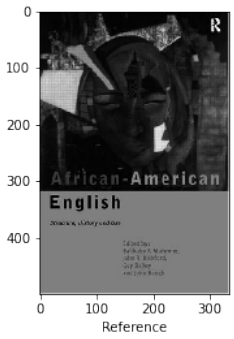
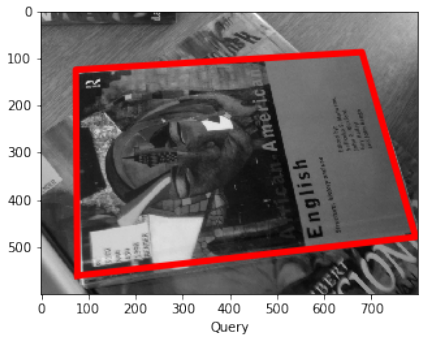
Assignment2

May 1, 2024

1 Question 1: Matching an object in a pair of images (60%)

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





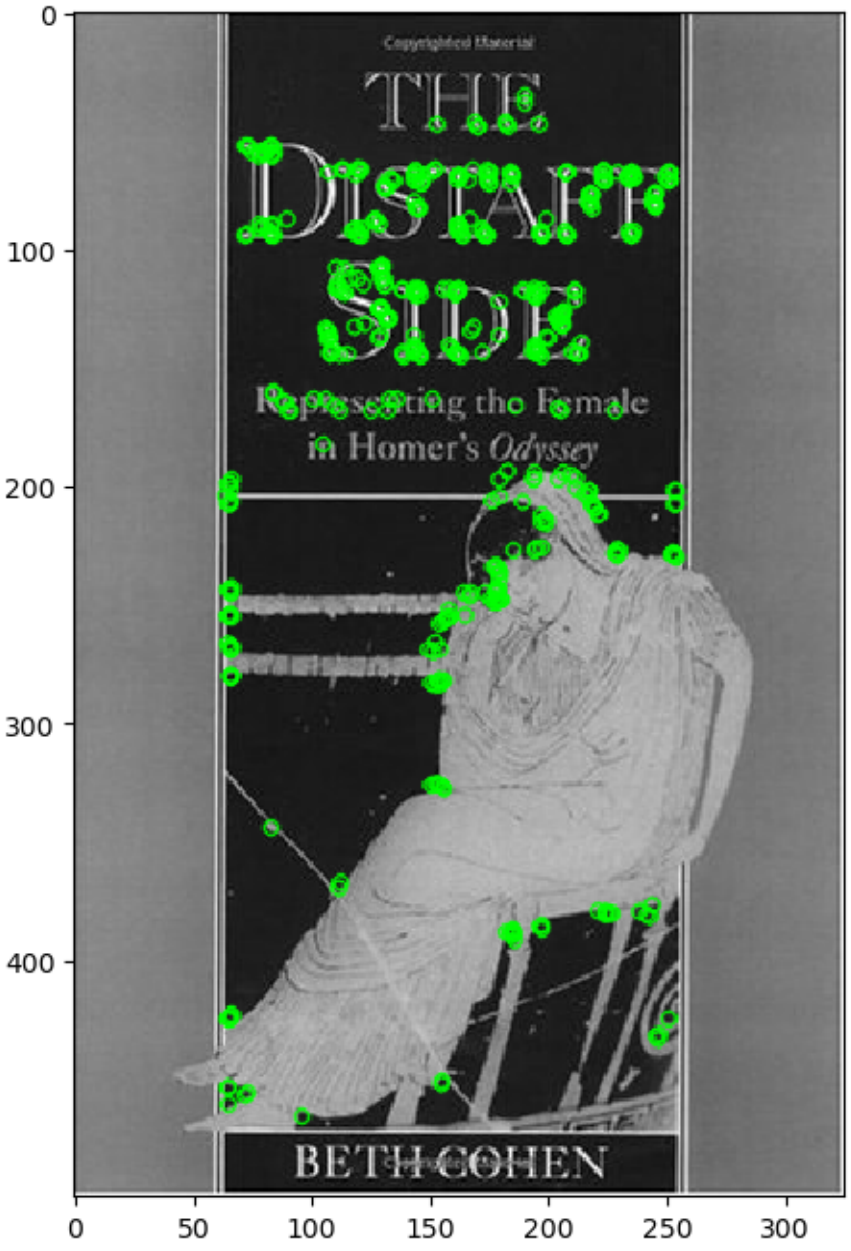
0. Download and read through the paper ORB: an efficient alternative to SIFT or SURF by Rublee et al. You don't need to understand all the details, but try to get an idea of how it works. ORB combines the FAST corner detector (covered in week 3) and the BRIEF descriptor. BRIEF is based on similar ideas to the SIFT descriptor we covered week 3, but with some changes for efficiency.

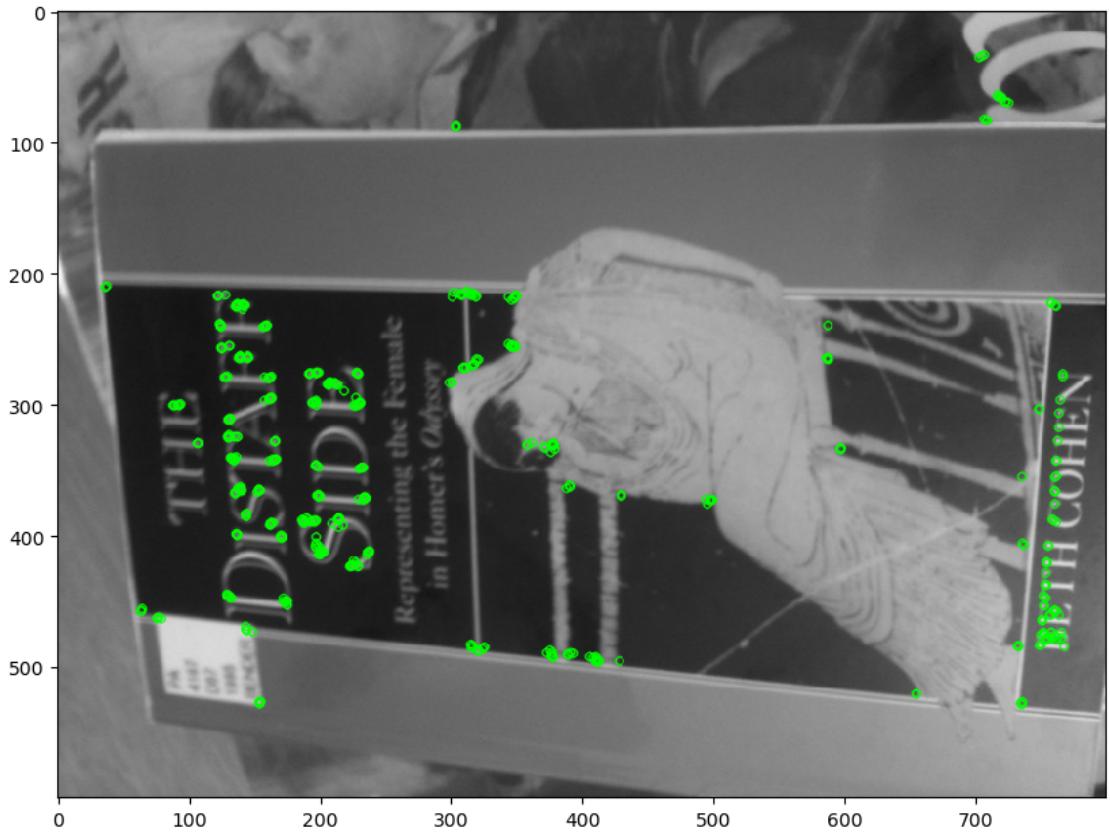
1. [Load images] Load the first (reference, query) image pair from the "book_covers" category using opencv (e.g. `img=cv2.imread()`). Check the parameter option in " cv2.imread()" to ensure that you read the gray scale image, since it is necessary for computing ORB features.

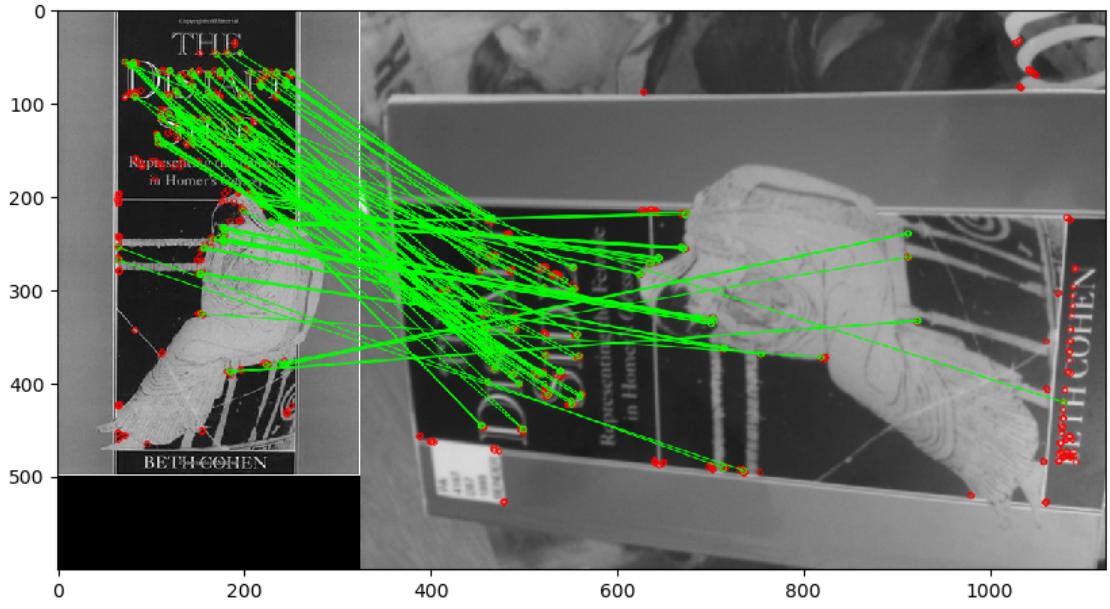
2. [Detect features] Create opencv ORB feature extractor by `orb=cv2.ORB_create()`. Then you can detect keypoints by `kp = orb.detect(img,None)`, and compute descriptors by `kp, des = orb.compute(img, kp)`. You need to do this for each image, and then you can use `cv2.drawKeypoints()` for visualization.

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.


Copyrighted Material
THE
DISTAFF
SIDE
Representing the Female
in Homer's Odyssey
BETH COHEN
0
100
200
300
400
0
50
100
150
200
250
300


THE DISTANT SIDE
Representing the Female in Homer's Odyssey
BETH COHEN


BETH COHEN

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

Starting by implementing "orb=cv2.ORB_create()", the report uses Oriented FAST and Rotated BRIEF algorithm to detect keypoints and compute descriptors for each image where keypoint represents distinct,scale and rotate invariant point on images and descriptors contain information about neighbourhood of each point, similar to SIFT. Some of ORB_create()'s parameter are 'nfeatures' - maximum of features to retain, 'scaleFactor' - pyramid scale factor and 'nlevels' - size of pyramid level, thus changing these factor could affect quantity and quality of detected points, for example by changing nfeatures to 10, it directly limit the number of detection to 10. By leaving the parameter empty, it is expected to return all possible point using default setting.

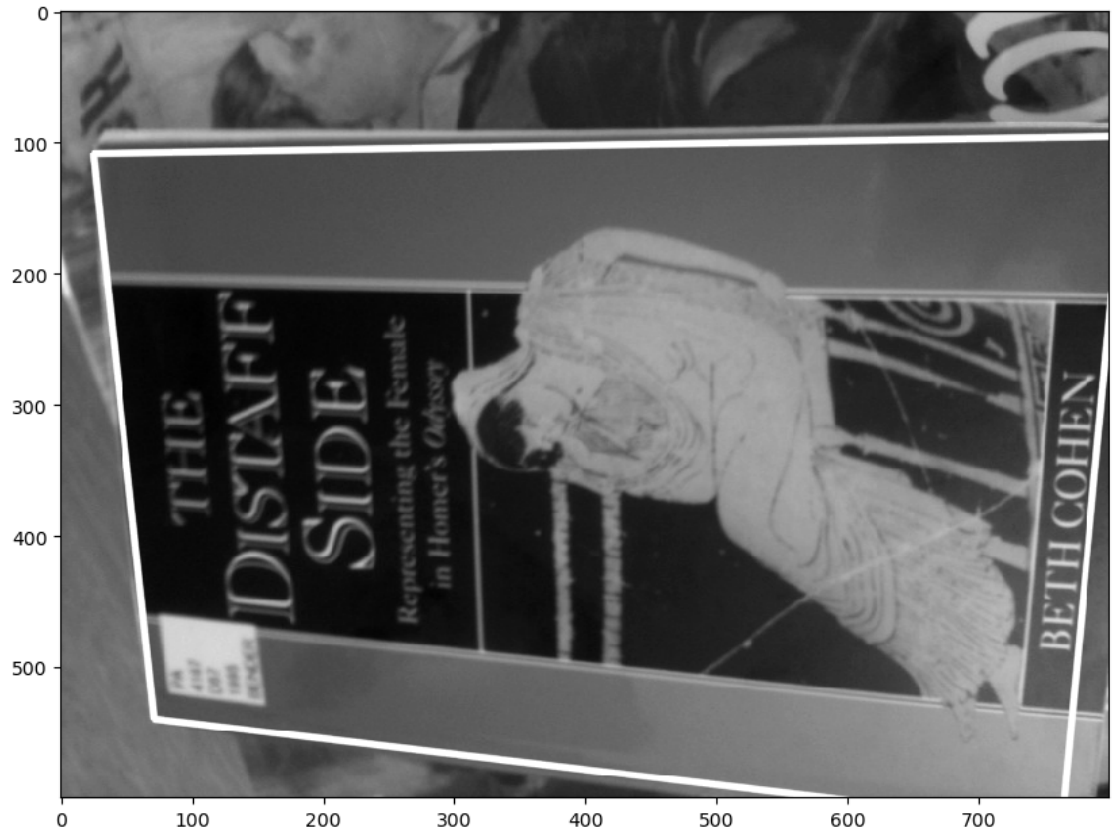
After dectecting all keypoints and descriptors then visualize them through "cv2.drawKeypoints()", "cv2.BFMatcher(cv2.NORM_HAMMING)" is implied to create a brute force matcher using Hamming distance as the distance measurement. By default, without specifying the parameter with "cv2.NORM_HAMMING", cv2.NORM_L2 will be used which is less efficient in this case as L2 distance is euclidean distance taht provides more false negative for binary descriptors. The other parameter is 'crossCheck' which is used for returning only best matches that will conflit with KNN Matching and the proccess of finding good matches later on, thus it is set by default as False.

Finaly, after implementing KNN matching through "bf.knnMatch()", the report have filtered out good matches using given ratio. The ratio_test regards the comparision between the best match's distance - "m.distance" with the distance of second best match or "n.distance". Thus, high ratio tends to provide more matches but low in quality while lower ration provide less matches but higher in quality as it filter out matches that are not significantly better.

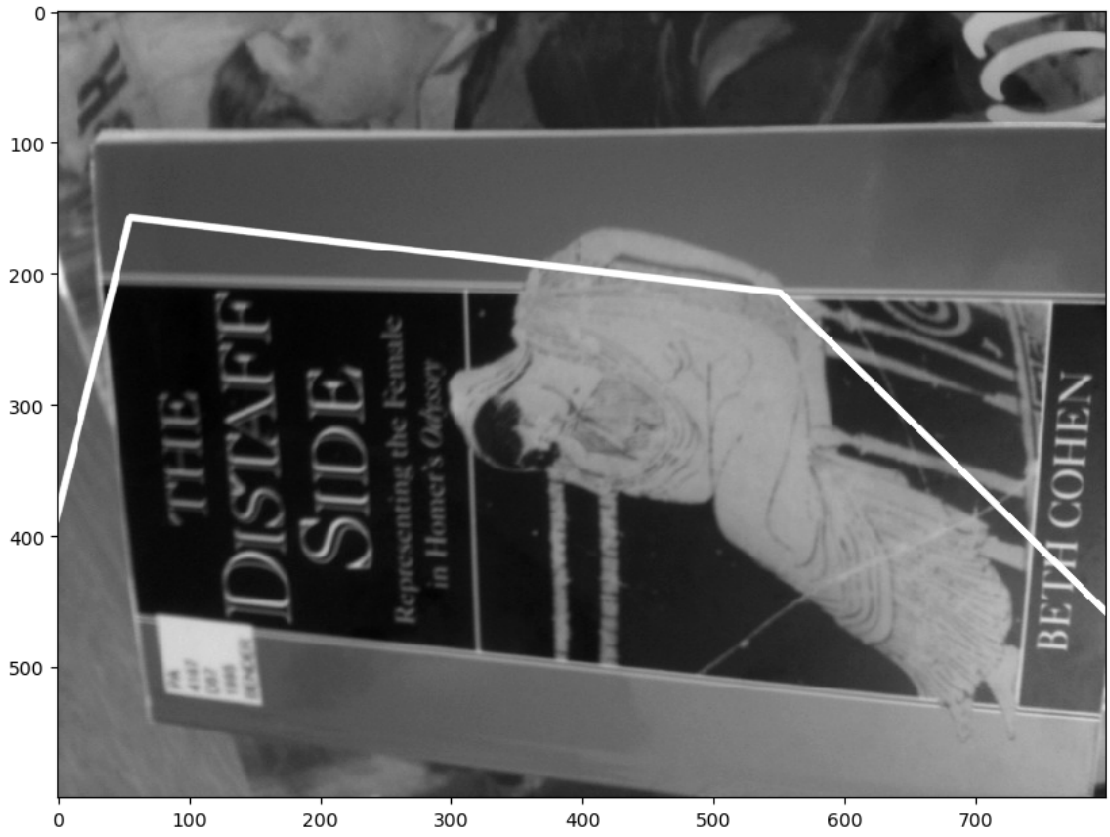
3. Estimate a homography transformation based on the matches, using `cv2.findHomography()`. Display the transformed outline of the first reference book cover image on the query image, to see how well they match.

    - 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.

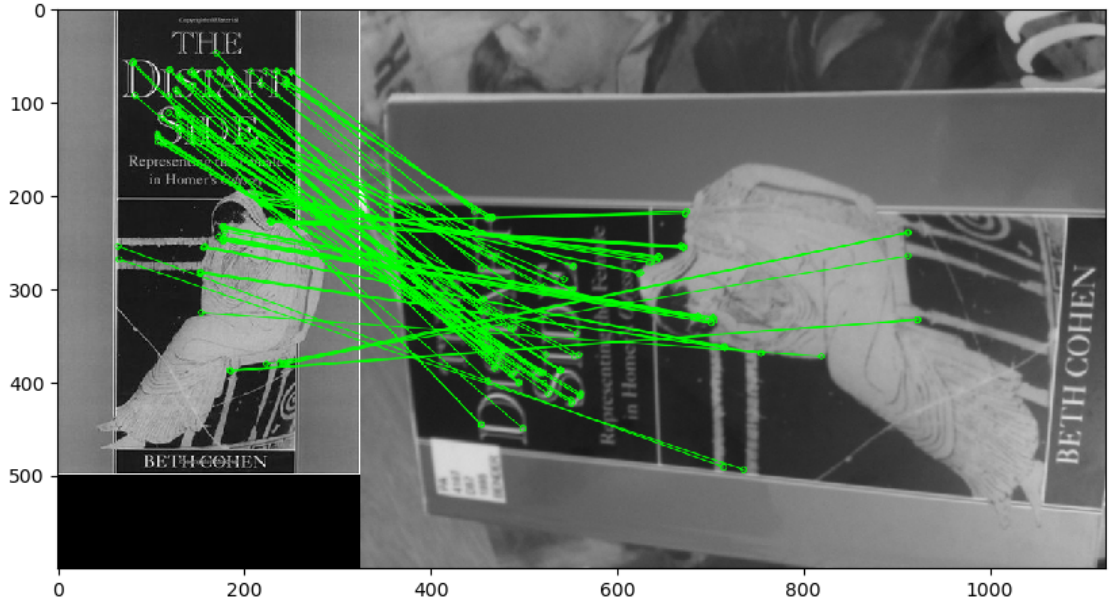
`With Least-Median of squares robust method`



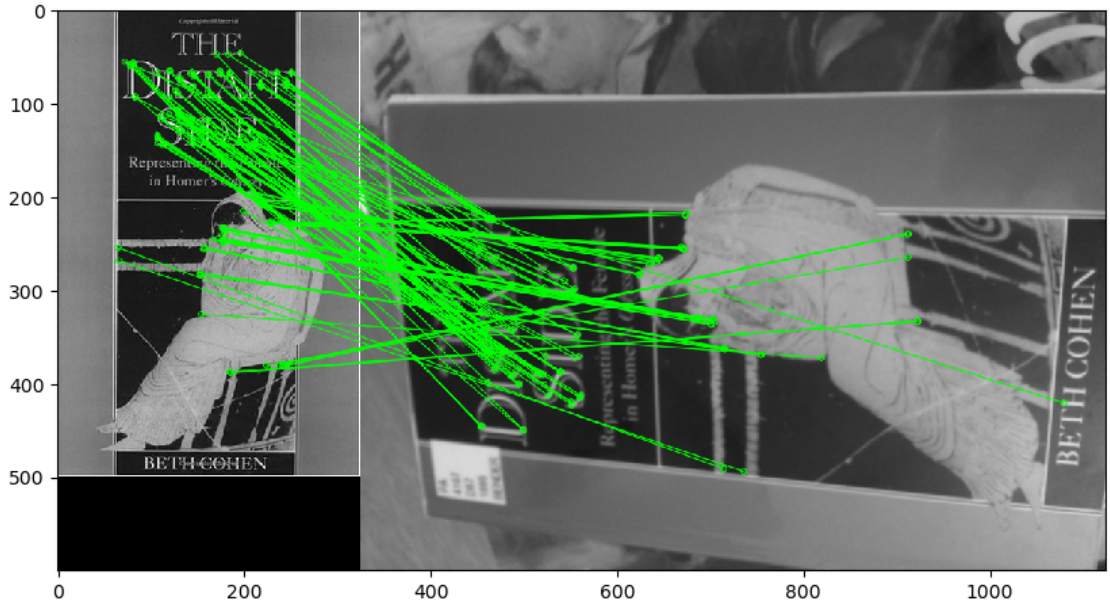
With Least Square Method



With Least-Median of squares robust method



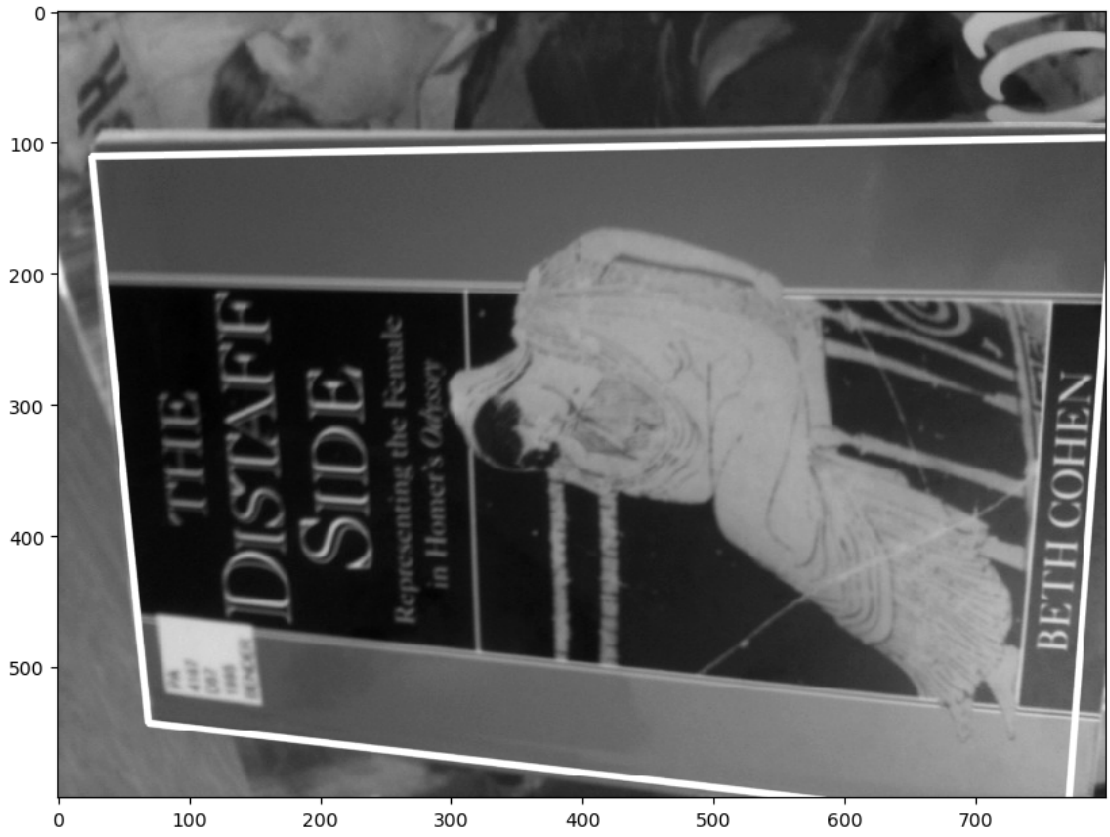
With Least Square Method

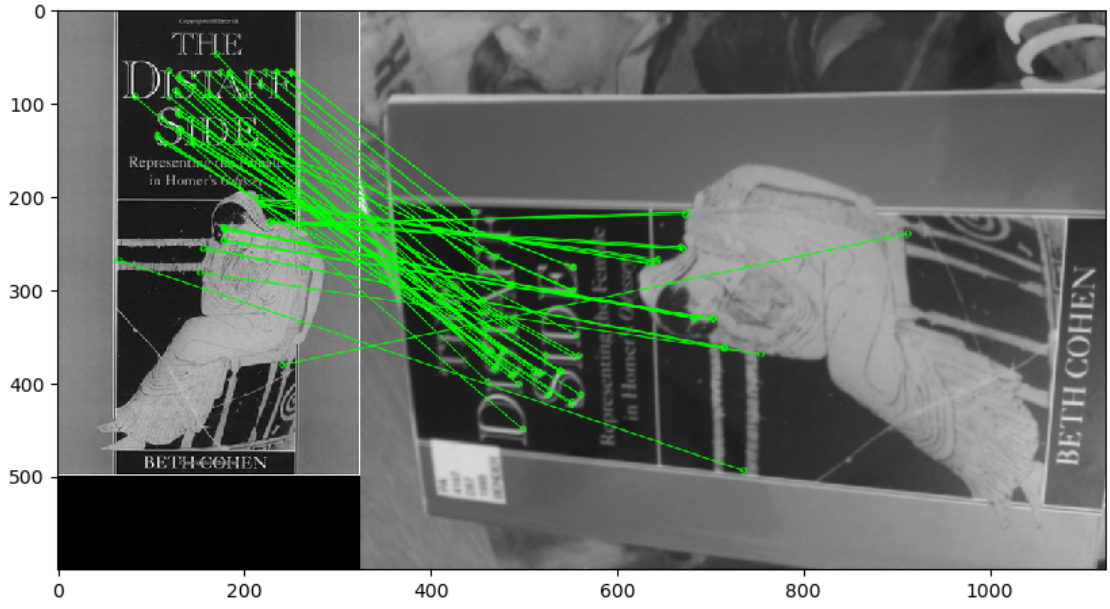


Your explanation of results here

By setting "method = 0" inside the parameter of "findHomography()", it implies the method of Least Square while "cv2.LMEDS" enable Least-Median of squares robust method. Visually, the two method provide almost identical inliners with Least Square Method provides some extra point while there is a different in the present of outlier as Least-Median method provide a more fit outlier.

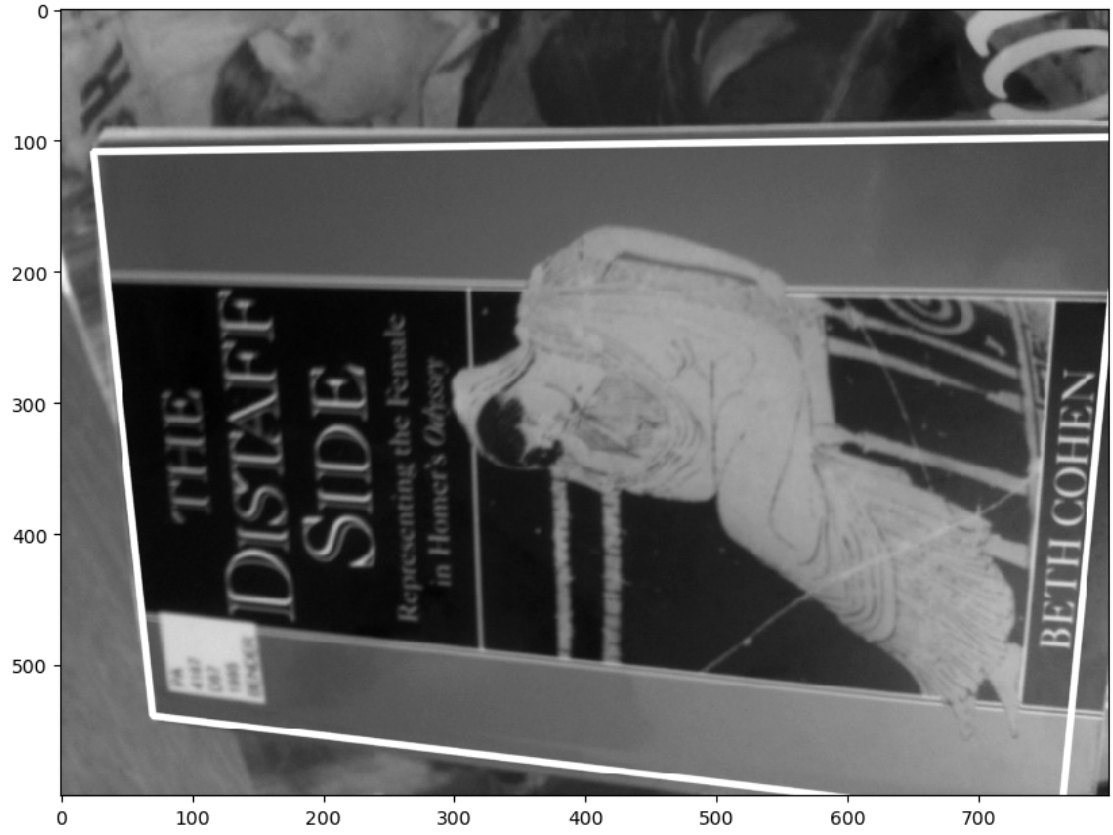
Try the RANSAC option to compute homography. Change the RANSAC parameters, and explain your results. Print and analyze the inlier numbers.

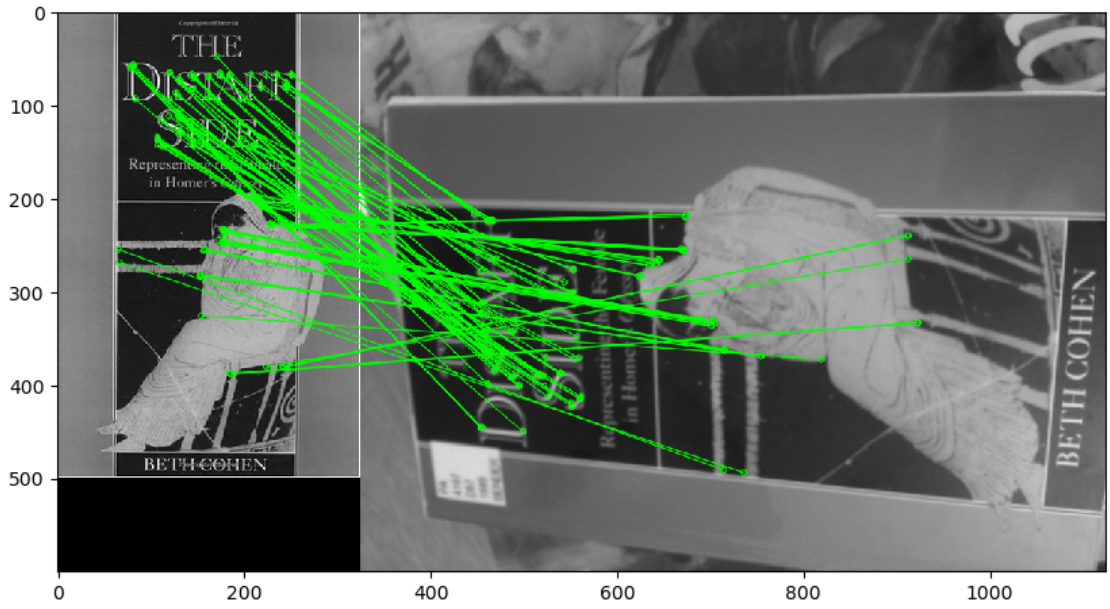

THE
DISTAFF
SIDE
Representing the Female
in Homer's Odyssey
BETH COHEN


THE
DISTAFF
SIDE
BETH COHEN
BETH COHEN

```
The number of inliers is: 64
```

```
RANSAC with threshold of 9
```



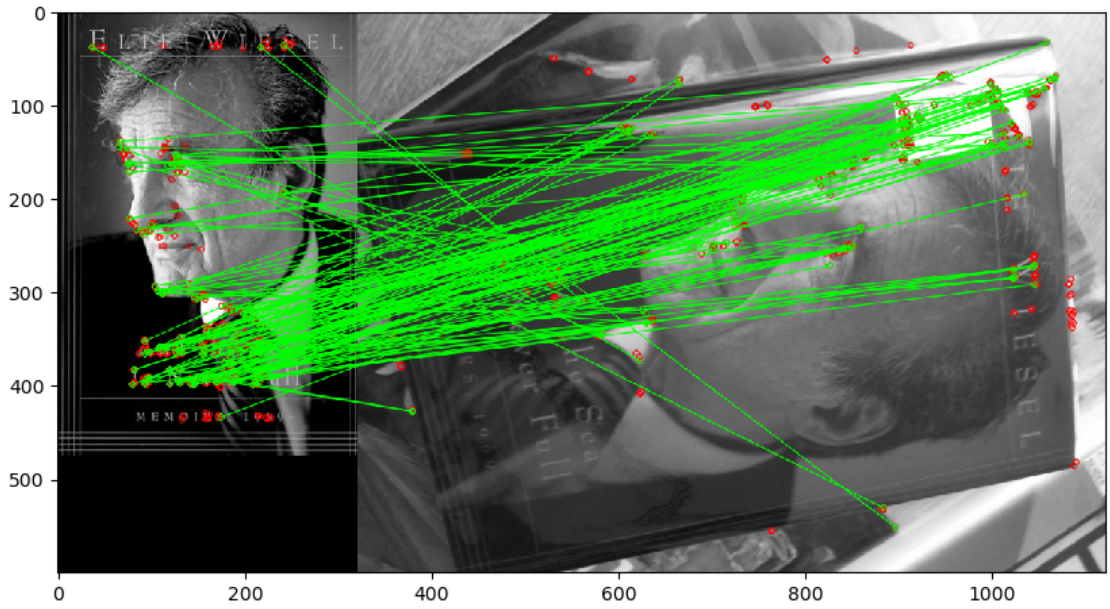



```
The number of inliers is: 116
```

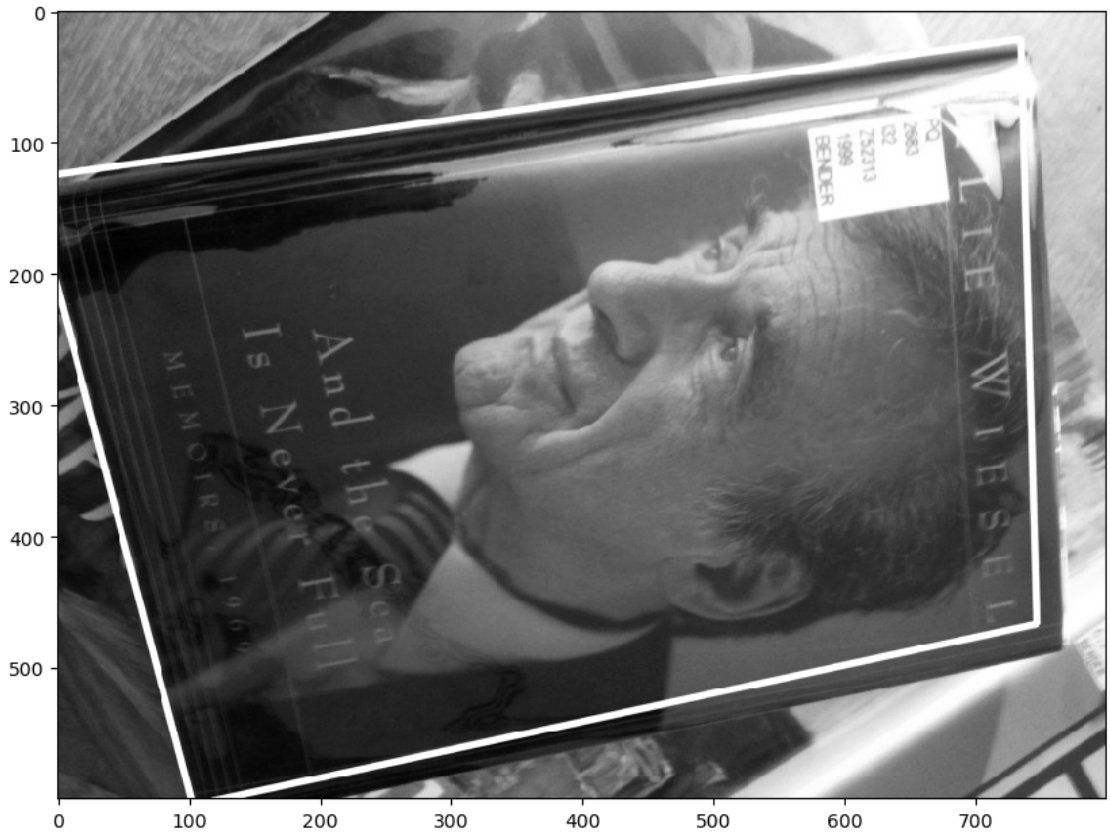
Your explanation of what you have tried, and results here

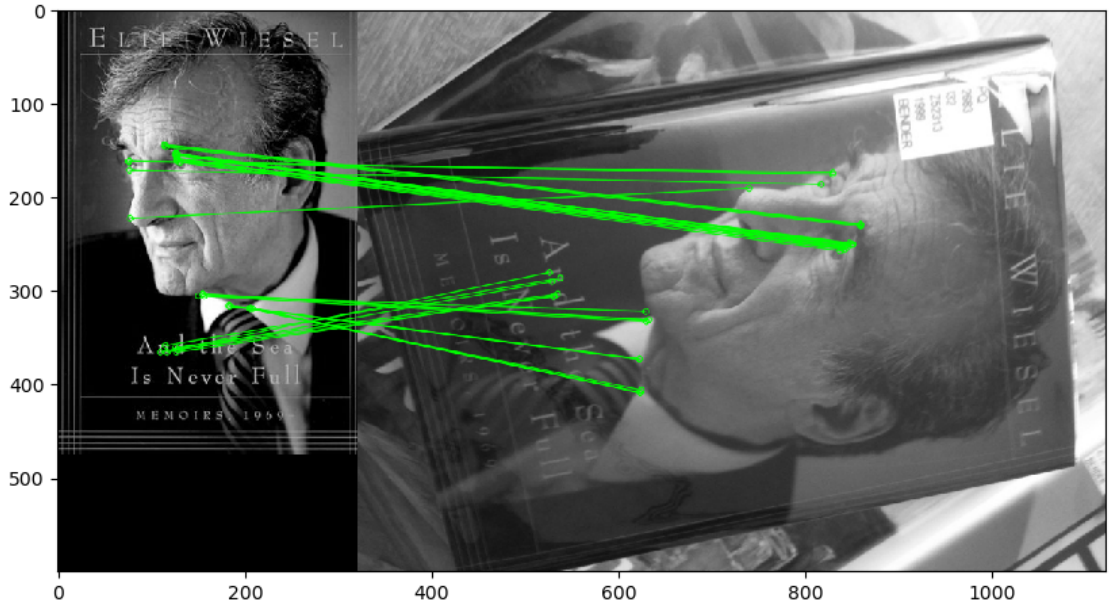
"findHomography()" takes in coordinates of points in the original and target plan as its first parameter. Then by specifying the method, it apply the chosen method for computing homography matrix. By setting the method to "cv.RANSAC", RansacReprojThreshold or maximum allowed reprojection error to treat a point pair as an inlier, is set by default of 3.0. I've tried to change this number and have noticed some change. As the threshold reduce, it also reduce the number of inliers as it requires the pairs to have less reprojections which means more exact required. Similarly, increasing the threshold closer to 10.0, also witness the increase in the number of inliers. For example, the number of inliers increase from 64 to 116 as we're increasing the threshold from 2.0 to 9.0.

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.



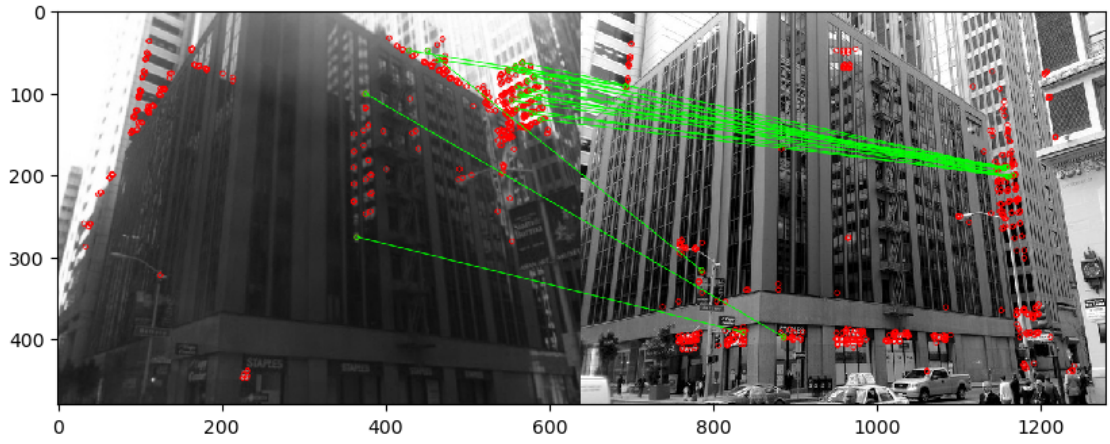
Using Ransac



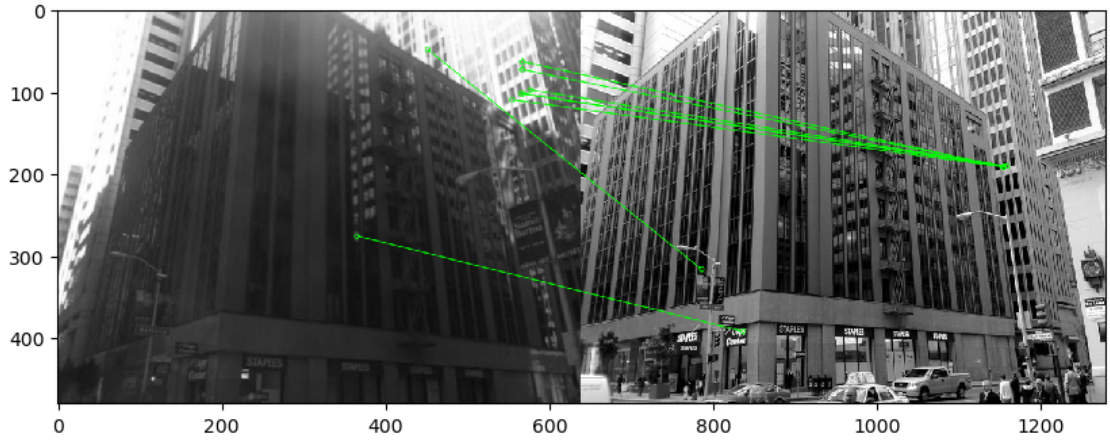


The number of inliers is: 36

LANDMARK



Using Ransac




```
The number of inliers is: 8
```

Your explanation of results here

For both test i've kept the good matches ratio to be 0.8 as well as 5.0 threshold for RANSAC method. Good matches's ratio is retained from the previous question as the ratio ensure a fair

number of matches while also to catch "good enough" pair. Beside that, keeping a consant ratio will also help me to make comparisions between book covers images and landmarks easier that help evaluate its efficiency. In the other hand, due to picking nearly extreme thresholds for testing in the previous part, within this part i want to pick a more "balance" threshold of 0.5 in order to obtain a good amount of matches but still expect a good quality from these matches.

SUCCESSFULL CASE:

SET UP: For the first pair of images, i've chosen the image of a book cover with a human depicted on it as the reference image, and another image from query that showing the book upside down. This is because book's cover with human's images and a large label provides ranges of different textures, pattern and could provide some challenges for matching, especially when the query image is rotated. However, given that book cover contains distinct visual elements and less affected by light's condition or roration, it is also expected to have multiple inliners as well as good matches.

After matching proccess, the algorithm performed well overall, as it have identified multiple quality keypoints that are particularly distinct and present a unique feature. However, in regions containing name of the book, there were instances of incorrect matches due to the blurring pixel and position of these characters. Despite this, the alogrithm still provide a good overview matches.

To improve the accuracy of matching, I adjusted the RANSAC threshold to 9.0 which resulted in the detection of additional true positive matches in the image. While getting additional false negative but it is considered acceptable

FAILURE CASE:

SET UP: The second pair of images depicts the corner of a building covered by glass windows. As glass windows create reflection depending on light source and there is a different in time,camera and angle of the two, these images pose a significant challenge for feature matching due their unconsistent and non distinctive nature. As result, it might not be distinctive enough for the algorithm to accurately detect and match keypoints. In addition, "false images" of other building causes by reflection within these pictures could potentially lead to the mismatch and confusion within ORB algorithm After matching proccess, the algorithm are able to identify multiple keypoints however as i've concerned, most of the detected keypoint belongs to the reflective images of other building through glass rather than the main building itself. Despite having some good keypoints along the edge of the building, the matches was fail between multiple points as well as wrong matches for example some higher windows was matching with lower windows that're not share the same column.

Feature detector identifies distinctive points or keypoints in the images, reflection and low contrast cause detecting keypoints to be difficult as the alogrithm confuses to distinct pixels. Similarly, light, viewpoint and reflection also affect feature descriptor as after keypoints are detected, the descriptor extracts information to represent their local appearance. To improve the accuracy of matching, I attempted to reduce the threshold fir RANSAC used in the matching process to 2.0 which result in eliminating additional false positive from the inliners (Number of inliners reduce from 10 to 8). Thus, by reducing the threshold the accuracy of matching increase.

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

```
Query image - 009  matches Reference image - 9 with matching score - 63
End step 1->3, starting step 4
```

```
Query image - 1  matches Reference image - 1 with matching score - 111
Query image - 2  matches Reference image - 2 with matching score - 64
Query image - 3  matches Reference image - 3 with matching score - 79
Query image - 4  matches Reference image - 4 with matching score - 30
Query image - 5  matches Reference image - 5 with matching score - 47
Query image - 6  matches Reference image - 6 with matching score - 94
Query image - 7  matches Reference image - 7 with matching score - 47
Not in dataset - Query image: 8
Query image - 9  matches Reference image - 9 with matching score - 63
Query image - 10  matches Reference image - 10 with matching score - 109
Query image - 11  matches Reference image - 11 with matching score - 84
Query image - 12  matches Reference image - 12 with matching score - 52
Query image - 13  matches Reference image - 13 with matching score - 62
Query image - 14  matches Reference image - 14 with matching score - 65
Query image - 15  matches Reference image - 15 with matching score - 61
Query image - 16  matches Reference image - 16 with matching score - 31
Query image - 17  matches Reference image - 17 with matching score - 59
Not in dataset - Query image: 18
Query image - 19  matches Reference image - 19 with matching score - 26
Query image - 20  matches Reference image - 20 with matching score - 117
Query image - 21  matches Reference image - 21 with matching score - 53
Query image - 22  matches Reference image - 22 with matching score - 20
Query image - 23  matches Reference image - 23 with matching score - 44
Query image - 24  matches Reference image - 24 with matching score - 55
Query image - 25  matches Reference image - 25 with matching score - 70
Query image - 26  matches Reference image - 26 with matching score - 47
Query image - 27  matches Reference image - 27 with matching score - 50
Query image - 28  matches Reference image - 28 with matching score - 109
Query image - 29  matches Reference image - 29 with matching score - 37
Query image - 30  matches Reference image - 30 with matching score - 91
Query image - 31  matches Reference image - 31 with matching score - 59
Query image - 32  matches Reference image - 32 with matching score - 64
Not in dataset - Query image: 33
Query image - 34  matches Reference image - 34 with matching score - 76
Query image - 35  matches Reference image - 35 with matching score - 33
Query image - 36  matches Reference image - 36 with matching score - 28
Not in dataset - Query image: 37
Query image - 38  matches Reference image - 38 with matching score - 30
Query image - 39  matches Reference image - 39 with matching score - 88
Query image - 40  matches Reference image - 40 with matching score - 93
The success rate is 90.0%
```

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

Starting by creating subset of Query Image and their matching Reference Image. Then,Similar to the previous question, for the step 1->3, i've repeated the same proccess. Starting from choosing Reference and Query image 9 for testing out the number of inliers. Using ORB detector, i've detected keypoints descriptors for both of the images while keeping the old ratio of 0.8 (As from previous question, i think 0.8 is a good ratio to obtain a fair amount of matches with good quality).

I've also used brute force matcher that specify Hamming distance to decide whether a match is a good match. Then RANSAC had been used to eliminate any outliers and obtaining inliers. Then by implying function "matching_score" that take in Reference image and Query image as parameters, i've successfully traversed the chosen query image - image 009 over the first 40 images and counting numbers of inliers as matching score for each of them. Afterward, I've picked out the image with the highest matching score (or highest number of inliers) as the query object. NOTED: The code also eliminate any image with 4 or less good matches as it's not enough to run RANSAC and assign the matching score to be 0.

Moving to step 4, i picked the threshold matching score to be 20 as obtaining less than 20 inliers is considered unsuccessful match, aiming to balance between false positives and false negatives. After running the loop that match every query images with every reference images, then by counting success cases (higher than 20) i've achieved 90% of accuracy which is reasonable for identifying book covers's task. In cases of failure,the actual match receives the lowest possible ranking among all other matches which due to none of reference image achieve a high enough matching score to be considered successfull. Top-k accuracy measures could potentially affect the result but not by much. As method provides additional successful matches, it could increase the matching score slightly by considering these extra matches.

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.

```
Query image - 1  matches Reference image - 1 with matching score - 111
Query image - 2  matches Reference image - 2 with matching score - 64
Query image - 3  matches Reference image - 3 with matching score - 79
Query image - 4  matches Reference image - 4 with matching score - 30
Query image - 5  matches Reference image - 5 with matching score - 47
Query image - 6  matches Reference image - 6 with matching score - 94
Query image - 7  matches Reference image - 7 with matching score - 47
Not in dataset - Query image: 8
Query image - 9  matches Reference image - 9 with matching score - 63
Query image - 10  matches Reference image - 10 with matching score - 109
Query image - 11  matches Reference image - 11 with matching score - 84
Query image - 12  matches Reference image - 12 with matching score - 52
Query image - 13  matches Reference image - 13 with matching score - 62
Query image - 14  matches Reference image - 14 with matching score - 65
Query image - 15  matches Reference image - 15 with matching score - 61
Query image - 16  matches Reference image - 16 with matching score - 31
Query image - 17  matches Reference image - 17 with matching score - 59
Not in dataset - Query image: 18
Query image - 19  matches Reference image - 19 with matching score - 26
Query image - 20  matches Reference image - 20 with matching score - 117
Query image - 21  matches Reference image - 21 with matching score - 53
Query image - 22  matches Reference image - 22 with matching score - 20
Query image - 23  matches Reference image - 23 with matching score - 44
Query image - 24  matches Reference image - 24 with matching score - 55
```

```
Query image - 25  matches Reference image - 25 with matching score - 70
Query image - 26  matches Reference image - 26 with matching score - 47
Query image - 27  matches Reference image - 27 with matching score - 50
Query image - 28  matches Reference image - 28 with matching score - 109
Query image - 29  matches Reference image - 29 with matching score - 37
Query image - 30  matches Reference image - 30 with matching score - 91
Query image - 31  matches Reference image - 31 with matching score - 59
Query image - 32  matches Reference image - 32 with matching score - 64
Not in dataset - Query image: 33
Query image - 34  matches Reference image - 34 with matching score - 76
Query image - 35  matches Reference image - 35 with matching score - 33
Query image - 36  matches Reference image - 36 with matching score - 28
Not in dataset - Query image: 37
Query image - 38  matches Reference image - 38 with matching score - 30
Query image - 39  matches Reference image - 39 with matching score - 88
Query image - 40  matches Reference image - 40 with matching score - 93
Not in dataset - Query image: 41
Query image - 42  matches Reference image - 33 with matching score - 21
Not in dataset - Query image: 43
Not in dataset - Query image: 44
Not in dataset - Query image: 45
Accuracy measured by correctly identified as not occurring in the dataset. 80.0%
```

Your explanation of results and any changes made here

Starting step 5 by adding extra 5 images that does not have corresponding reference image in "ref_img" img, i discover that out 5 image, the program is able to identify 4 of them that does not in the dataset which achieve the accuracy of 80%. To improve this, i'd have to set a higher threshold (i've tried threshold of 25), however it's will also led to the increase of false negatives cases where valid matches are incorrectly labeled as not in the dataset that reduce the accuracy calculated in step 4.

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.

```
Query image - 1  matches Reference image - 1 with matching score - 48
Not in dataset - Query image: 2
Not in dataset - Query image: 3
Not in dataset - Query image: 4
Not in dataset - Query image: 5
Query image - 6  matches Reference image - 6 with matching score - 22
Query image - 7  matches Reference image - 7 with matching score - 58
Query image - 8  matches Reference image - 8 with matching score - 126
Not in dataset - Query image: 9
Query image - 10  matches Reference image - 10 with matching score - 34
Query image - 11  matches Reference image - 11 with matching score - 31
Query image - 12  matches Reference image - 12 with matching score - 81
```

```
Not in dataset - Query image: 13
Query image - 14  matches Reference image - 14 with matching score - 36
Not in dataset - Query image: 15
Query image - 16  matches Reference image - 16 with matching score - 28
Query image - 17  matches Reference image - 17 with matching score - 107
Query image - 18  matches Reference image - 18 with matching score - 115
Query image - 19  matches Reference image - 19 with matching score - 40
Query image - 20  matches Reference image - 20 with matching score - 36
Query image - 21  matches Reference image - 21 with matching score - 21
Not in dataset - Query image: 22
Not in dataset - Query image: 23
Not in dataset - Query image: 24
Not in dataset - Query image: 25
Not in dataset - Query image: 26
Not in dataset - Query image: 27
Not in dataset - Query image: 28
Query image - 29  matches Reference image - 29 with matching score - 77
Not in dataset - Query image: 30
Not in dataset - Query image: 31
Query image - 32  matches Reference image - 32 with matching score - 40
Query image - 33  matches Reference image - 33 with matching score - 43
Not in dataset - Query image: 34
Not in dataset - Query image: 35
Not in dataset - Query image: 36
Not in dataset - Query image: 37
Not in dataset - Query image: 38
Not in dataset - Query image: 39
Not in dataset - Query image: 40
The success rate is 42.5%


Testing with modified threshold
Query image - 1  matches Reference image - 1 with matching score - 48
Not in dataset - Query image: 2
Not in dataset - Query image: 3
Not in dataset - Query image: 4
Not in dataset - Query image: 5
Query image - 6  matches Reference image - 6 with matching score - 22
Query image - 7  matches Reference image - 7 with matching score - 58
Query image - 8  matches Reference image - 8 with matching score - 126
Not in dataset - Query image: 9
Query image - 10  matches Reference image - 10 with matching score - 34
Query image - 11  matches Reference image - 11 with matching score - 31
Query image - 12  matches Reference image - 12 with matching score - 81
Not in dataset - Query image: 13
Query image - 14  matches Reference image - 14 with matching score - 36
Not in dataset - Query image: 15
Query image - 16  matches Reference image - 16 with matching score - 28
```

```
Query image - 17  matches Reference image - 17 with matching score - 107
Query image - 18  matches Reference image - 18 with matching score - 115
Query image - 19  matches Reference image - 19 with matching score - 40
Query image - 20  matches Reference image - 20 with matching score - 36
Query image - 21  matches Reference image - 21 with matching score - 21
Not in dataset - Query image: 22
Not in dataset - Query image: 23
Not in dataset - Query image: 24
Not in dataset - Query image: 25
Not in dataset - Query image: 26
Not in dataset - Query image: 27
Not in dataset - Query image: 28
Query image - 29  matches Reference image - 29 with matching score - 77
Not in dataset - Query image: 30
Not in dataset - Query image: 31
Query image - 32  matches Reference image - 32 with matching score - 40
Query image - 33  matches Reference image - 33 with matching score - 43
Not in dataset - Query image: 34
Not in dataset - Query image: 35
Not in dataset - Query image: 36
Not in dataset - Query image: 37
Not in dataset - Query image: 38
Not in dataset - Query image: 39
Not in dataset - Query image: 40
The success rate is 85.0%
```

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

I repeat the same procedures for museum painting and getting the accuracy of 42.5% as result. This is significantly lower tahn the 90% accuracy achieved with the previous set of reference book covers images. Despite having some Query images matches Reference image, a significant number of query images were labeled as "Not in dataset", indicating that the algorithm failed to identify corresponding reference images for them. After going to more detail by asking the "Not in dataset" image to print its matching score, i observe potential issues with the matching algorithm's performance as museum painting or landmark(from previous question) are challenging objects for feature detector, descriptor as well as matching process due to their lack of distinct, unique keypoints. Thus, taking into account these challenges, i've decided to reduce the threshold from 20 to 18 for determining successfull matching, since the current threshold could be too high for "hard-for-matching" objects. By reducing the threshold by 2, i'd expect it to capture more matches without significantly increasing false positives. As result, after changing the threshold, the accuracy was jumping from 42.5% to 85% which show the effectiveness of threshold changing.