CMSC733: Project 3 - Buildings built in minutes: An Sfm Approach

Khoi Viet Pham - Gnyana Teja Samudrala
Email: khoi@terpmail.umd.edu - sgteja@terpmail.umd.edu
Use 1 late day (submitted at 2AM).

I. Introduction

The project is about reconstructing a whole 3D scene from a set of images taken by a camera at different locations and poses. The problem here is often referred as Structure from Motion (SfM). In this report, we will breifly explain each step in the pipeline along with the problems we encounter. The whole pipeline is as follows:

1) Feature matching and outlier rejection using RANSAC and fundamental matrix.
2) Estimate fundamental matrix between two initial views.
3) Estimate essential matrix from the fundamental matrix.
4) Estimate camera pose from the above essential matrix.
5) Camera pose disambiguation to find the correct camera pose of the 2nd view.
6) Nonlinear triangulation to refine the values of the triangulated points.
7) Perspective-n-Point to register new image, new points, new camera pose into the system.
8) Bundle adjustment for final optimization of the whole reconstructed scene and camera poses.

II. Report

A. Feature matching & outlier rejection using RANSAC and fundamental matrix

The first step in the pipeline is to figure out the correspondences between every pair of images. Luckily, we are already provided with the matching data in the *Data* folder. Since these feature matchings were found by some feature descriptors, the data tends to be noisy and contains outliers. Therefore, we will remove the outliers using RANSAC along with the help of the fundamental matrix.

Given a pair of images along with their noisy correspondences, we use RANSAC with the 8-point algorithm to estimate the fundamental matrix between them. This works by randomply sampling 8 correspondences, then we estimate the fundamental matrix F, count the number of correspondences that satisfy the epipolar constraint (i.e. $x'^T F x \approx 0$ by using a small threshold value), then select the fundamental matrix that results in the largest number of inliners and reject the remaining correspondences. One note is that we have to enforce the computed fundamental matrix to be rank 2 per the project instruction.

Example of outlier rejection using RANSAC with the fundamental matrix is shown in figure 1.

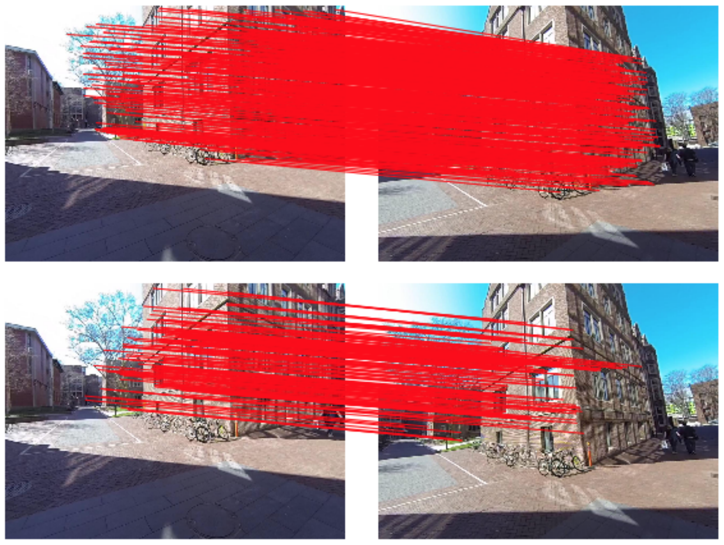
Fig. 1. Example of feature matching and outlier rejection using RANSAC with fundamental matrix: (above) before outlier rejection (below) after outlier rejection.

B. Estimate fundamental matrix between 2 initial views

We select image 1 and 2 as our two initial views for our scene reconstruction. From the previous step, we already got the fundamental matrix between these two views.

C. Estimate essential matrix from the fundamental matrix

Estimating the essential matrix is straightforward as we only need to use the formula $E = K^T F K$ where K is the camera intrinsic matrix. We also follow the trick per the instruction to correct E so that it has singular values $(1, 1, 0)$. We also make sure to test our computed essential matrix by using it for outlier rejection in step 1, which also produces similar resuts as in figure 1.

D. Estimate camera pose from the essential matrix

From the essential matrix, there are 4 possible solutions for the camera location and rotation (aka. camera pose) of the camera in image 2. Here, we already assume that the world coordinate system aligns with the coordinate system of the first camera, i.e. $C_1 = [0, 0, 0]^T, R_1 = I(3)$. Having 4 possible solutions is not bad because it's possible to use the Cheirality condition with 3D point triangulation to figure out the correct camera pose for image 2's camera in the next step.

E. Camera pose disambiguation to find correct camera pose for 2nd image

For each possible camera pose (C_2, R_2), we can use it with (C_1, R_1) (which aligns with the world coordinate system) to triangulate the 3D location of all correspondences between the two initial views. For each triangulation solutions X, we check if all triangulated points lie in front of both two cameras 1 and 2 by checking if $r_3(X - C_2) > 0$ (lie before camera 2) and $[0, 0, 1]^T X > 0$ (lie before camera 1). We present our initial triangulation plot with 4 camera poses in figure 1, and our correct triangulation plot after disambiguation in figure 2.

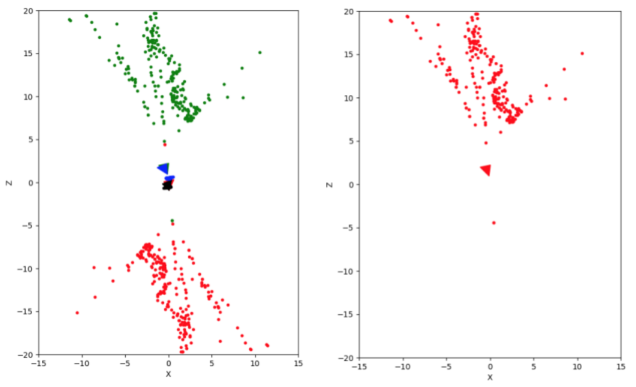

Fig. 2. (Left) Initial triangulation plot with 4 camera poses (Right) The correct triangulation plot after disambiguation. These plots are from top-view and are calculated from image pair 1 and 2.

F. Nonlinear triangulation to refine the triangulated points

Now that we got the camera pose for the 2nd camera as well as the linearly triangulated points X from the previous step, we can apply a nonlinear optimization function to refine X such that can minimize the reprojection error. We use the function *scipy.optimize.least_squares* to minimize the following reprojection error:

$$\min_x \sum_{j=1,2} \left(u^j - \frac{P_1^{jT}\tilde{X}}{P_3^{jT}\tilde{X}}\right)^2 + \left(v^j - \frac{P_2^{jT}\tilde{X}}{P_3^{jT}\tilde{X}}\right)^2$$

We present the results in figure 3.

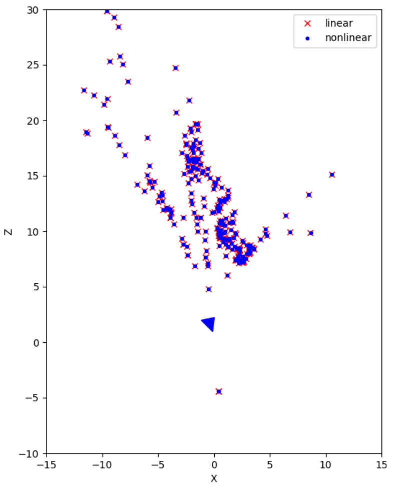

Fig. 3. Result after nonlinear triangulation: blue dots are results of nonlinear triangulation, red x's are results of linear triangulation. These plots are from top-view and are calculated from image pair 1 and 2.

G. Perspective-n-Points (PnP)

We have the world points in 3D(X) and their corresponding image points in 2D(x), using this correspondance we can calculate the rotation and translation of the camera in the world i.e 6 DOF pose of the camera. We first estimate the pose of the camera with linear least squares solution and make it robust by using RANSAC algorithm. This result from RANSAC is used as the initial estimate for the Non linear method. The steps are detailed below with plots for comparision and understanding the use of each method.

1) Linear PnP: The inputs to this are the image points(x), world points(X) and also the intrinsic parameters of the camera(K). At first the inverse of the intrinsic parameter matrix is calculated to normalise the image points. Then we get the equation 1 into the form $Ax = 0$ to solve it using linear least squares solution with SVD.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix} \begin{bmatrix} X \end{bmatrix} \tag{1}$$

where $P = R \begin{bmatrix} I_{3\times3} & -C \end{bmatrix}$

We manipulate this to arrive at,

$$\begin{bmatrix} 0_{1\times4} & -X^T & vX^T \\ X^T & 0_{1\times4} & -uX^T \\ -vX^T & uX^T & 0_{1\times4} \end{bmatrix} \begin{bmatrix} P_1^T \\ P_2^T \\ P_3^T \end{bmatrix} = 0$$

By doing SVD, the last column of V gives the solution to the equation. From this we extract the rotation matrix (R)and the 4th column will be the translation vector(T). The rotation matrix will not be orthogonal to make sure of which we decompose $R = USV^T$ and can get orthogonal R as $R = UV^T$. Also we check for the sign of the determinent and make it positive if not by negating both R and T. The centre is obtained by $C = -R^T T$.

TABLE I
TABLE OF REPROJECTION ERRORS AFTER CORRESPONDING STEPS

Pairs	LinTri	NonLinTri	LinPnP	NonLinPnP	BundleAdjustment
1_2	2.86549	2.82840	–	–	–
1_3	15.8999	15.57019	486181.167	43.2251	1.47299
1_4	30.4351	30.3962	200705.659	915.263	1.6456
3_5	6943.411	5346.143	17552064.40	31231.23922	11.5858
3_6	151.6227	141.8786	210636.399	7.4827	6.6519

2) PnP RANSAC: We use this to remove the outliers and get a better result of the initial linear estimate. This is run for a maximum of 1000 iterations and the threshold of the reprojection error for a point to be inlier is set to be 10. The formula used for calculating the reprojection error is

$$e = \left(u - \frac{P_1^T \tilde{X}}{P_3^T \tilde{X}}\right)^2 + \left(v - \frac{P_2^T \tilde{X}}{P_3^T \tilde{X}}\right)^2$$

3) Nonlinear PnP: We made use of the Nonlinear least squares solver from scipy to solve for optimal pose of the camera. The initial estimate for the solver is given from the values obtained by above method RANSAC PnP. In this to maintain the orthogonal condition of the rotation matrix we use quaternion as the input parameter for the solver. The equation to be minimized is

$$\min_{C,q} \sum_{i=1,J} \left(u^j - \frac{P_1^T \widetilde{X}_j}{P_3^T \widetilde{X}_j}\right)^2 + \left(v^j - \frac{P_2^{iT} \widetilde{X}_j}{P_3^{iT} X_j}\right)$$

H. Bundle Adjustment

We didnot use the visibility matrix for bundle adjustment, instead created a dictionary for each image which has tuple pairs with 2D image point and index to the corresponding 3D point. Most of the part and logic of implementation is adapted from [?] large scale bundle adjustment for scipy. In this the error we try to minimize is

$$\min_{\{C_i,q_i\}_{i=1}^{i},\{X\}_{j=1}^{J}} \sum_{i=1}^{I}\sum_{j=1}^{J} \left(\left(u^j - \frac{P_1^{jT} \tilde{X}}{P_3^{jT} \tilde{X}}\right)^2 + \left(v^j - \frac{P_2^{jT} \tilde{X}}{P_3^{jT} \tilde{X}}\right)^2\right)$$

The parameters to be optimized are given as a single 1D array. In this the first N($N = no.of cameras * 7$) variables are camera poses and the later are the 3D points. Each camera pose is represented as 7 variables which are rotation matrix in quaternion form (q_w, q_x, q_y, q_z) and centre position of camera (C_x, C_y, C_z). Two 1D arrays for all the possible observations are made of camera and point indices which maps the right 3D point with the camera pose. Also the 2D image points are ordered in the same fashion. And the norm between calculated and ground truth image points is minimized by the optimizer.

Following are our results after each step of bundle adjustment.

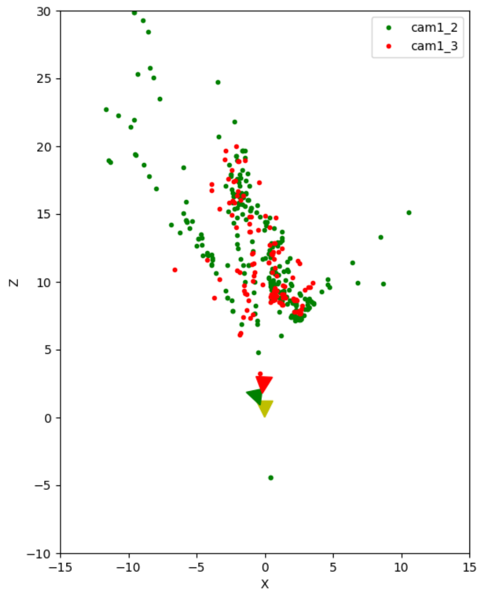

Fig. 4. Bundle adjustment result after adding pair (image 1, image 3). Camera 1, 2, 3 have color yellow, green, red respectively.

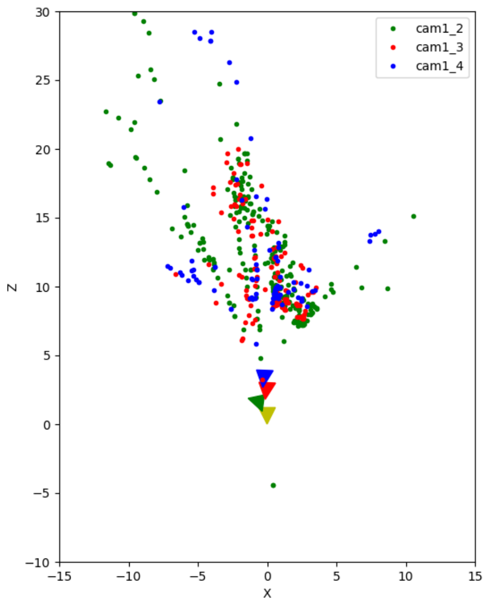

Fig. 5. Bundle adjustment result after adding pair (image 1, image 4). Camera 1, 2, 3, 4 have color yellow, green, red, blue respectively.

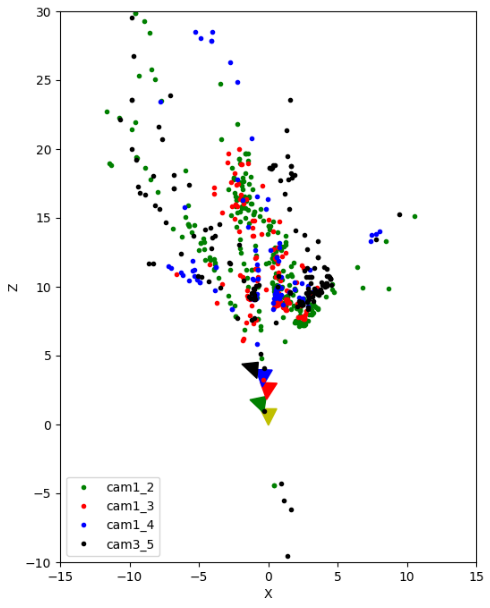

Fig. 6. Bundle adjustment result after adding pair (image 3, image 5). Camera 1, 2, 3, 4, 5 have color yellow, green, red, blue, black respectively.

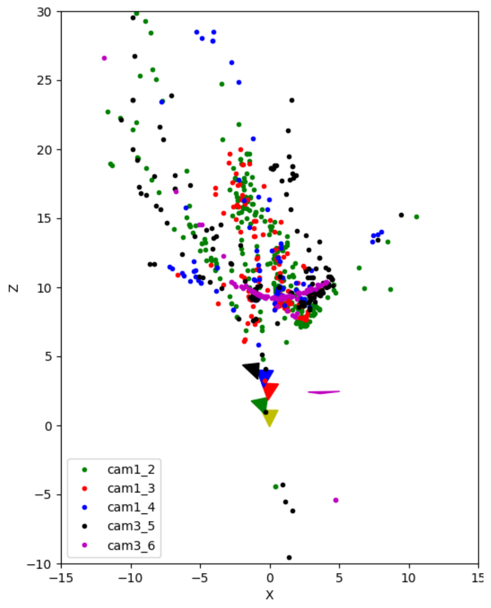

Fig. 7. Bundle adjustment result after adding pair (image 3, image 6). Camera 1, 2, 3, 4, 5, 6 have color yellow, green, red, blue, black, purple respectively. Notice that camera 6 is weirdly plotted, this is because the reprojection error after nonlinear PnP is too high and our estimated pose for camera 6 is not good enough.

I. VSFM

Following are the results we got from VSFM (figure 8 and figure 9). We remove points that are too far away for better visualization.

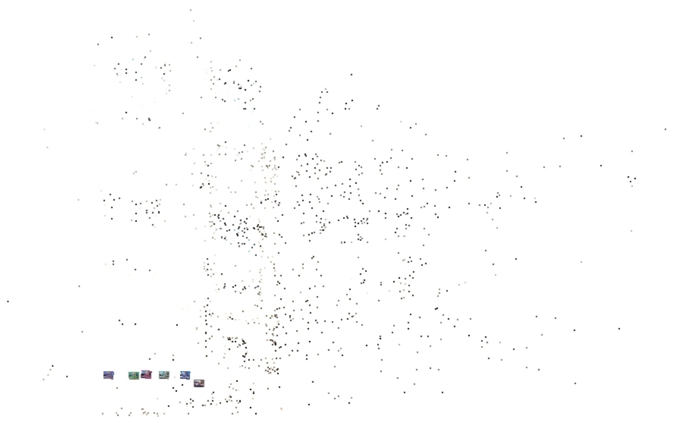

Fig. 8. VSFM: before bundle adjustment.

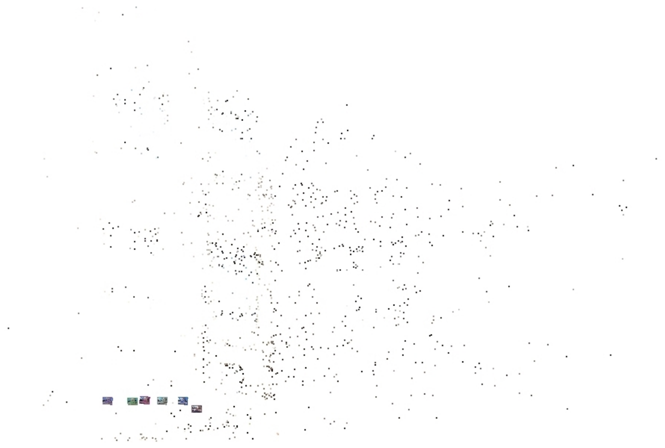

Fig. 9. VSFM: after bundle adjustment.