Computer Vision Programming Assignment 2

Structures from motion(SFM)

20195053 Dongjun Nam
june6423@gm.gist.ac.kr

Abstract

Structure from motion (SFM) is the process of estimating the 3-D structure of a scene from a set of 2-D images. We will look for fundamental step of SFM. To enable this, we will use SIFT (Scale Invariant Feature Transform) method from multi images to obtain their features. Then we will choose Image with the maximum number of key point matches. We will apply 3-point RANSAC (Random Sample Consensus) to get the fundamental matrix(camera pose). Based on the obtained camera pose, use the triangulation method to reconstruct 3D structures. Lastly, we will apply Bundle Adjustment to optimize obtained camera pose and 3d points.
Our code is available on https://github.com/june6423/PA2

1. Introduction

Structure from motion (SfM) is the process of estimating the 3-D structure of a scene from a set of 2-D images. SfM is used in many applications, such as 3-D scanning , augmented reality, and visual simultaneous localization and mapping (vSLAM). In this assignment, we will reconstruct the 3d-structure from multi images by following 3 steps and optimize it.

- Extract features from images using SIFT.
- Find the most reasonable matching features using k-nnMatch and implement RANSAC. While doing RANSAC, find camera pose [R|t] by using 3-point algorithm.
- Obtain 3D points using SVD (Singular Value Decomposition) from obtained camera pose and feature points. (Triangulation)
- Optimize 3d points and camera pose by minimizing reprojection error(Bundle Adjustment)

2. Methodology

2.1. SIFT and normalization

We used SIFT method to extract feature points from given images. To normalize the coordinates, We used the equation described below.

$$P_{norm} = K^{-1} P_{image} \tag{1}$$

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

Where:

- p_{image}: original image coordinate
- p_{norm}: normalized image coordinate
- K: Camera intrinsic
- f_x, f_y: camera focal length (x,y axis).
- c_x, c_y: camera center point (x,y axis).

2.2. Matching

We used cv2.BFMatcher() to select the most appropriate images with the most valid matches. cv2.BFMatcher() brute-forces matching between all keypoints. Since it is difficult to judge the suitability of a match in this way, add the k=2 option to the cv.BFMatcher.knnMatch() function to get the two closest matches. After that, a valid matching is determined by a ratio test. A ratio test is a method that determines a valid matching if the distance of the closest matching / the distance of the second closest matching is less than a threshold. In this experiment, we used threshold=0.95.

2.3. RANSAC

We used 3-point RANSAC to obtain camera pose. cv2.solveP3P() function returns at most 4 possible camera poses. Figure1 describes how 3-point algorithm obtains camera poses from three point matching. It's hard to check every possible 3 points matchings ($\binom{n}{3}$ is too large). Instead, RANSAC method randomly choose 3 points and check obtained camera pose is reasonable by checking number of points whose reprojection error is small. By iteratively choosing random points(In this experiment, we used 10,000 iterations), we can obtain a pretty good pose.
The output format of cv2.solveP3P() function is rotation vector (3*1 matrix) and translation vector(3*1). We used cv2.Rodrigues() function to transform rotation vector (3*1 matrix) to rotation matrix(3*3 matrix). Obtained Camera pose is $[R|t]$ (4*3 matrix).

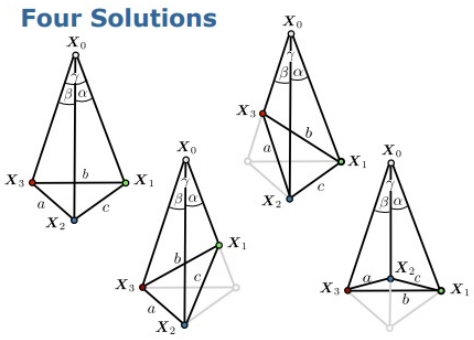

Figure 1: visualization of 3point algorithm

2.4. Reprojection Error

We checked the reprojection error to determine if the obtained camera pose and 3D points were appropriate. We can obtain coordinate of projection from formula described as below and normalized coordinate as described in Equation 1. We used threshold value as $5e-4$ to check our obtaining is appropriate.

$$P_{proj} = [R|t] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3}$$

$$|P_{proj} - P_{norm}| < threshold \tag{4}$$

Where:

- p_{proj}: projection of 3d points to normalized image plane
- p_{norm}: normalized image coordinate
- $[R|t]$: camera pose matrix.
- X, Y, Z: coordinate of 3d points.

2.5. Triangulation

In this part, we will reconstruct 3D coordinates from 2D pixel images by following equations.

$$P_1 = [R_1|t_1] = \begin{bmatrix} p^{1t} \\ p^{2t} \\ p^{3t} \end{bmatrix} \tag{5}$$

$$P_2 = [R_2|t_2] = \begin{bmatrix} p'^{1t} \\ p'^{2t} \\ p'^{3t} \end{bmatrix} \tag{6}$$

$$AX = \begin{bmatrix} x(p^{3t}) - p^{1t} \\ y(p^{3t}) - p^{2t} \\ x'(p'^{3t}) - p'^{1t} \\ y'(p'^{3t}) - p'^{2t} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = 0 \tag{7}$$

where

- p_i: camera pose of i-th image
- x, y: coordinate of key points in first image
- x', y': coordinate of key points in second image
- X, Y, Z: coordinate of reconstructed 3d points.

We can solve the equation by obtaining eigenvectors of matrix A. And we can obtain eigenvectors by performing singular value decomposition (SVD) as follows.

$$A = UDV^* \tag{8}$$

where

- D: diagonal matrix with diagonal element is eigenvalue of A
- V^*: column vector of V^* is eigenvector of A corresponding to its diagonal element

We can obtain X,Y,Z coordinate by dividing column of V^* from its last elements.

2.6. Bundle Adjustment

From step 2.1 to 2.4, we obtained camera pose of image and reconstructed 3d points. We can optimize coordinate of reconstructed 3d points and camera pose to minimize the sum of reprojection error. We used Leverberg-Marquardt method for optimization.

Leverberg-Marquardt method is similar with the gradient descent (GD) method. It calculates gradient (Jacobian in multivariable) and update parameters to gradient direction. The Levenberg-Marquardt method is a combination of the Gauss-Newton method and the gradient descent method, which works as a gradient descent method when far from the solution and as a Gauss-Newton method when near the solution. However, the Levenberg-Marquardt method can find the solution more reliably than the Gauss-Newton method (it has a higher probability of finding the solution even when the initial value is far from the solution) and converges to the solution relatively quickly.

Update rules for Leverberg-Marquardt is explained in equation 9.

$$[J^T J + \lambda diag(J^T J)]\delta = J^T F \tag{9}$$

where

- J: Jacobian matrix
- F: reprojection error (loss function)

- λ: damping parameter. When λ is large, LM method approaches Gradient Descent. When λ is small, LM method approaches Gauss-Newton method.
- δ: Update parameter. parameter X will update as $X + \delta$

3. Result

3.1. Matching results

With images 3 and 4 already matched, the next most matched image was image 2. We used threshold=0.95 for ratio test and result is shown in Fig.2.

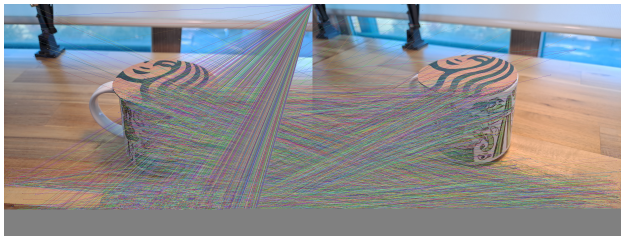

Figure 2: Matching result. left : image2, right : image3

3.2. Triangulation results

The result of performing triangulation on all images from image 0 to image 14 can be seen in Fig.3, where each yellow dot is a reconstructed 3D point.

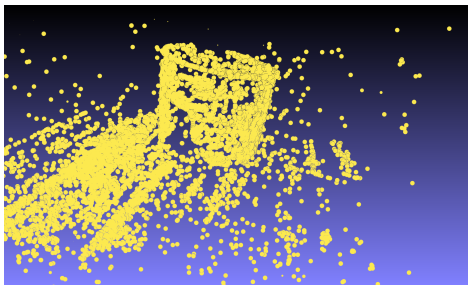

Figure 3: The result of triangulation for every given image

3.3. Bundle Adjustment results

In this part, we will discuss about result of bundle adjustment. Since we can get good results with the previous method alone, it is difficult to check the performance of the bundle adjustment. Therefore, we created a noisy image by giving a gaussian error of mean 0 and variance 0.5 to the previously obtained 3d points, and performed bundle adjustment. Our result is shown in Fig.4. and Fig.5. Fig.4. is the noisy image we created and Fig.5. is the result of the bundle adjustment.

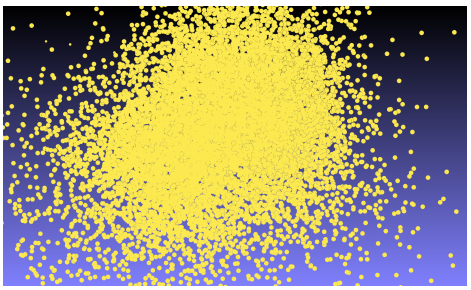

Figure 4: The result of 3d point with noise

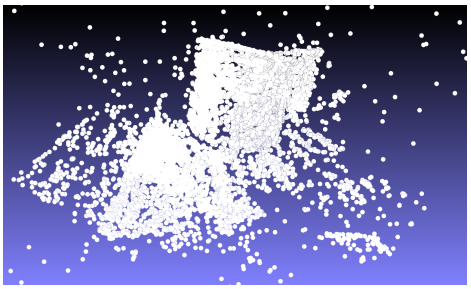

Figure 5: The result of the bundle adjustment

3.4. Discussion

From the results in 3.1, we can see that the keypoint at the bottom of image 2 is highly matched to the keypoint at the top left of image 3. This is because the feature in the upper left corner of image 3 is similar to the floor where the object is located, and it cannot be solved by adjusting the threshold value of the ratio test. Instead, it won't matter because it will be filtered out by the reprojection error. The idea is to use a high threshold to get as many keypoints as possible, and then filter out bad matchings through reprojection error.

The results in 3.2 showed that many points on the target object and the floor were reconstructed, and we were able to find some of the diverging points. The divergent points (noise) are likely due to the matching of very distant floor features in some of the images. Also, the cylindrical shape of the cup was well reconstructed, but the cup handle was not. This is because the cylindrical part of the cup has a variety of patterns and many keypoints were matched, while the handle is unpatterned and smooth, so no keypoints were found.

4. Result from custom dataset

I made custom dataset using my phone camera with fixed focus. I printed calibration checkerboard and used given calibration toolbox to obtain camera intrinsic matrix K. From obtained K, I utilized it to reconstruct the 3D coordinates from my image.

4.1. Calibration results

I printed checkerboard (9 box in x axis, 7 box in y axis, 20mm square each) and take a photo of it. The checkerboard is located in the specific location where we will place the object that we want to reconstruct. The result of camera calibration is shown in Fig.6 and Table.1

Figure 6: checkerboard used in camera calibration

camera parameters	x axis	y axis
focal length	2883.88	2874.07
principle point	1616.23	1598.68

Table 1: Result of camera calibration.

4.2. results

Our target image and reconstructed 3d points are shown in Fig.7 and Fig.8

Figure 7: target object

4.3. Discussion

In Figure 6, we can see that the dots at the back of the calendar are not properly reconstructed. This is likely due to the fact that the front part of the calendar is flat, so the keypoints were not properly extracted, and the sides of the calendar are blank, so continuous matching could not be performed.

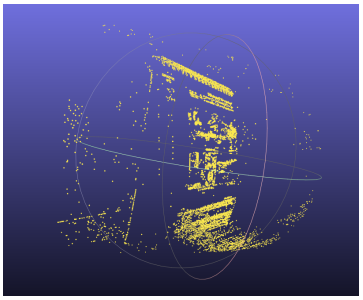

Figure 8: reconstruct 3d points on custom dataset

For successful SFM, it is necessary to select objects that have continuous keypoints.