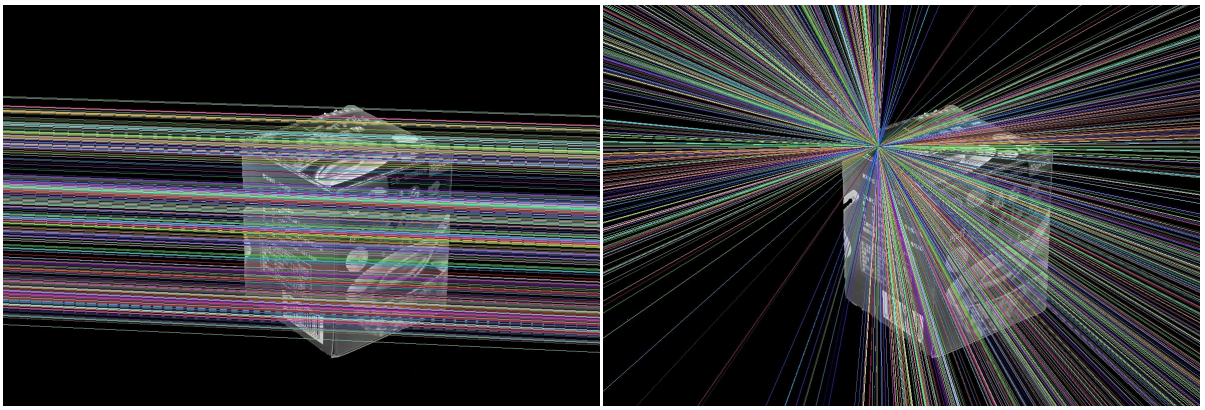
View 1 View 2 View 3

Figure. 4 Multiview of 3D meshes of images.

III. Discussion

1. Comparison of different fundamental matrix estimation method

We compare 2 different fundamental matrix estimation methods in this homework: RANSAC and 8-point algorithm. Figure 3 shows the epipolar lines drawn applying different fundamental matrix methods. We can see that the result of the 8-point algorithm is quite strange, this may be because the algorithm does not handle outliers in the point correspondence, leading to distortion in the estimation of the fundamental matrix.



RANSAC 8-Point

Figure. 3 Comparison of epipolar lines with different fundamental matrix estimation methods.

2. Difficulties we met

We met 2 main difficulties when doing the homework: (1) to implement the theory and (2) rendering a 3D model. For the first difficulty, we met the main problem when calculating the R and t matrix. Ideally, we won't have to do the eigenvalue correction, but when it comes to the real-world, the essential matrix may not have two same eigenvalues. This takes us some time to figure it out. As for rendering a 3D model, since we have no experience in this field, and it's also hard for us to understand the matlab code, we checked a lot of websites to overcome this problem.

IV. Conclusion

In conclusion, we've accomplished 3D model reconstruction from two different views of an object using the SfM technique.

V. Work Assignment Plan

- 葉彥谷 : Programming and report.
- 顏琦恩 : Programming and report.
- 黃梓濤 : Programming and report.

Appendix I : Filename Description

Path	Description
main.py	Main code
read_file.py	Python code used to read the calibration parameters
\my_data\	Our dataset
\output\[dataname]\	Output results of data named [dataname]
\output\[dataname]\8-point\	Epilines, interest points, and matched image results utilizing the 8-point algorithm
\output\[dataname]\ransac\	Epilines, interest points, and matched image results utilizing the RANSAC algorithm
\output\[dataname]\mesh\	Mesh of data
\output\[dataname]\point_cloud\	Point cloud of data
\output\[dataname]\multiview\	3D object multiview rendered