Computer Vision Assignment 2

Yating Jing

October 16, 2015

1 Feature Detection

Note that for different sets of images, above parameters are chosen differently in order to achieve the best effect.

Parameters Take image set bikes as an example here.

In the Harris corner detection, the Gaussian window size is set to be 7 by 7.

For the calculation of the R map, k is set to be 0.5.

For the nonmaxsuppts parameters, nonmax_radius is set to 5 and corner_thresh is $mean(R) * 1.2$. R map is normalized to $[0.0, 10.0]$ before thresholding.

The R map for bikes1.png looks like:

Figure 1: R map for bikes1

2 Feature Matching

To compare two features, normalized cross-correlation (NCC) is calculated between image patches around two features. The matching results:

Figure 2: Matches between bikes1 and bikes2

Figure 3: Matches between bikes1 and bikes3

Figure 4: Matches between graf1 and graf2

Figure 5: Matches between graf1 and graf3

Figure 6: Matches between leuven1 and leuven2

Figure 7: Matches between leuven1 and leuven3

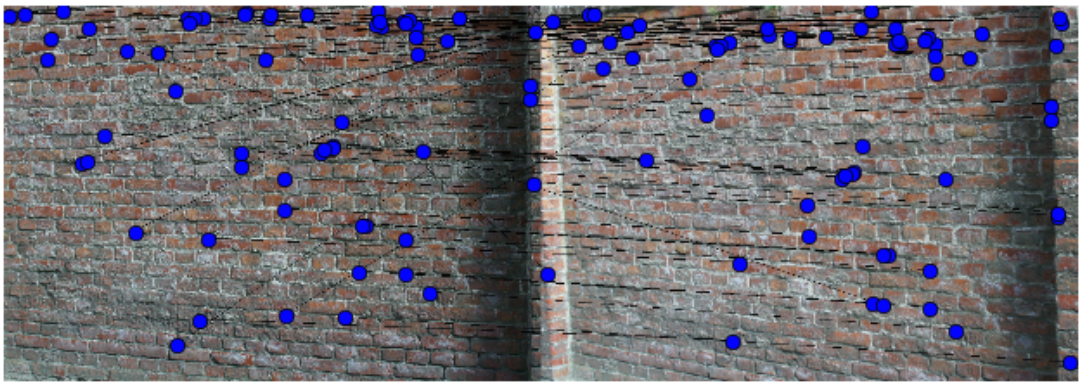

Figure 8: Matches between wall1 and wall2

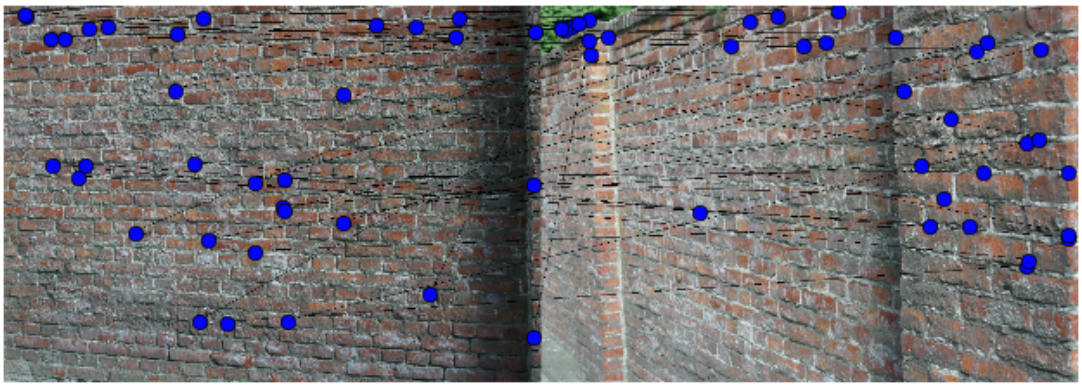

Figure 9: Matches between wall1 and wall3

3 Alignment and Stitching

Note: please close the matching display and wait(for a while) for the alignment display.

3.1 Ransac Algorithm

Number of samples randomly chosen in each round: 3.

Inlier Counting After computing the transformation matrix using randomly chosen 3 matches, for all the other matches, assume feature coordinate (x_1, y_1) is from image1 and its matching coordinate (x_2, y_2) is from image2. Now we want the transformation successfully transform (x_1, y_1) to (x_2, y_2).

Transform (x_1, y_1) to (x, y) using the transformation matrix, compute two errors $e_x = (x - x_2)^2$ and $e_y = (y - y_2)^2$. Rather than just calculate the scalar error sum $\sum(e_x + e_y)$ of all the matches, use the following method:

Use all the e_x as the x-coordinates and corresponding e_y as the y-coordinates, fit a line

$e_y = ae_x + b$. For each coordinate (x_j, y_j) from image1 in the matches, transform it and calculate the errors (e_{x_j}, e_{y_j}). Then compute the distance from (e_{x_j}, e_{y_j}) to the line and threshold this distance. The error threshold thus defines a margin around the line. Only those whose error distances to this line are within this margin($\leq$ error threshold) can be counted as inliers. Finally, pick the transformation matrix that gives the maximum number of inliers.

For different image sets, different Ransac rounds and error thresholds are chosen. For example, bikes1 to bikes2 transformation uses 20000 rounds and an error threshold of 30.

Zero Padding Zeros are padded to the image borders before the alignment and stitching. Different sets of images use different padding widths.

3.2 Affine Transformation Display and Analysis

As shown below, most of the alignments are tolerable. However, transformations of graf1→graf3 and wall image sets are very inaccurate. These are later improved with Simple-SIFT features in next section.

The reason for the poor matches and alignment might be due to the following reasons: the graf and wall images are more complicated and less distinguishable compared to the bikes and leuven. In graf and wall there exists many similar small patches, whereas in bikes and leuven, the objects are very well separated and distinct. Moreover, in the graf1→graf2 and wall1→wall3, the geometric perspective transformations are way more complex and larger different areas are present. In particular, the portion of the wall captured by wall1 is only half of that captured by wall3, which makes the transformation much harder.

Due to the time limit, I did not have enough time to carefully tune all the parameters in different procedures. Further experiments with the parameters should result in better alignment, especially for graf1→graf2 and wall1→wall2.

Figure 10: bikes1 and bikes2

Figure 11: bikes1 and bikes3

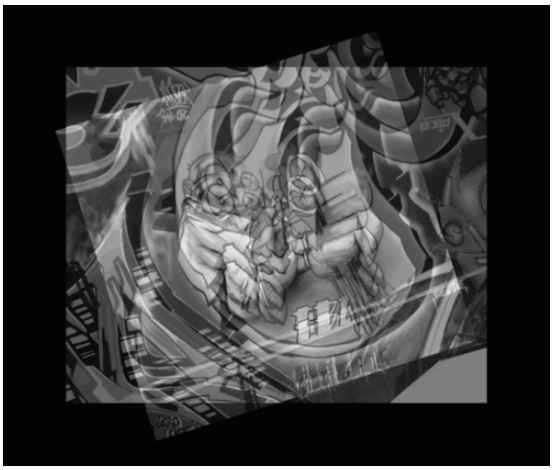

Figure 12: graf1 and graf2

Figure 13: leuven1 and leuven2

Figure 14: leuven1 and leuven3

3.3 Fully-Projective Transformation

Number of samples randomly chosen for each round: 5.

Although this model is more flexible with more degrees of freedom, since NCC cannot guarantee good matching, the affine model is preferred. In affine transformation, there are only six unknowns, so at least 3 matches are needed. Whereas in the fully-projective model that has 8 degrees of freedom, at least 4 matches are needed. While we might have trouble coming up with enough good matches, sometimes the latter might yield inaccurate transformation.

Another issue is that more samples means lower probability of choosing only good matches, since in the first step of Ransac, the samples are selected randomly. As a result,

the affine model has a better chance of ending up with the best set of matches.

The affine transformation seems more stable than the fully-projective model, but when they both yield acceptable transformations, the results do not differ much. Here the warping results from bikes1, bikes2 and bikes3 are displayed for comparison:

Figure 15: bikes1 and bikes2 (using Fully-Projective Transformation)

Figure 16: bikes1 and bikes3 (using Fully-Projective Transformation)

4 Feature Description

A script **match_ssifts.py** that matches Simple-SIFT features using ratio test is added. Its return is just like the match_features.py (matching coordinates). The thresholds used in the test are different for different images. For most sets 0.6 suffices, while for graf image set, it should be at least 0.85.

In general, ssift features lead to better matching. But even for wall1→wall3 and graf1→graf3, both the matching and the transformation fail to be acceptable, due to the significant geometric difference in these images.

The result of graf1→graf3 is not displayed here since it is completely off.

Figure 17: Matches between graf1 and graf2 (using Simple-SITF)

Figure 18: Matches between graf1 and graf3 (using Simple-SITF)

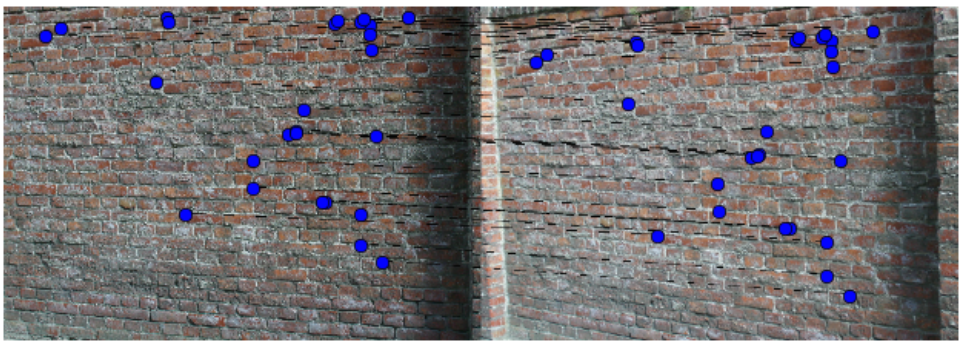

Figure 19: Matches between wall1 and wall2 (using Simple-SITF)

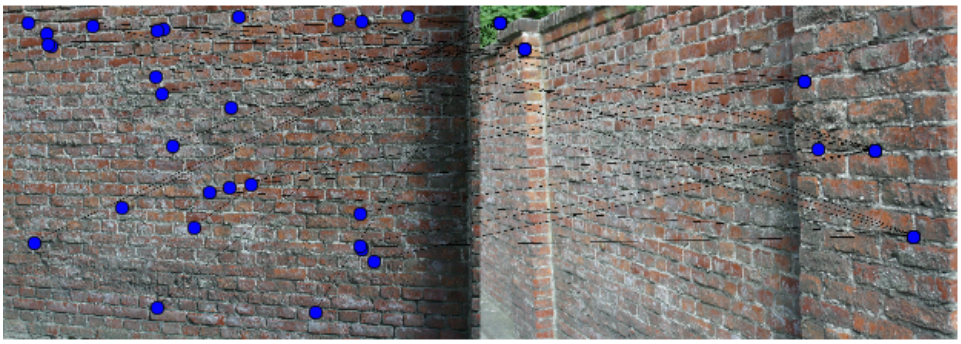

Figure 20: Matches between wall1 and wall3 (using Simple-SITF)

Figure 21: bikes1 and bikes2 (using Simple-SITF)

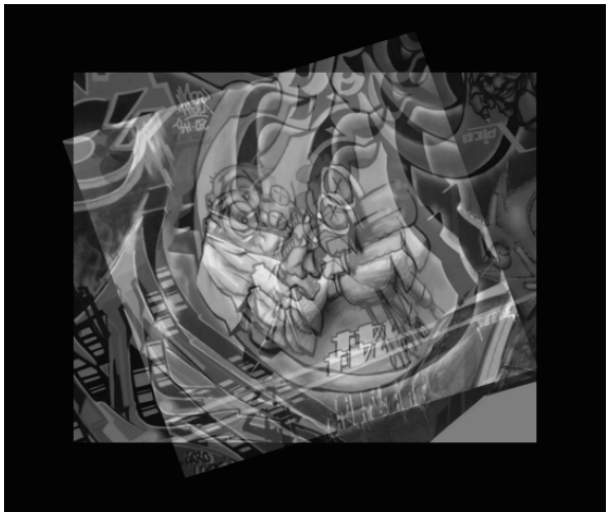

Figure 22: graf1 and graf2 (using Simple-SITF)

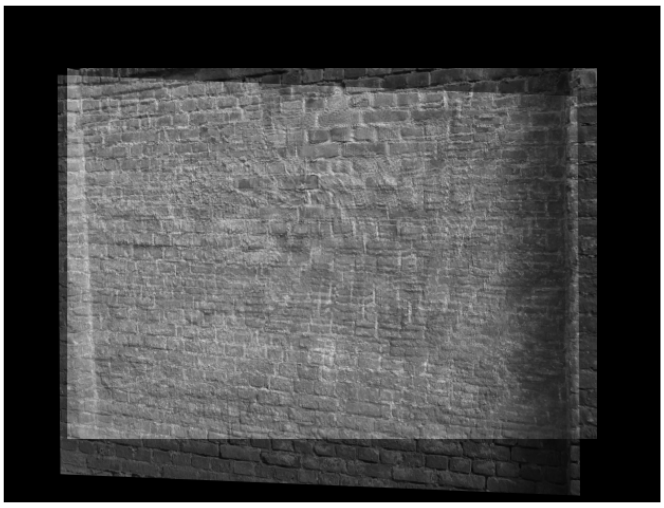

Figure 23: wall1 and wall2 (using Simple-SITF)

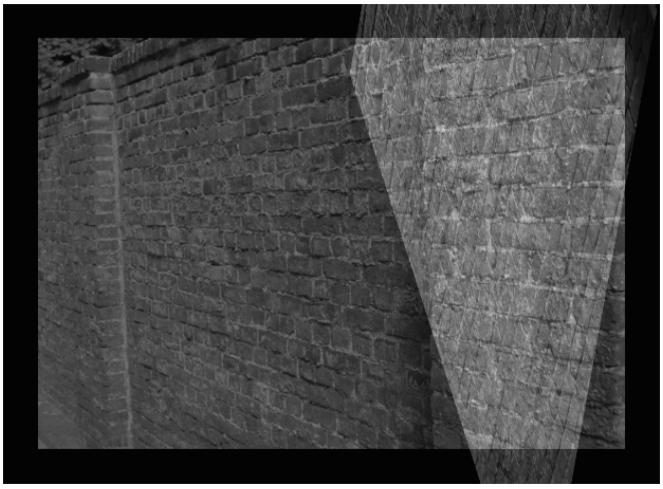

Figure 24: wall1 and wall3 (using Simple-SITF)