ETH ZURICH, D-INFK

Computer Vision Exercise 6: Structure from Motion

Xingchang Huang

December 3, 2017

1 FEATURE EXTRACTION AND INITIALIZATION WITH EPIPOLAR GEOMETRY

Here we use SIFT again to extract features in every image. We start with extracting features on the first and the last image that have large baseline but still we can find matched features from them. As they are the most different views among these 5 pictures, we use SIFT's vl_sift in MATLAB with peak threshold that filters peaks of the DoG scale space that are too small, which means I try to find more exact inliers for them later. Then, I use the matching feature points to determine the Fundamental matrix using 8-point RANSAC algorithm. After this step, I can get the inliers and the Fundamental matrix. The result for the feature extraction, matching and inliers matching are shown below:

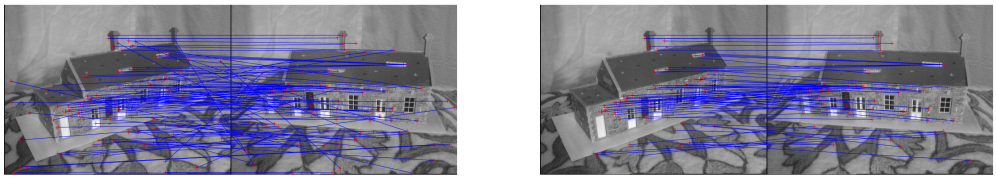

Figure 1.1: Feature Extraction, Matching and Inliers Matching.

As we want to get the projection matrix (R and t), we first need to transform Fundamental

matrix to Essential matrix using equation $E = K' * F * K$ and then decompose Essential matrix into Rotation and Translation matrix. As a result, we know the right projection matrix for and therefore we know the camera position of the second view. And for the first view, we just initialize its position as a basis [I|0]. After knowing both the projection matrix for two views, and 2D points in the images, we can now use linear triangulation to find the corresponding 3D points in the space. The result will be shown in the last section.

For this task, we should also draw the epipolar lines (geometry) for both images.

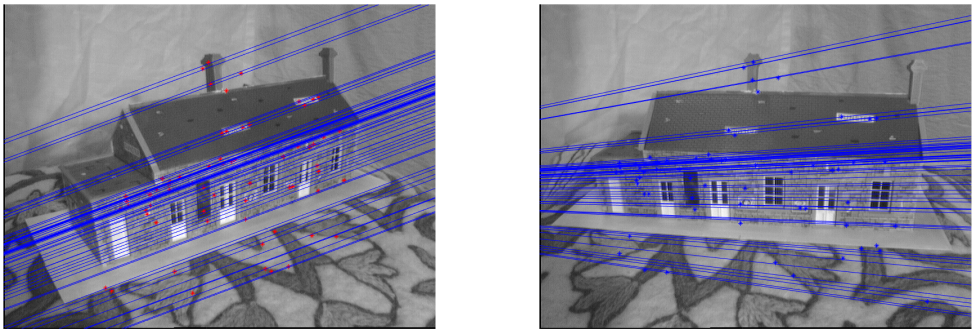

Figure 1.2: Initialization with Epipolar Geometry for the first and last views.

2 Triangulation and adding new views

Actually, we can get those 3D points using two views of the object. But with more views, we can somehow get more accurate reconstruction result and if we use more views as basis like image0 in this tasks, we can reconstruct the whole house from different views. Here, we just need to add more views based on image0. As we have known the 3D points, we don't need to compute the Fundamental matrix or Essential matrix again. Instead, we use correspondence between 2D points extracted by SIFT and matched with inliers in image0, and 3D points by triangulation of previous step, to compute the projection matrix. Notice that we should first calibrated the points with K^{-1} because the function projmatrix is to calculate the R and t using RANSAC DLT. After getting the projection matrix using 2D-3D correspondence, we can also get 3d points from the new view, which may has a bit error compared with the reconstructed 3D points in the previous section. For more new views, we have the same methods and process, but should pay attention to the index carefully for implementation.

Here I show the feature matching between the first view and other new views from Figure 2.1-2.3. As shown in the figures, the inliers are less than that of image 1 and 4 because those inliers are extracted from the inliers of image 1 and 4, which means that it is normal that we will get less inliers while getting additional view.

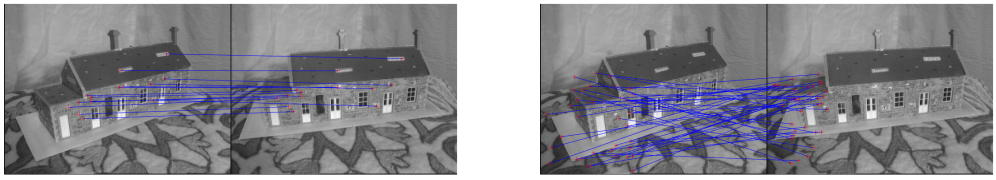

Figure 2.1: Inlier and Outlier Matching using 6-point RANSAC: images 0 and 2.

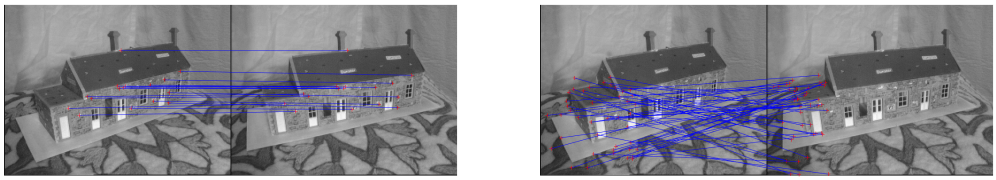

Figure 2.2: Inlier and Outlier Matching using 6-point RANSAC: images 0 and 1.

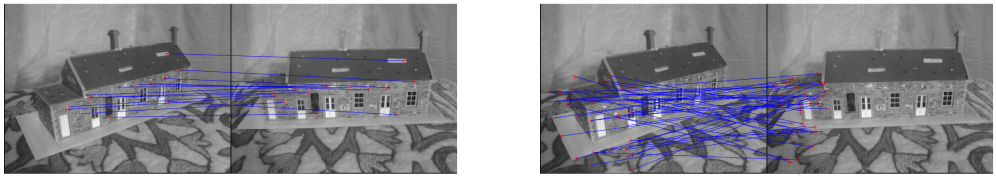

Figure 2.3: Inlier and Outlier Matching using 6-point RANSAC: images 0 and 3.

3 PLOTTING

To analyze my result, I will show those triangulated inliers with cameras in 3D space to see whether it is correct. Figure 3.1 shows good results that all the points are in front of the cameras (blue arrows). Green, Red, Blue, Black shows the triangulated inliers using 4 views respectively. Here we have more green points as we will get less inliers for adding new views compared with the initialization. And those points are close but not completely overlap because of errors. For plotting with dots in Figure 3.1, we can clearly found that those points are totally not overlap and thus have obvious error for some points. And I think the problem lies in the choice of threshold while computing the projection matrix.

And here I use one more trick to ensure that the camera points to the positive direction by check the determinant of Rotation matrix. If it is negative, just add a minus symbol to let it positive, which do not have any influence for the result and correct the problem for negative term in Rotation matrix.

Finally, it should be noticed that I have done multiple runs to get this result because some-

times one or two triangulated points will be in the back of the cameras, which may result from wrong inliers matching and this can be fixed by running multiple times in my implementation.

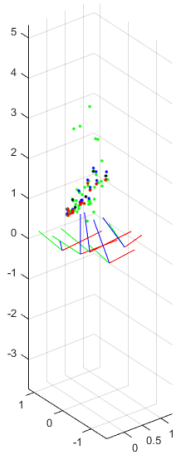

Figure 3.1: Triangulated points and camera poses plotted in 3D using dots.