Hengyue Liu
4107-2966-75
hengyuel@usc.edu
CSCI574 Homework #3
October 20, 2015

# Homework #3 Report

## Ideas and steps:

This homework assignment is included 3 steps to locate an object in test images. The basic idea is:

Step 1: SIFT feature extraction

The SIFT features remain unchanged in different camera views, so it can be utilized for finding the corresponding matched key points in other views.

Step 2: Find matching pairs

Based on the feature vectors obtained in first step, calculate the Euclidean distance of each matched key points between original image and test image. However, some procedures need to be conduct for miss-match and one-to-many match. After these procedures, good match pairs can be derived. Note that the number of matched pairs will be less than key points found in the first step.

Step 3: Locate object

This is the most important step for detecting the object in different images. The coordinates of two images can be transformed by the homography transform matrix. To find the homography, only 4 matched pairs are needed. However, RANSAC algorithm is used to find the best homography model that can satisfy most points on the object.

As long as the homography matrix is found, project the key points on the object onto the test image, then show both the projected points and the realistic key points of the test image.

## Experiment results:

1. Image_1.jpg as the object image, image_3.jpg as the test image.

Feature matching pairs:


Object and overlay feature points:

Homography matrix obtained:

```
Homography matrix:
[0.5537462896618912, 0.09026521474690631, -191.8116447565114]
[-0.1124338183070337, 0.5169957383702024, 144.9441016317862]
[-4.755799453892252e-05, -5.571335451539714e-05, 0.9999999999999999]
Program ended with exit code: 0
```

2. Image_1.jpg as the object image, image_4.jpg as the test image.

Feature matching pairs:

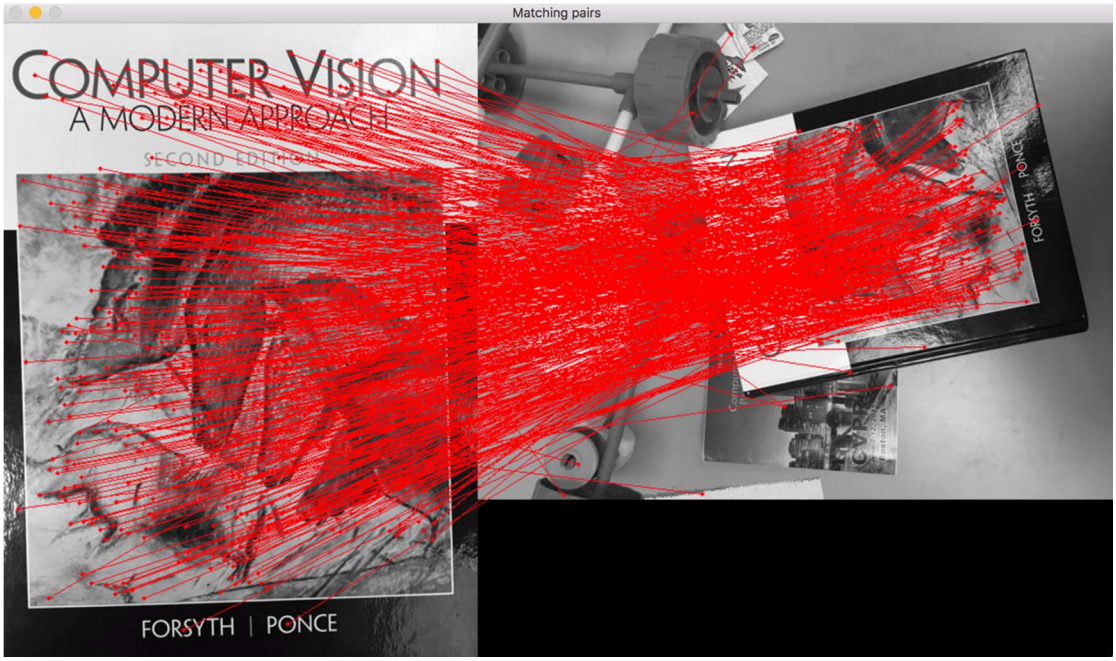

Object and overlay feature points:


Homography matrix obtained:

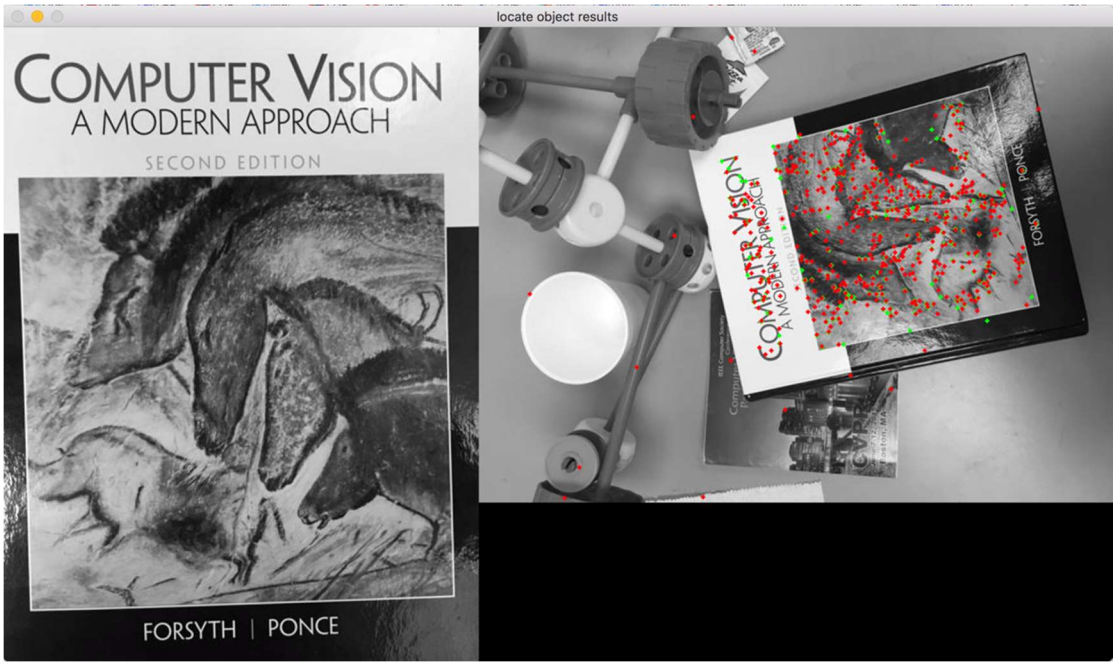
```
Homography matrix:
[-0.2443141501630884, 0.5579916280804772, -262.3180356795171]
[-0.5587270697102293, -0.3346326291230061, 641.152284905858]
[2.293904782763995e-05, -0.0001937606959689621, 1]
Program ended with exit code: 0
```

3. Image_1.jpg as the object image, image_5.jpg as the test image.

Feature matching pairs:




Object and overlay feature points:

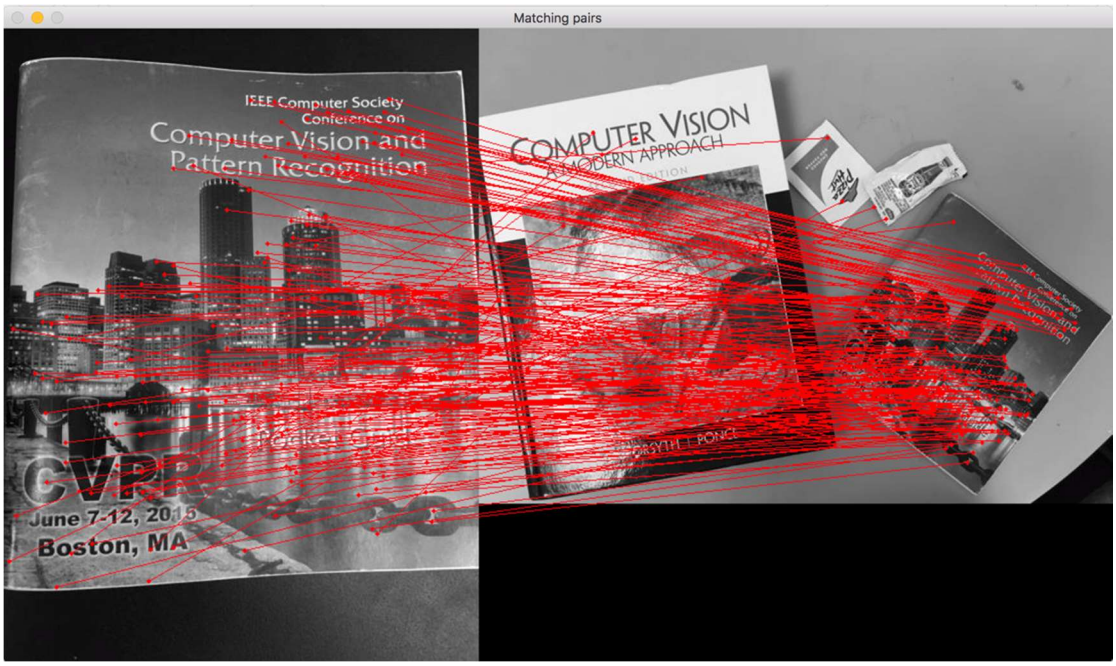



Homography matrix obtained:

```
Homography matrix:
[-0.09754893125931217, 0.4421401234060686, -208.7616057507873]
[-0.4710304593167752, -0.1646035372122745, 503.2008475087304]
[-4.91970338479394e-05, -0.0001533767889768068, 1]
Program ended with exit code: 0
```

4. Image_2.jpg as the object image, image_3.jpg as the test image.

Feature matching pairs:



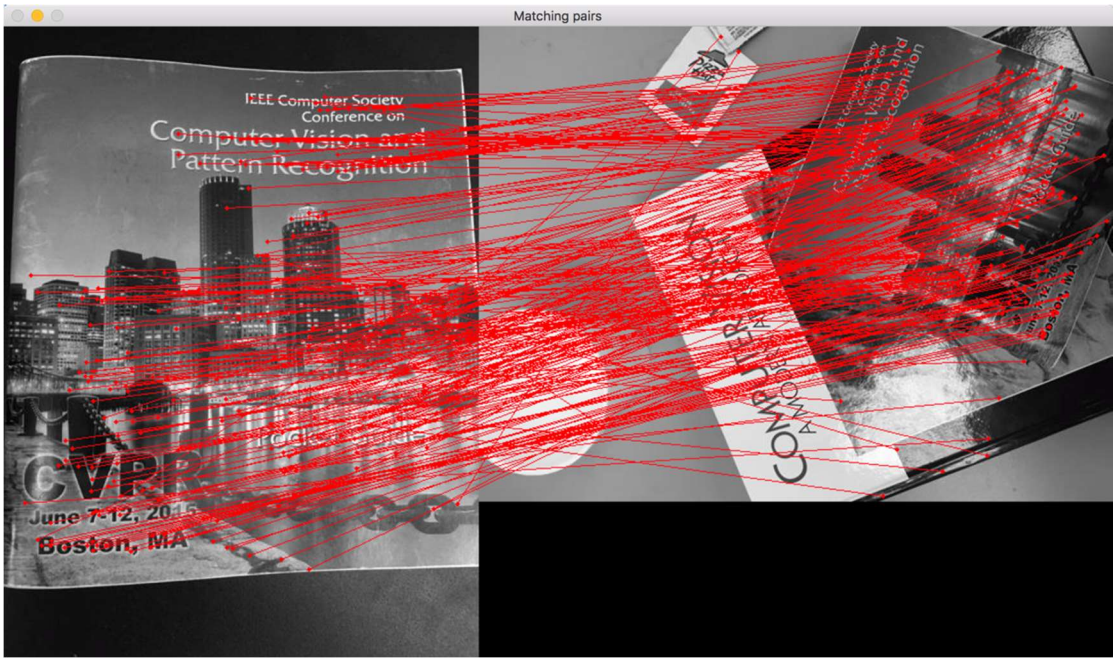

Object and overlay feature points:


Homography matrix obtained:

```
Homography matrix:
[0.3622522465583989, -0.2760515777220469, 361.0432017975423]
[0.1916039043133407, 0.5612841699609624, 311.4598300289734]
[-0.0002135521236844096, 0.0002819167693563227, 1]
Program ended with exit code: 0
```

5. Image_2.jpg as the object image, image_4.jpg as the test image.

Feature matching pairs:



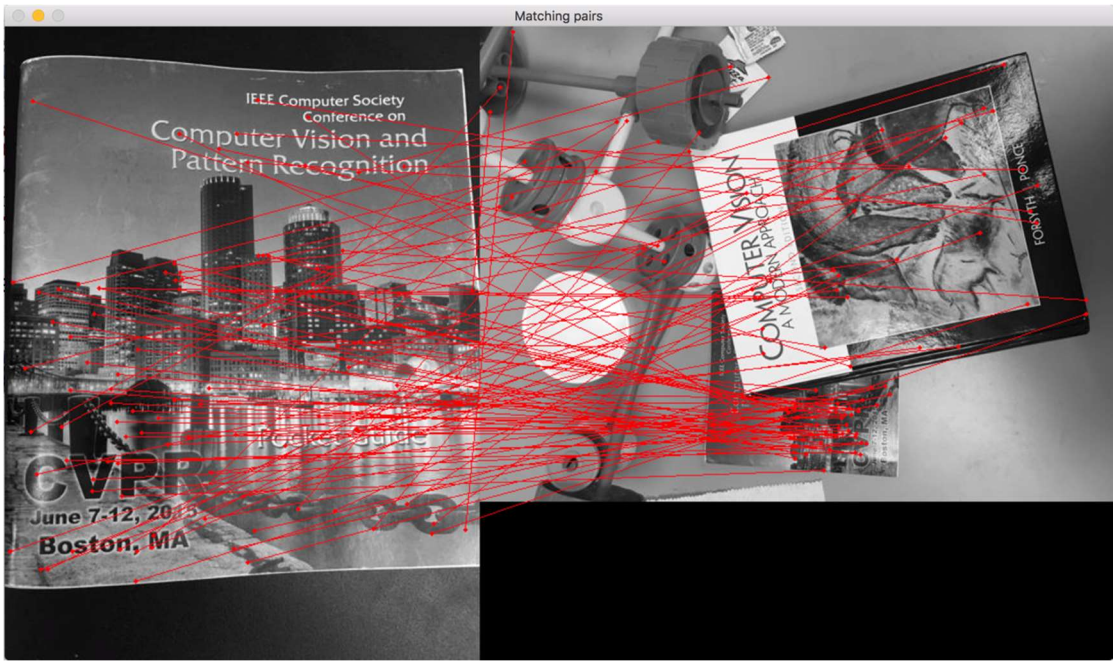

Object and overlay feature points:


Homography matrix obtained:

```
Homography matrix:
[0.0119436495633393, 0.4307471042118751, -253.6609209299693]
[-0.60308068506039, 0.1676539595543163, 578.7208064709754]
[-0.0002681152064800406, 0.0002200269090516706, 1]
Program ended with exit code: 0
```

6. Image_2.jpg as the object image, image_6.jpg as the test image.

Feature matching pairs:




Object and overlay feature points:

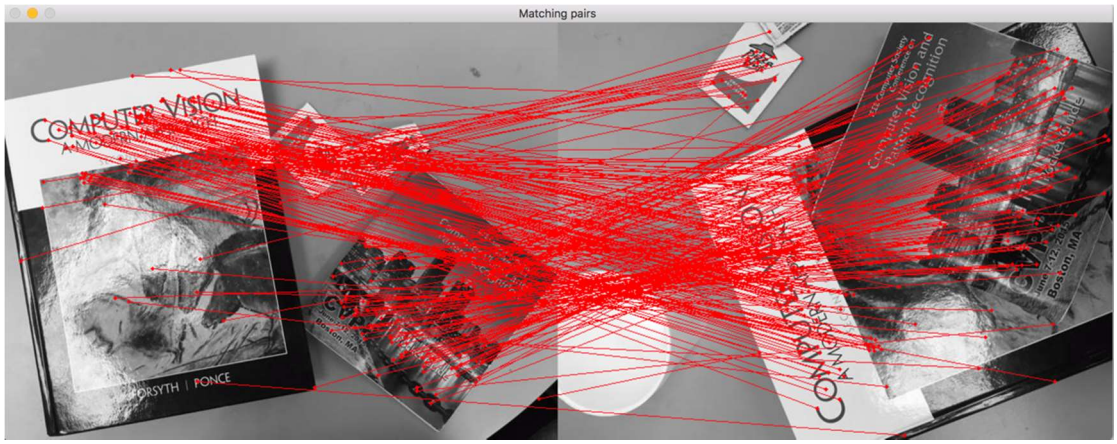

Homography matrix obtained:

```
Homography matrix:
[0.0119436495633393, 0.4307471042118751, -253.6609209299693]
[-0.60308068506039, 0.1676539595543163, 578.7208064709754]
[-0.0002681152064800406, 0.0002200269090516706, 1]
Program ended with exit code: 0
```

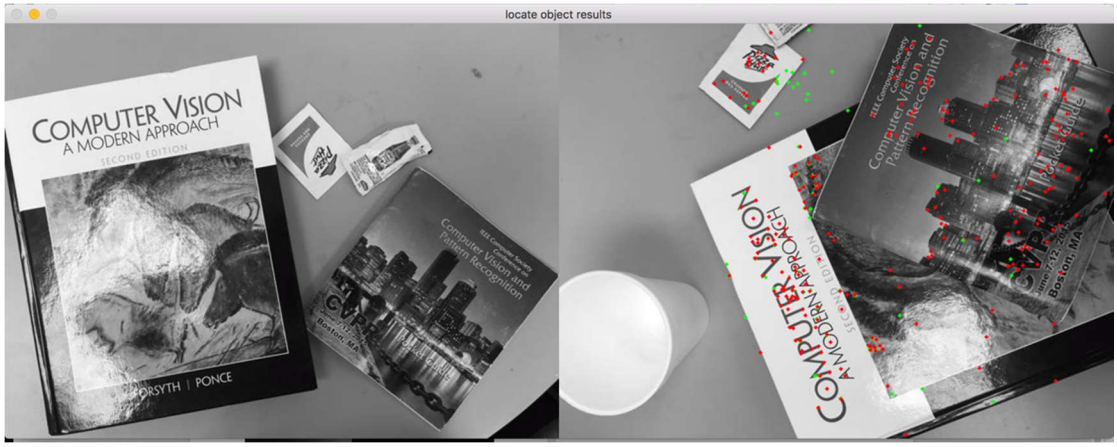
Above 6 test cases are normal cases, below are some special cases.

7. Image_3.jpg as the object image, image_4.jpg as the test image.

   Feature matching pairs:




Object and overlay feature points:




Homography matrix obtained:

```
Homography matrix:
[0.0119436495633393, 0.4307471042118751, -253.6609209299693]
[-0.60308068506039, 0.1676539595543163, 578.7208064709754]
[-0.0002681152064800406, 0.0002200269090516706, 1]
Program ended with exit code: 0
```

8. Image_4.jpg as the object image, image_1.jpg as the test image.

   Feature matching pairs:


Object and overlay feature points:


Homography matrix obtained:

```
Homography matrix:
[-0.5360125873327779, -1.292189137724218, 687.8297087770885]
[1.462678553847238, -0.6053983536532571, 771.6421777864439]
[0.0002963822035967002, -8.285582725455677e-05, 1]
Program ended with exit code: 0
```

## Discussion

The test results mostly locate the objects successfully. Case 1, 3, 4 and 5 have the best results of locating the corresponding books; case 2, 6 and 8 have fair performances; case 7 has worst performance.

From the results above, it can be drawn that the performances of each test case are different, and the accuracy of the program is image-dependent.

One factor is the number of objects in the original image. The best performance of locating the object is achieved by choosing the object as a whole image, i.e. the input image only contains one object. Take case 1 as example, the original image is the whole “F.P.” book, and the test image shows the whole book on the left side of the image. Thus, the object can be located accurately.

Another factor is the wholeness of the object. If the object is covered by other objects in the test image, like case 2 or 6, the performance is too good to predict the points that is not shown in the test image. This introduces good recall, but bad precision. For case 8, the program can only detect part of the object, so this gives good precision but bad recall.

For case 7, the performance is the worst since there are many objects in the original image, and one object is covered by another object.

In conclusion, the performance of this SIFT+RANSAC method is decided by the content of object image and test image. For better performance, object image should contain only the target object, and the object is not covered by other object in test image.

**Function declaration Appendix**

```
/*
 * Summary: check if value is in the vector v.
 * Parameters: vector v, int value.
 * Output: bool value of if the value is in the vector; int value of the index of the
matched pair
 * Note: This function is for determing if a keypoint has already been matched.
 */

pair<bool,int> InVector(vector<pair<int,int>> v, int value);

/*
 * This is another version for Point2f
 */
bool InVector(vector<Point2f> samples, Point2f test);

/*
 * Summary: calculate Euclidean distance of SIFT feature vectors of two images;
then find the best matched keypoints between object image and test image.
 * Parameters: feature descriptors of two images
 * Output: matched pairs vector<>.
 */
vector<pair<int,int>>   matchPairs(Mat features, Mat testFeatures);

/*
 * Summary: This function selects non-repeated random numbers from [1,totalNum].
 * Parameters: number of selection; total number select from.
 * Output: an array of random needed number of numbers
 */

vector<int> RandomIndex(int totalNum, int neededNum);

/*
 * Summary: This function is for solving the linear equations for Homography
coefficients.
 * Parameters: keypoints and keypoints of test image;sampled keypoints and
sampled test image keypoints
 * Output: a homography model fits the 5 keypoints
 */
Mat HomographyModel(vector<Point2f> p1, vector<Point2f> p2,vector<Point2f>
sampleP1, vector<Point2f> sampleP2);
```

```
/*
 * Summary: This function is for checking if the test point satisfied with the model.
 * Parameters: test point, real correct point, homography model, threshold t.
 * Output: bool value of if satisfying
 */
bool SatisfyModel(Point2f test, Point2f p,    Mat model,int t);

/*
 * Summary: This function is for caluculating the error of the model compared to the
corresponding correct points in test image.
 * Parameters: inliers and their pairs of test image, homography model
 * Output: error distance of this model
 */
float CalculateError(vector<Point2f> p1,vector<Point2f> p2,Mat model);

/*
 * Summary: This function is the realization of RANSAC algorithm.
 * Parameters: keypoints and keypoints of test image
 * Output: a best homography model fits for the keypoints
 */
Mat algorithmRANSAC(vector<Point2f> p1, vector<Point2f> p2);
```