Assignment2

April 5, 2022

1 Computer Vision 2022 Assignment 2: Image matching and retrieval

In this assignment, you will experiment with image feature detectors, descriptors and matching. There are 3 main parts to the assignment:

- matching an object in a pair of images
- searching for an object in a collection of images
- analysis and discussion of results

This assignment will have a minimum hurdle of 40%. You will fail if you can not reach the minimum hurdle.

1.1 General instructions

As before, you will use this notebook to run your code and display your results and analysis. Again we will mark a PDF conversion of your notebook, referring to your code if necessary, so you should ensure your code output is formatted neatly.

When converting to PDF, include the outputs and analysis only, not your code. You can do this from the command line using the `nbconvert` command (installed as part of Jupyter) as follows:

```
jupyter nbconvert Assignment2.ipynb --to pdf --no-input --TagRemovePreprocessor.remove_cell_tag
# Or
jupyter nbconvert Assignment2.ipynb --TagRemovePreprocessor.remove_cell_tags='{"remove_cell"}'
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

2 Question 1: Matching an object in a pair of images (45%)

In this question, the aim is to accurately locate a reference object in a query image, for example:

0. Download and read through the paper ORB: an efficient alternative to SIFT or SURF by Rublee et al. You don't need to understand all the details, but try to get an idea of how it works. ORB combines the FAST corner detector (covered in week 4) and the BRIEF descriptor. BRIEF is based on similar ideas to the SIFT descriptor we covered week 4, but with some changes for efficiency.

1. [Load images] Load the first (reference, query) image pair from the "book_covers" category using opencv (e.g. `img=cv2.imread()`). Check the parameter option in " cv2.imread()" to ensure that you read the gray scale image, since it is necessary for computing ORB features.

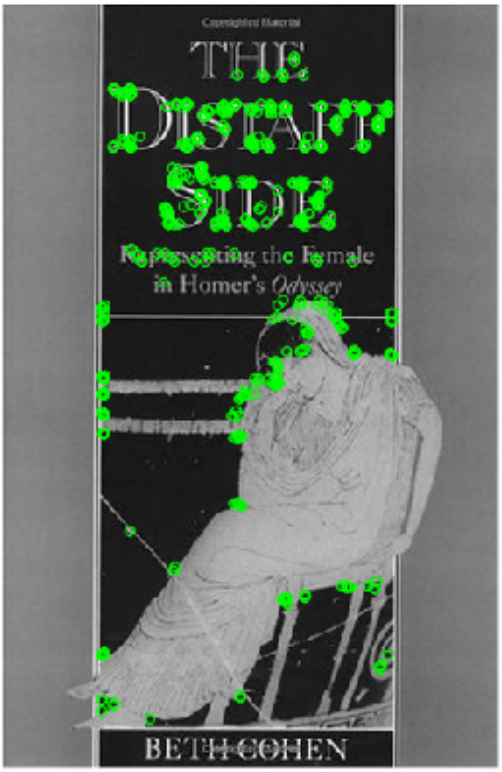
2. [Detect features] Create opencv ORB feature extractor by `orb=cv2.ORB_create()`. Then you can detect keypoints by `kp = orb.detect(img,None)`, and compute descriptors by `kp, des = orb.compute(img, kp)`. You need to do this for each image, and then you can use `cv2.drawKeypoints()` for visualization.

3. [Match features] As ORB is a binary feature, you need to use HAMMING distance for matching, e.g., `bf = cv2.BFMatcher(cv2.NORM_HAMMING)`. Then you are requried to do KNN matching (k=2) by using `bf.knnMatch()`. After that, you are required to use "ratio_test" to find good matches. By default, you can set `ratio=0.8`.

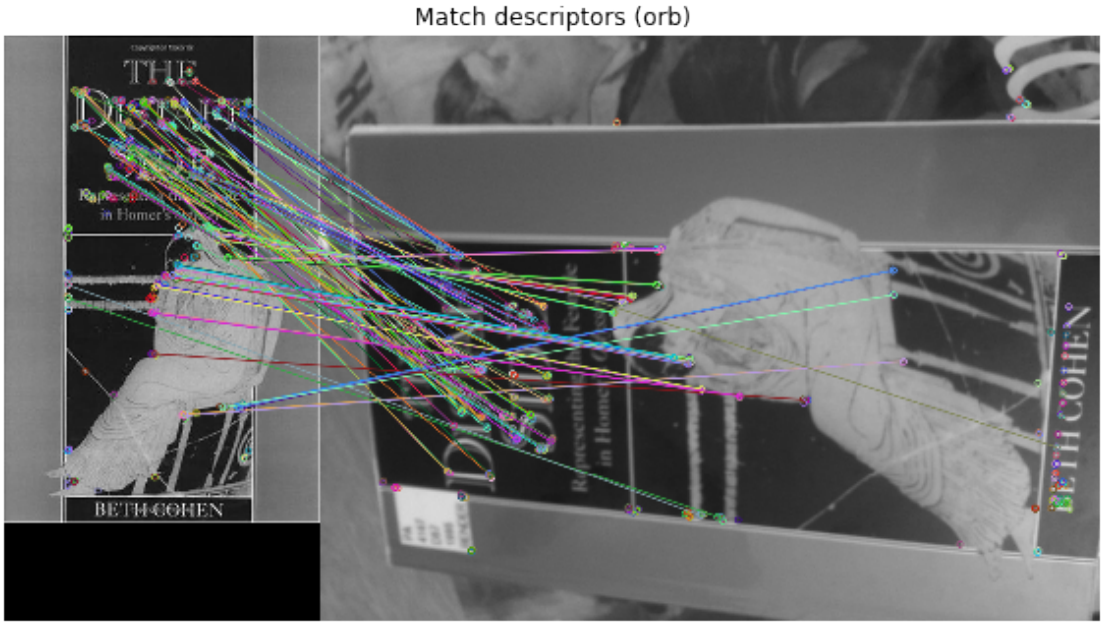
4. [Plot and analyze] You need to visualize the matches by using the `cv2.drawMatches()` function. Also you can change the ratio values, parameters in `cv2.ORB_create()`, and distance functions in `cv2.BFMatcher()`. Please discuss how these changes influence the match numbers.

---

keypoints of reference image (orb)



keypoints of query image (orb)



Your explanation of what you have done, and your results, here

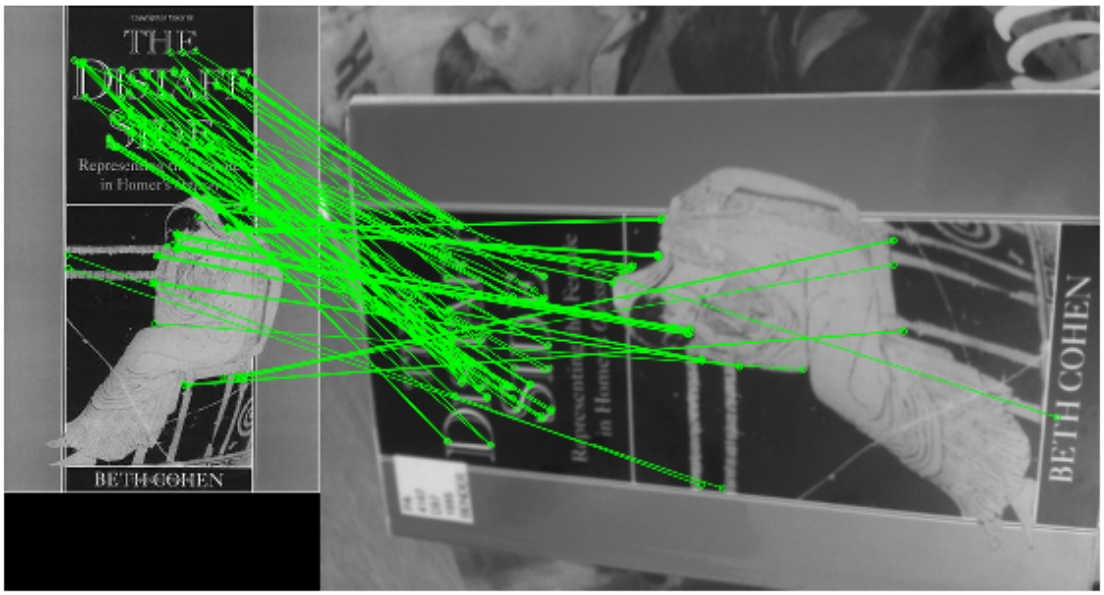
Change the ratio: - when the ratio become small, the keypoints and match numbers become less. - when the ratio become large, the keypoints and match numbers become more.

---

3. Estimate a homography transformation based on the matches, using `cv2.findHomography()`. Display the transformed outline of the first reference book cover image on the query image, to see how well they match.
    - We provide a function `draw_outline()` to help with the display, but you may need to edit it for your needs.
    - Try the 'least squre method' option to compute homography, and visualize the inliers by using `cv2.drawMatches()`. Explain your results.
    - Again, you don't need to compare results numerically at this stage. Comment on what you observe visually.

draw outline (using regular method)

draw inliers (orb)

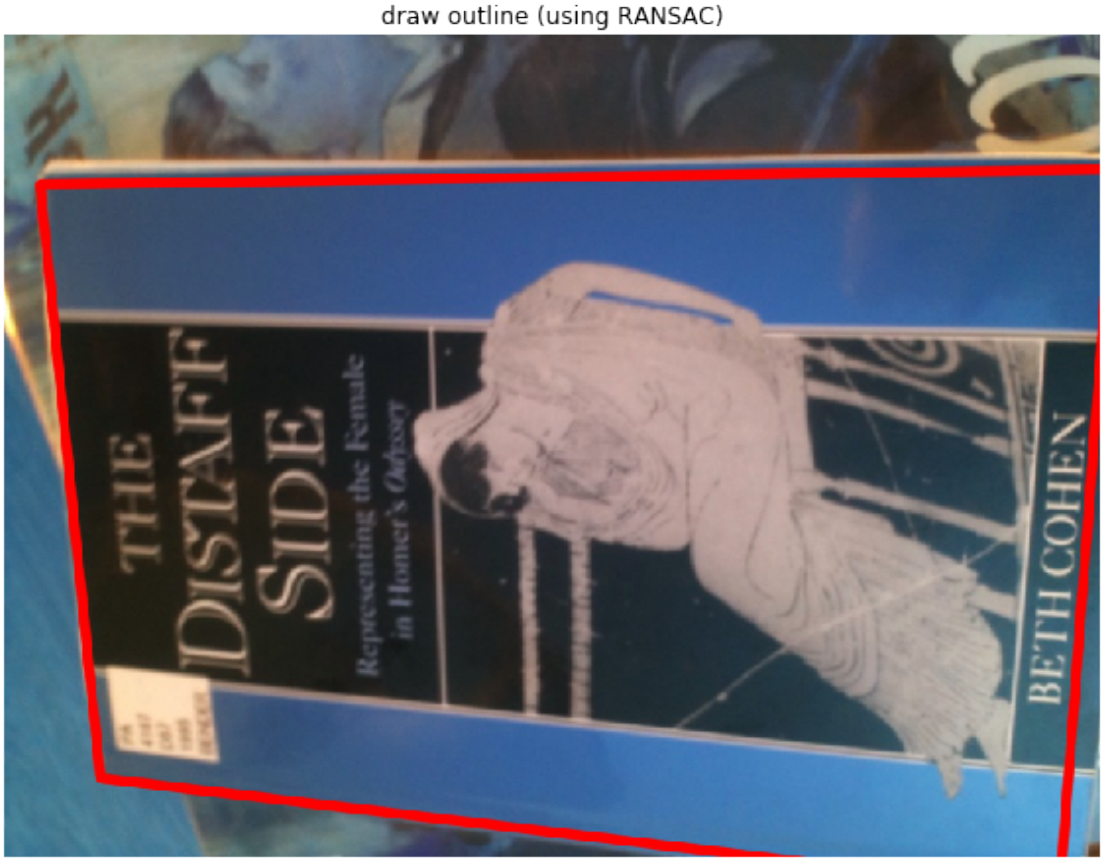


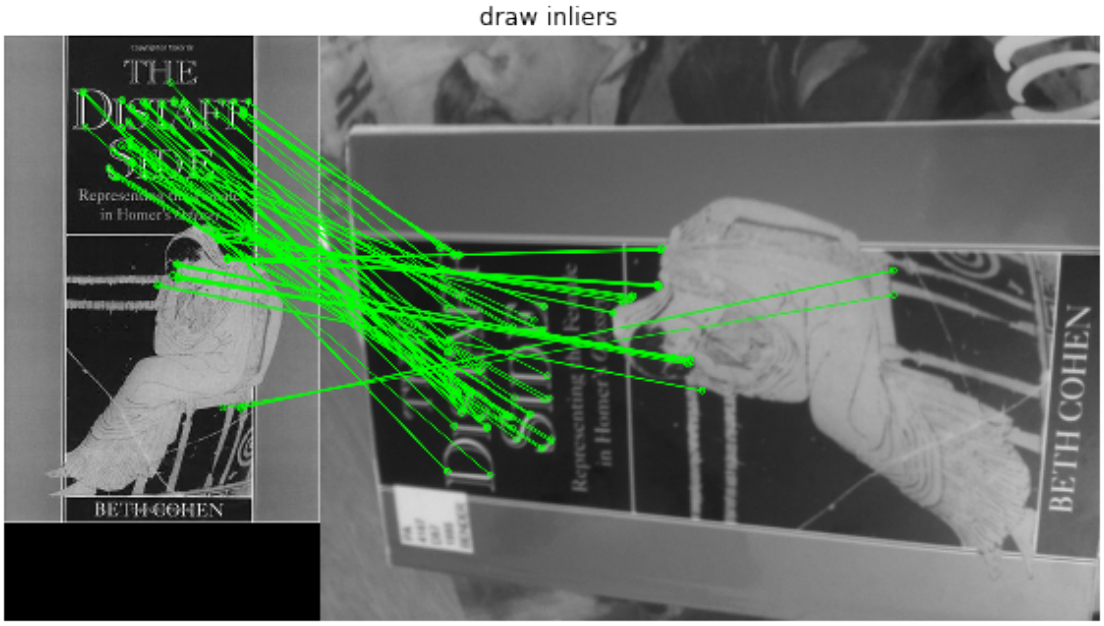
```
number of detected inliers:  131
number of detected outliers:  0.0
```

Your explanation of results here

When I try to use regular method, I notice draw oulines does not work correctly.

Try the RANSAC option to compute homography. Change the RANSAC parameters, and explain your results. Print and analyze the inlier numbers. Hint: use cv2.RANSCA with cv2.findHomography.






```
number of detected inliers:  100
number of detected outliers:  31.0
```

Your explanation of what you have tried, and results here

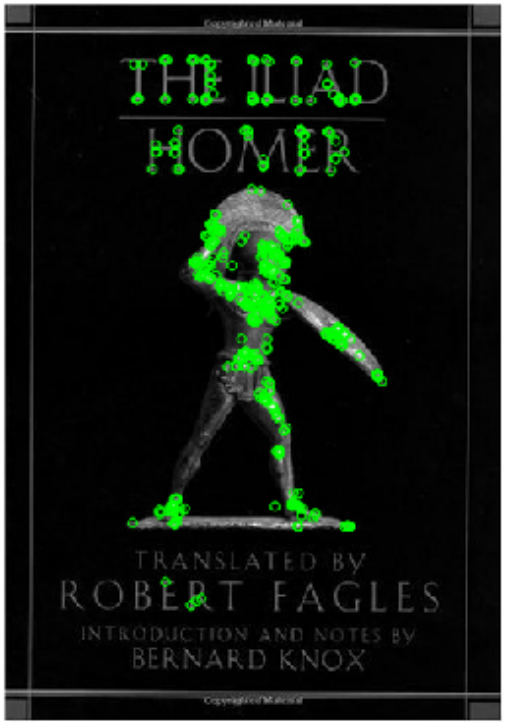
Compare to the RANSAC and the regular method, I notice the number of dectected inliers is different. The total number of detected points are 131, and when I use RANSAC method, the number of detected inliers are only 100, however, when I use regular method, the number of detected inliers are 131. There is a difference of 31 points. However, it may influence when draw outlines.

---

6. Finally, try matching several different image pairs from the data provided, including at least one success and one failure case. For the failure case, test and explain what step in the feature matching has failed, and try to improve it. Display and discuss your findings.
    1. Hint 1: In general, the book covers should be the easiest to match, while the landmarks are the hardest.
    2. Hint 2: Explain why you chose each example shown, and what parameter settings were used.
    3. Hint 3: Possible failure points include the feature detector, the feature descriptor, the matching strategy, or a combination of these.

2.0.1 book covers

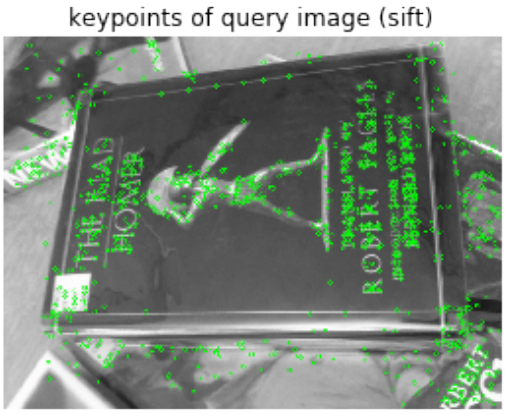
keypoints of reference image (orb)



keypoints of query image (orb)

keypoints of reference image (brisk)

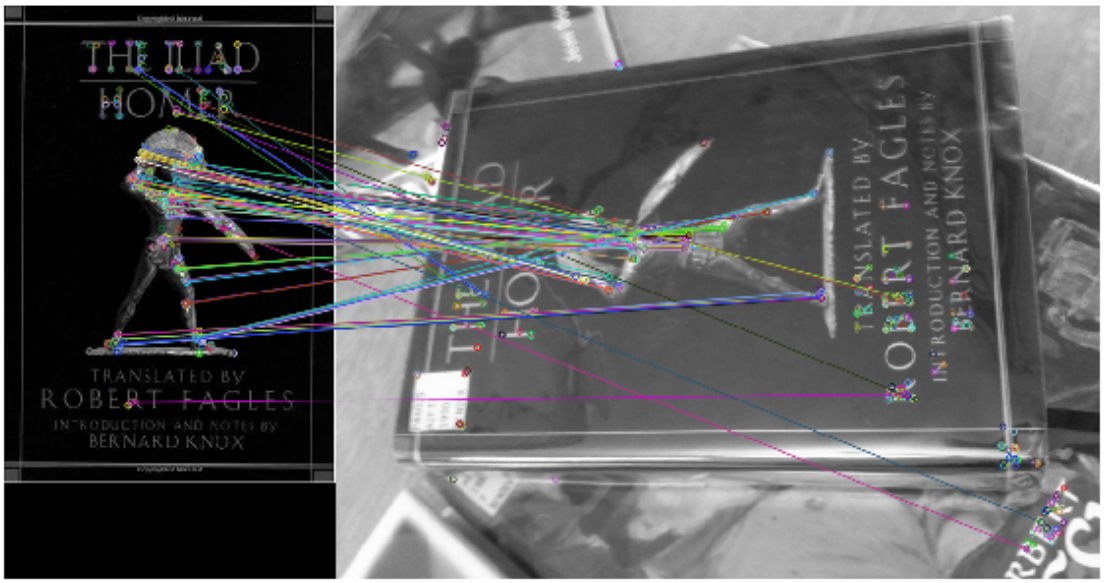
keypoints of query image (brisk)



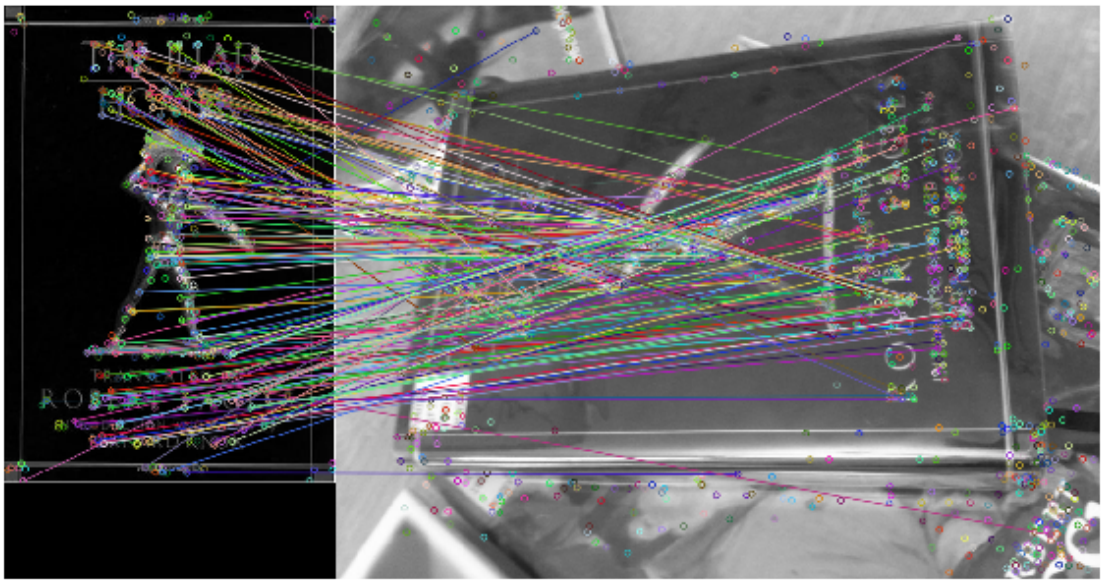
2.0.2 find key ponits

In ***getting keypoints***, the shift works better than orb and the brisk, it returns more key points in the query image and the reference image.

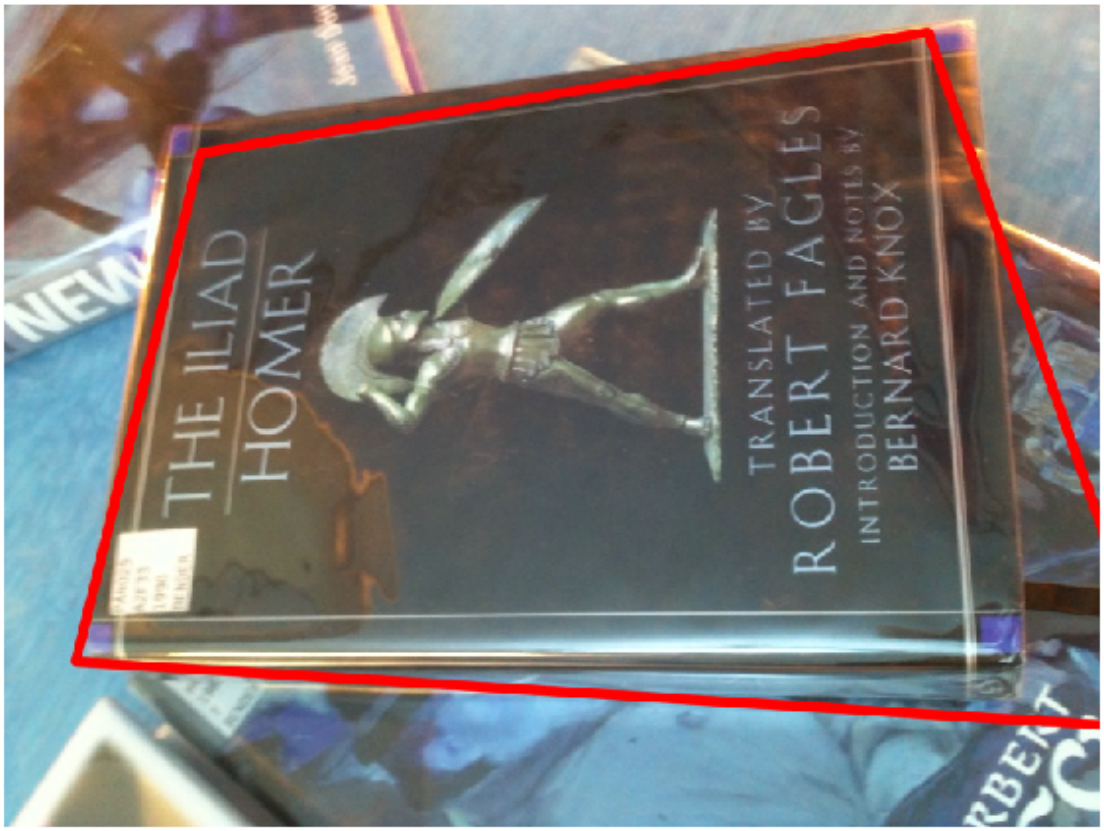
matches (orb)



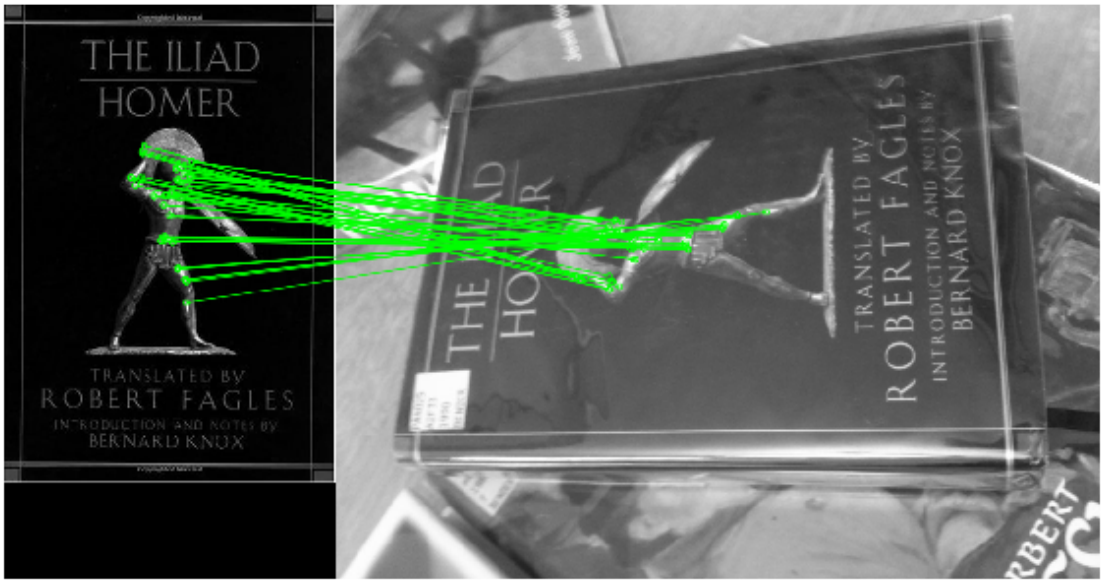
matches (sift)



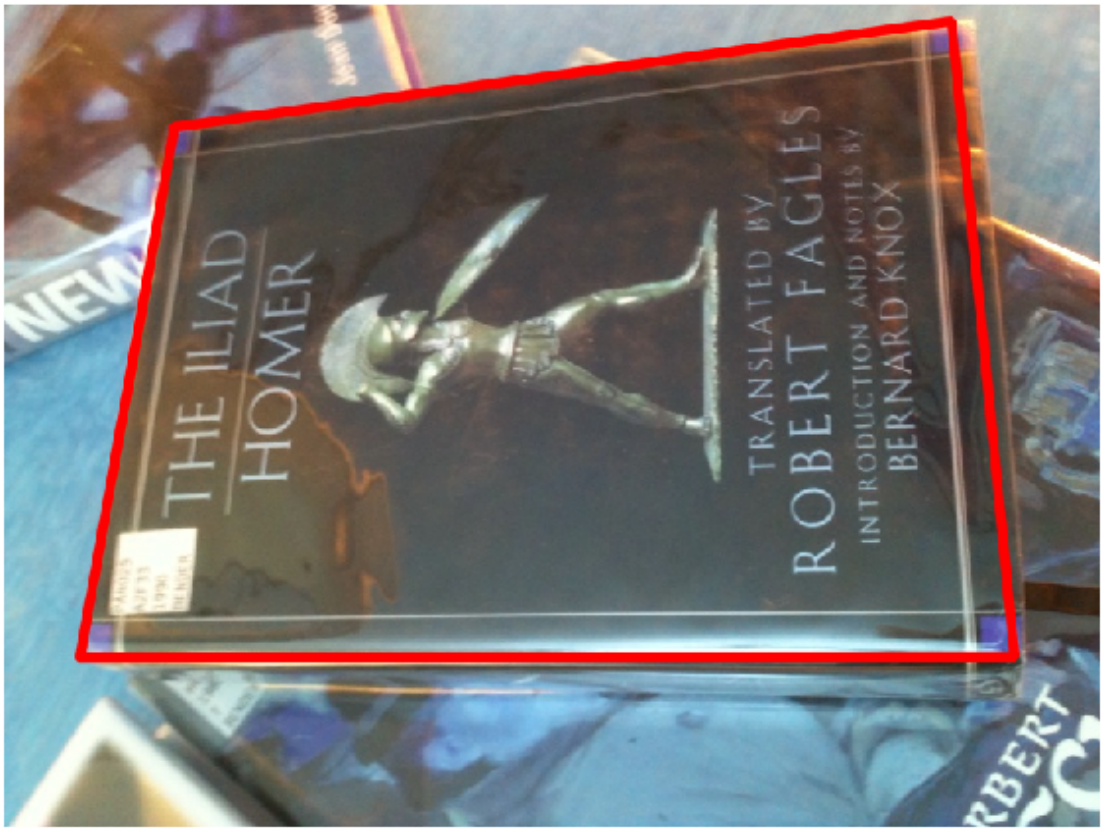
draw outline (orb)



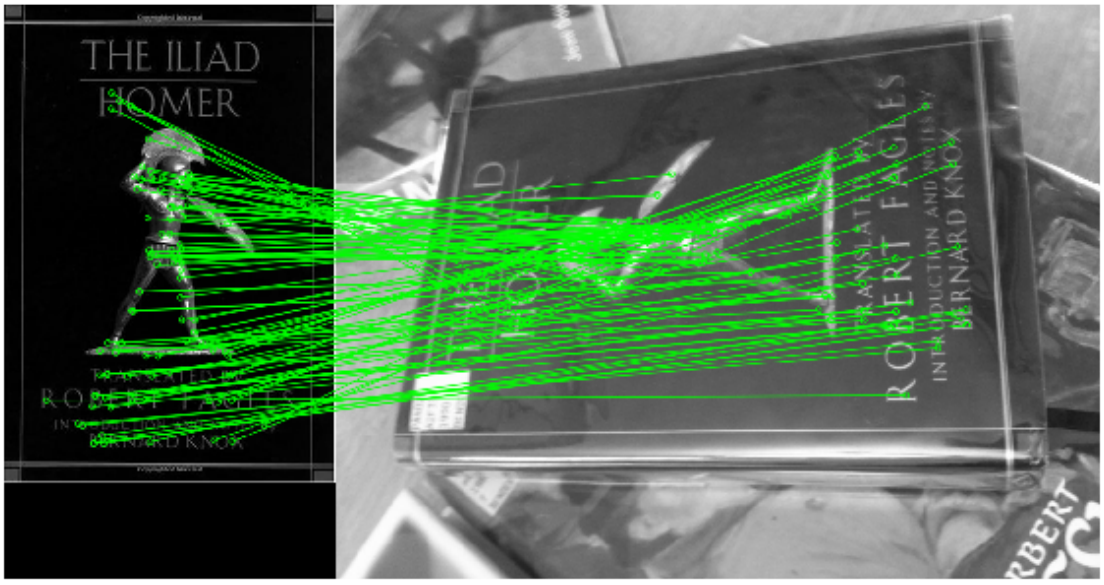
draw inliers (orb)



draw outline (sift)



draw inliers (sift)



2.0.3 draw outline (example book_cover_025)

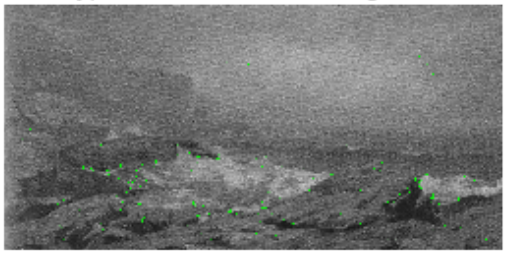
In image 25 from book_covers. we can see when we use orb method to get the outline, it is not very precise, however, sift method works good.

Reason (why orb method will fail): when I use orb method, the keypoints of query image do not detect many keypoints in this area (draw in the picture_01). So when do matches, some keyponits will match the keypoints which detect ouside the book cover, this is why we can not get the correct area when we draw outlines.

picture_01

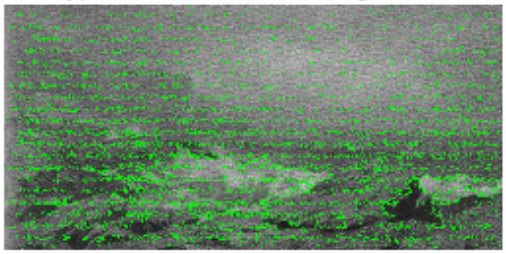
2.0.4 museum_paintings

keypoints of reference image (orb)



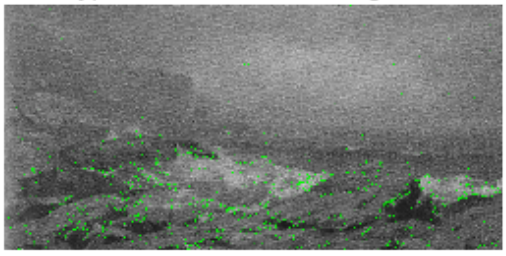
keypoints of query image (orb)

keypoints of reference image (brisk)



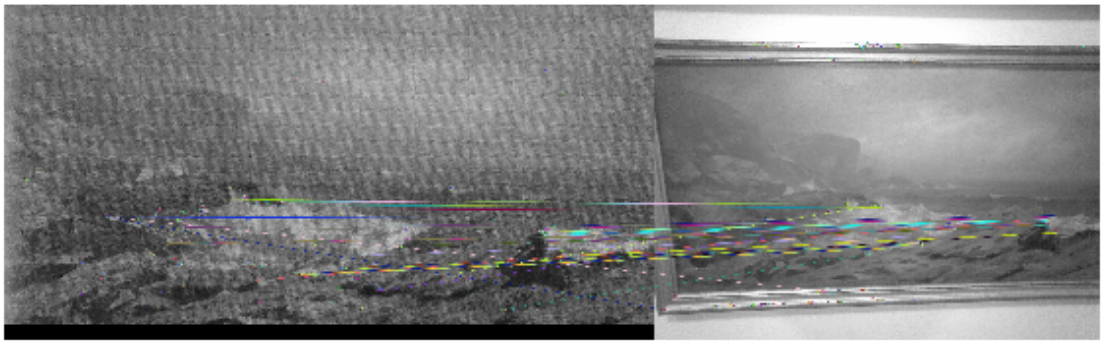
keypoints of query image (brisk)

keypoints of reference image (sift)

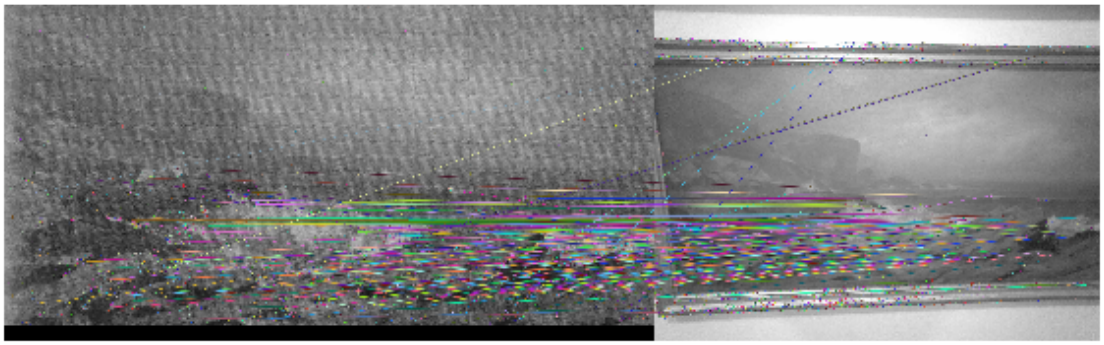


keypoints of query image (sift)

matches (orb)

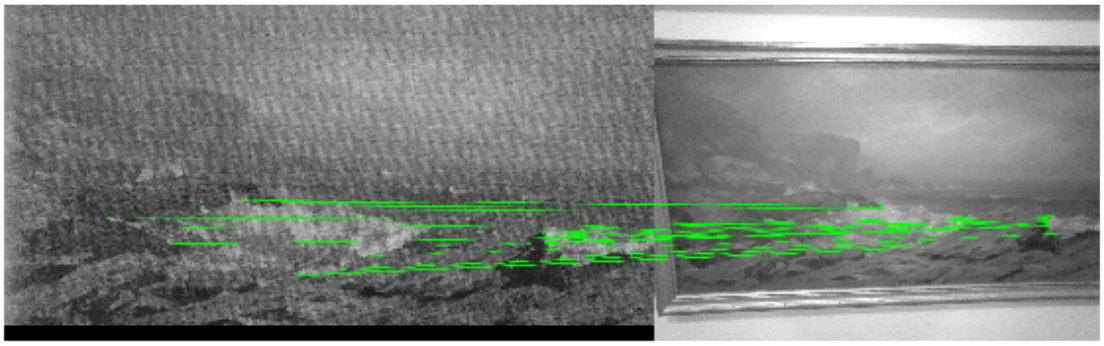


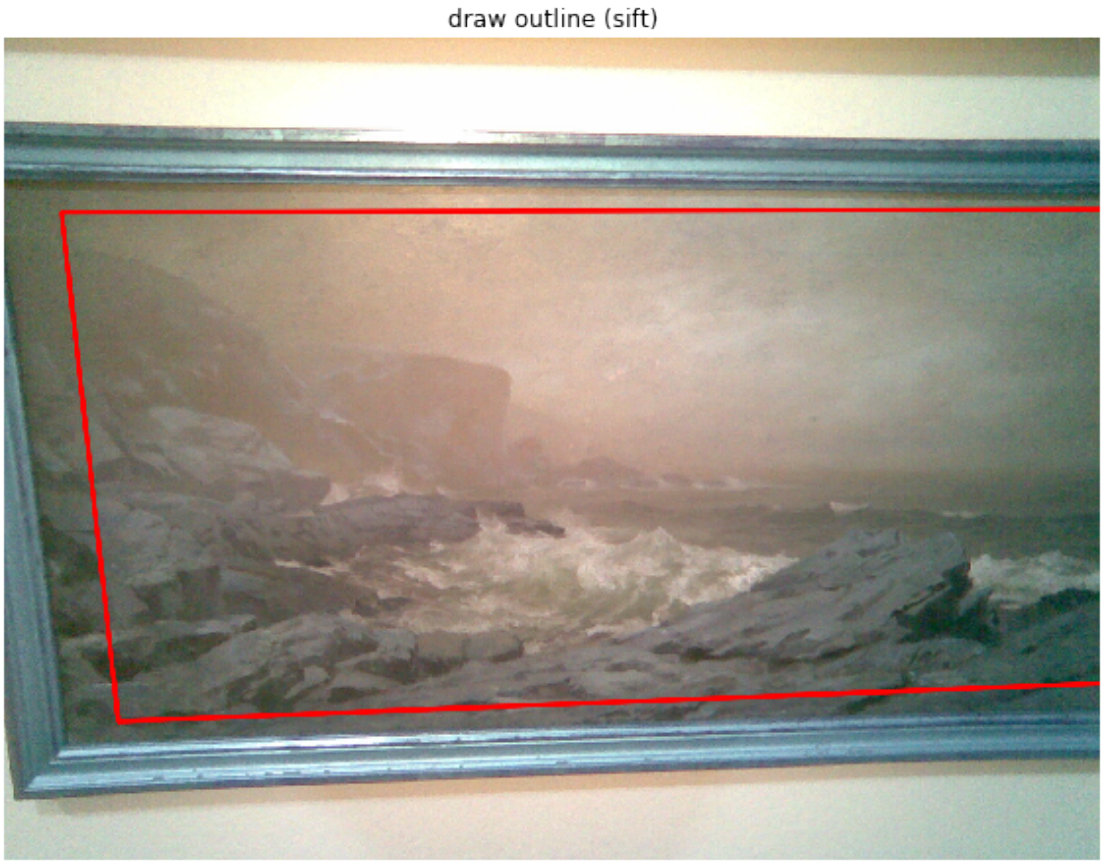
matches (sift)

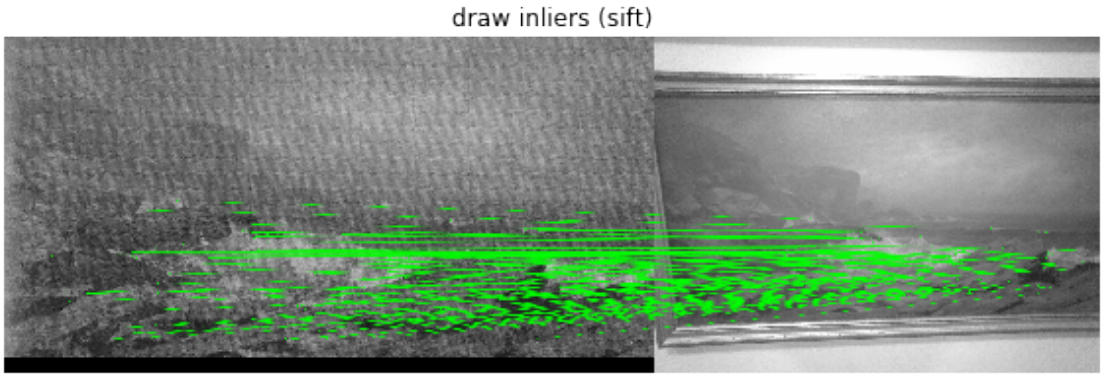


draw outline (orb)

draw inliers (orb)







Your explanation of results here

2.0.5 museum paintings

When I use orb to detect the keypoints of the image, it returns only a few keypoints of the query image and the reference image.

I find the main reason that find the refernce image in the query image (draw outlines) will fail is that there is too few keyponits in the reference image.

such as some museum paintings (34, 54, 74 and so on) which have reflections, so I think it is an interfering factor which cause the fail.

3 Question 2: What am I looking at? (40%)

Your explanation of what you have done, and your results, here

5. Choose some extra query images of objects that do not occur in the reference dataset. Repeat step 4 with these images added to your query set. Accuracy is now measured by the percentage of query images correctly identified in the dataset, or correctly identified as not occurring in the dataset. Report how accuracy is altered by including these queries, and any changes you have made to improve performance.

Your explanation of results and any changes made here

6. Repeat step 4 and 5 for at least one other set of reference images from museum_paintings or landmarks, and compare the accuracy obtained. Analyse both your overall result and individual image matches to diagnose where problems are occurring, and what you could do to improve performance. Test at least one of your proposed improvements and report its effect on accuracy.

Your description of what you have done, and explanation of results, here

4 Question 3 (10%)

In Question 1, We hope that `ratio_test` can provide reasonable results for RANSAC. However, if it fails, the RANSAC may not get good results. In this case, we would like to try an improved matching method to replace the `ratio_test`. Here, the `gms_matcher` is recommened. You need to implement it and save results of 3 image pairs (you can select any image pairs from the dataset), where you new method is better than 'ratio_test'.

1. Hint 1: `cv2.xfeatures2d.matchGMS()` can be used, but you need to install the opencv-contrib by `pip install opencv-contrib-python`
2. Hint 2: You do not need use KNN matching, because GMS does not require second nearest neighbor.
3. Hint 3: You need to change the parameters in `cv2.ORB_create()` for best results. See the setting in Github.
4. Hint 4: If your are interested in more details. Read the paper "GMS: Grid-based Motion Statistics for Fast, Ultra-robust Feature Correspondence", and the Github "https://github.com/JiawangBian/GMS-Feature-Matcher".

Your results here

5 Question 4: Reflection Questions (5%)

1. Describe the hardest situation you faced during the first two assignments. And how you overcome it? (3%)

2. How do you plan to finish the assignment to meet tight deadline? (2%)