Assignment2

May 4, 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"}'
```

Please do try this ccommond early before the last day! As the commend may be a little bit difference depending on your computer and the environment.

This will also remove the preamble text from each question. We will use the `OpenCV` library to complete the prac. It has several built in functions that will be useful. You are expected to consult documentation and use them appropriately.

This being the second assignment, we have provided less strict direction this time and you have more flexibility to choose how you answer each question. However you still need to ensure the outputs and report are clear and easy to read. This includes:

- sizing, arranging and captioning image outputs appropriately
- explaining what you have done clearly and concisely
- clearly separating answers to each question

1.2 Data

We have provided some example images for this assignment, available through a link on the MyUni assignment page. The images are organised by subject matter, with one folder containing images of book covers, one of museum exhibits, and another of urban landmarks. Within each category, there is a "Reference" folder containing a clean image of each object and a "Query" folder containing images taken on a mobile device. Within each category, images with the same name contain the same object (so 001.jpg in the Reference folder contains the same book as 001.jpg in the Query folder). The data is a subset of the Stanford Mobile Visual Search Dataset which is available at

http://web.cs.wpi.edu/~claypool/mmsys-dataset/2011/stanford/index.html.

The full data set contains more image categories and more query images of the objects we have provided, which may be useful for your testing!

Do not submit your own copy of the data or rename any files or folders! For marking, we will assume the datasets are available in subfolders of the working directory using the same folder names provided.

Here is some general setup code, which you can edit to suit your needs.

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:

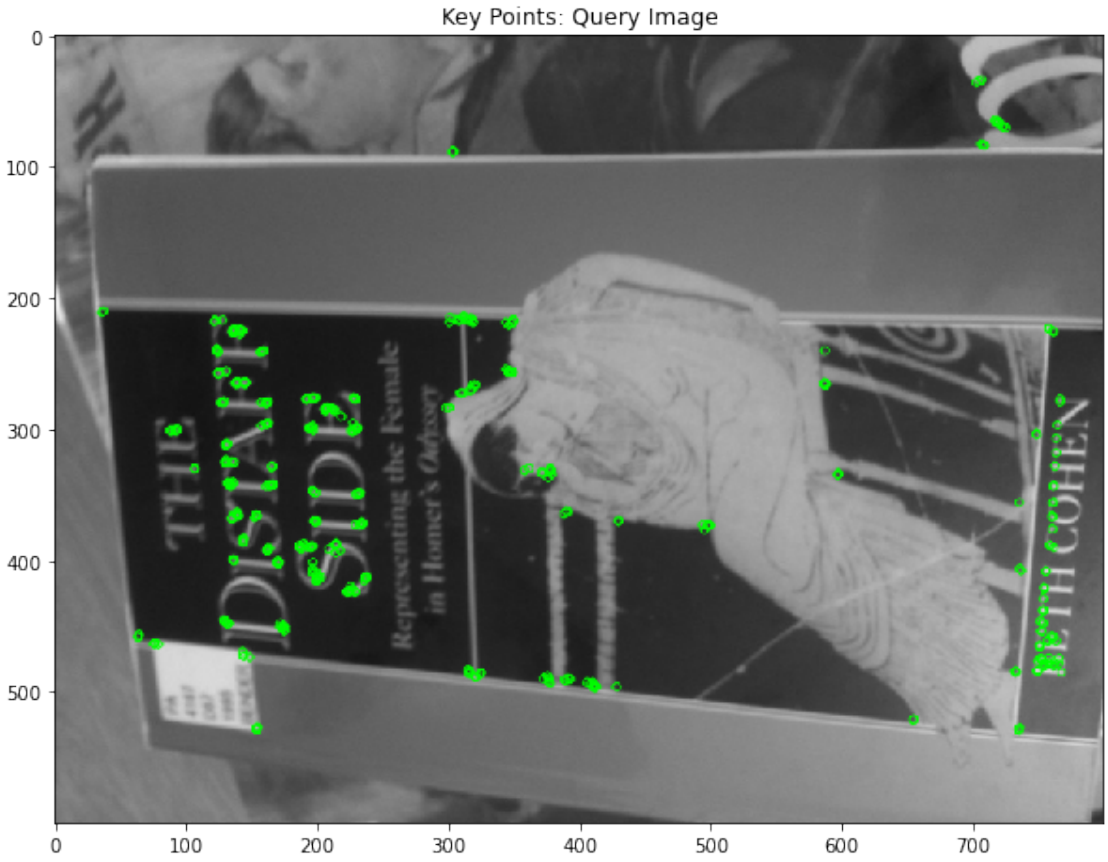
0. Download and read through the paper ORB: an efficient alternative to SIFT or SURF by Rublee et al. You don't need to understand all the details, but try to get an idea of how it works. ORB combines the FAST corner detector (covered in week 4) and the BRIEF descriptor. BRIEF is based on similar ideas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

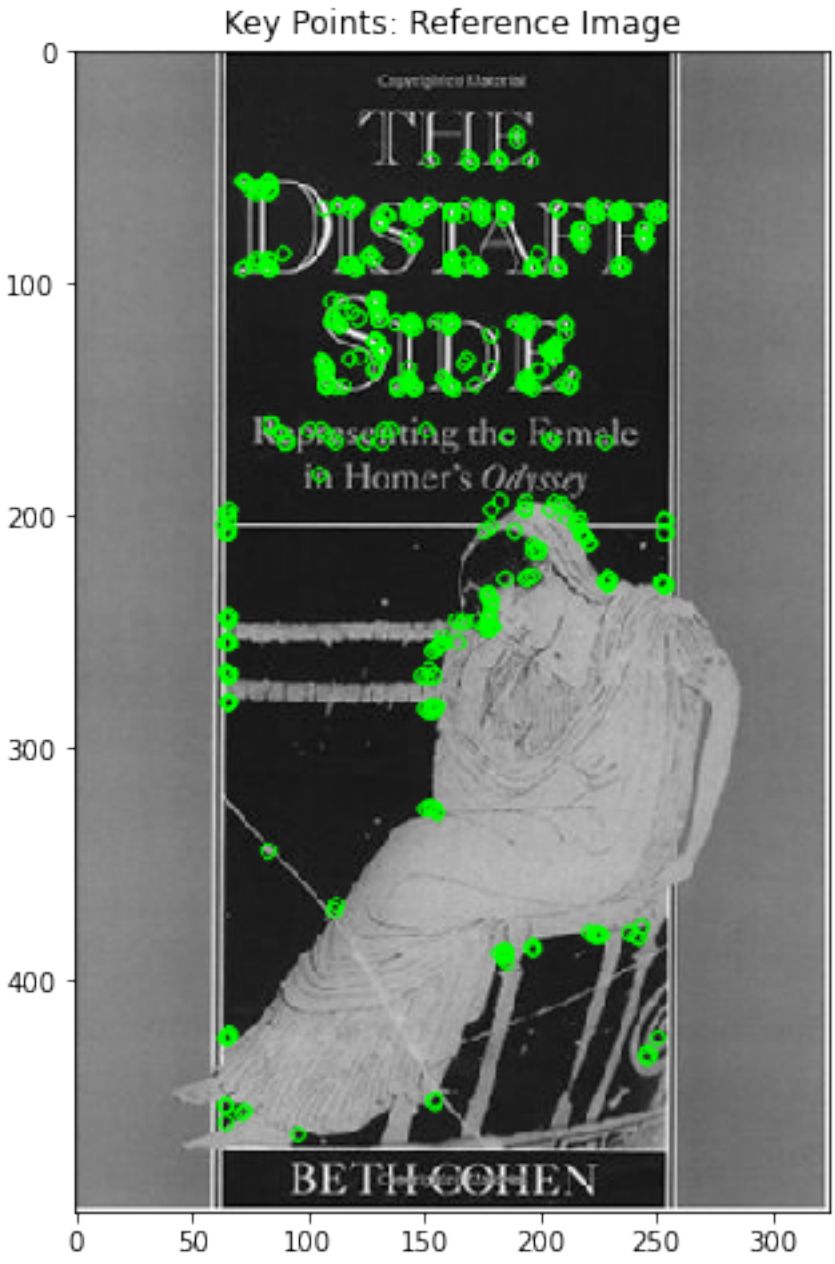
1. [Load images] Load the first (reference, query) image pair from the "book_covers" category using opencv (e.g. `img=cv2.imread()`). Check the parameter option in " cv2.imread()" to ensure that you read the gray scale image, since it is necessary for computing ORB features.

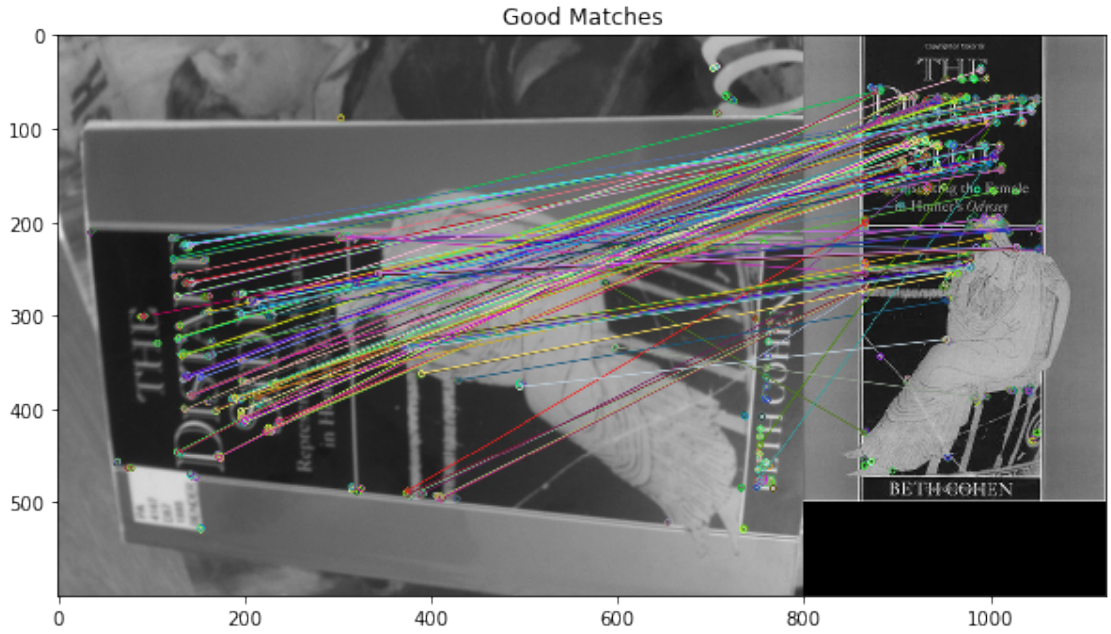
2. [Detect features] Create opencv ORB feature extractor by `orb=cv2.ORB_create()`. Then you can detect keypoints by `kp = orb.detect(img,None)`, and compute descriptors by `kp, des = orb.compute(img, kp)`. You need to do this for each image, and then you can use `cv2.drawKeypoints()` for visualization.

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`.

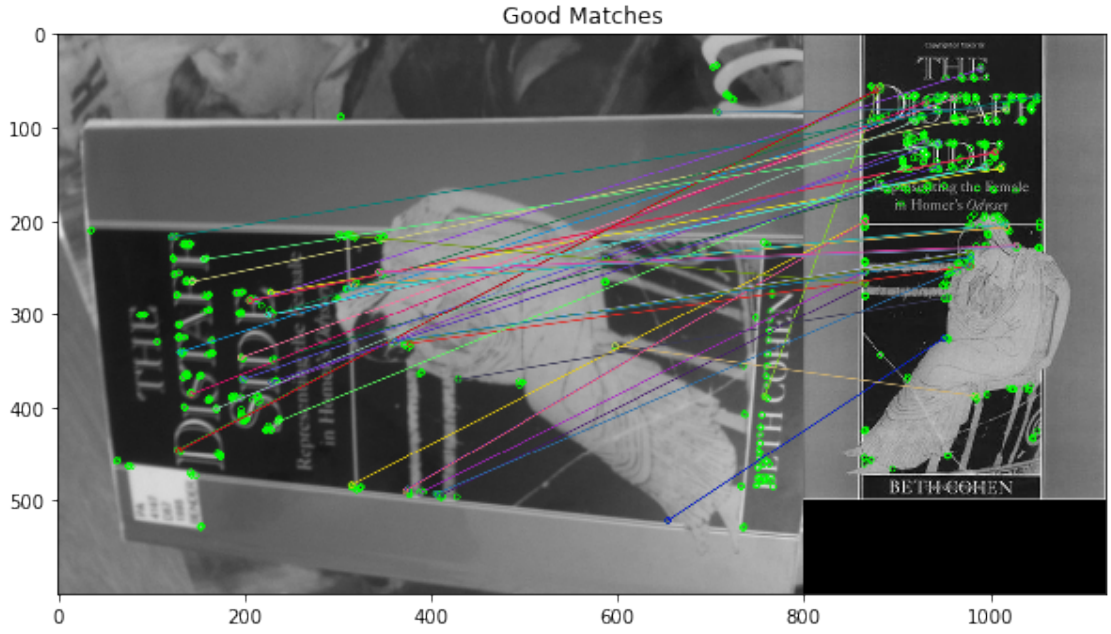
4. [Plot and analyze] You need to visualize the matches by using the `cv2.drawMatches()` function. Also you can change the ratio values, parameters in `cv2.ORB_create()`, and distance functions in `cv2.BFMatcher()`. Please discuss how these changes influence the match numbers.


Key Points: Query Image
0
100
200
300
400
500
0
100
200
300
400
500
600
700
THE
DISTANT
SIDE
Representing the Female
in Homer's Odyssey
BETH COHEN


Key Points: Reference Image
THE
DISTANT
SIDE
Representing the Female
in Homer's Odyssey
BETH COHEN
0
100
200
300
400
0
50
100
150
200
250
300



NUMBER OF GOOD MATCHES: 130



NUMBER OF GOOD MATCHES: 46

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

Change in the parameters:

1: When we change the ratio value in the ratio test, the numbers of matches change. If we change the ratio value to 0.4, the number of matches is less than half of the original number, which is understandable as with 0.4 (strong filtering)less matches are being considered as good matches.

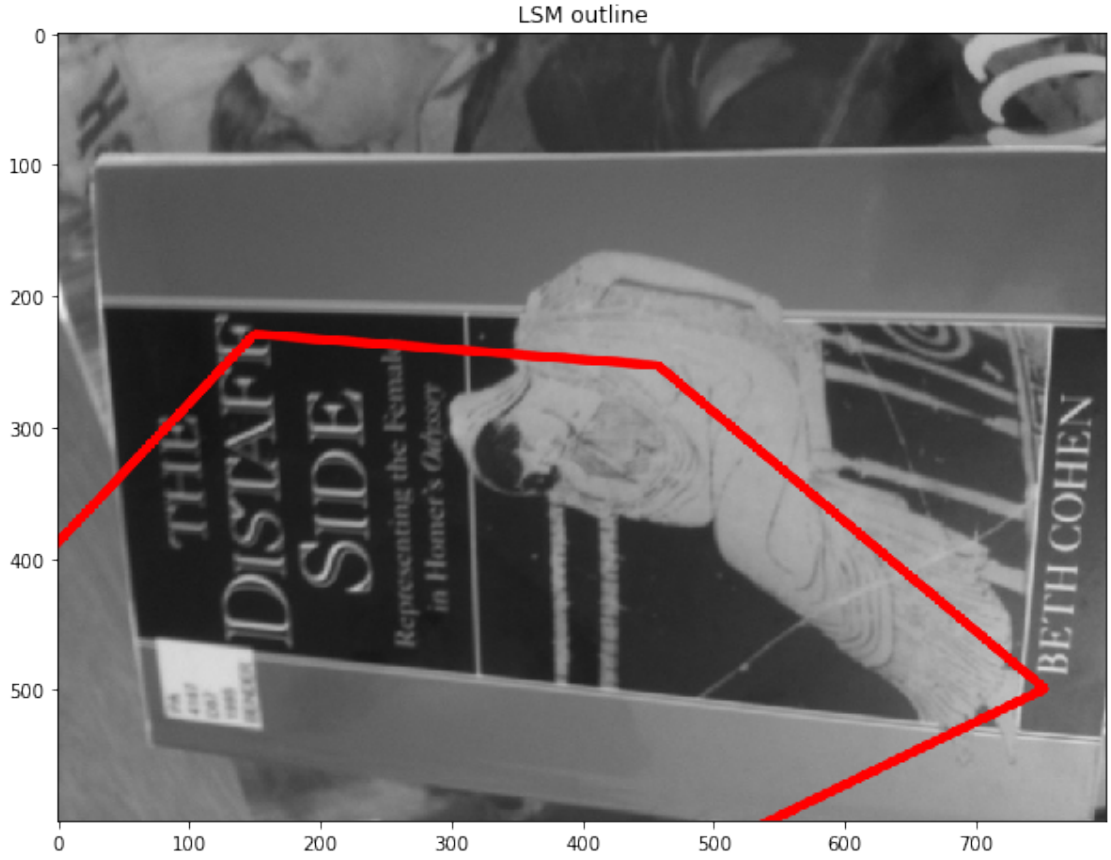
2: When we change the number of features in the ORB detector keeping the ratio in the ratio test constant, there is a change observed in the number of good matches. If we change the number of features to 100, the number of matches is less than half of the original number. And if we change the number of features to 1000, the number of matches is greater than the orginal number.

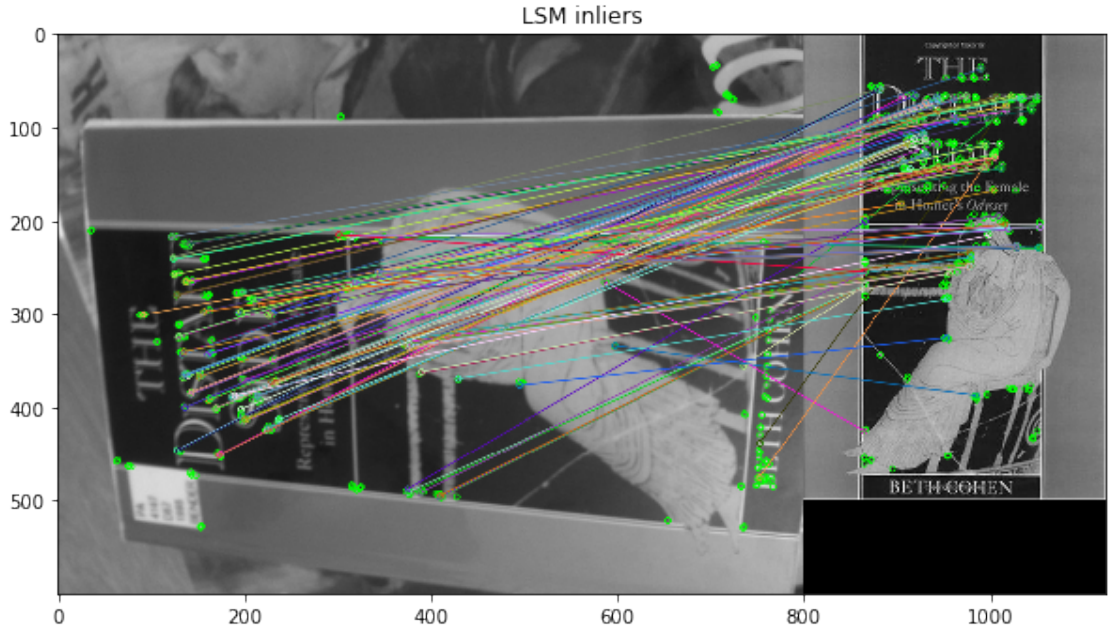
3: In the cv2.BFMatcher(), if we change the distance function from cv2.NORM_HAMMING to cv2.NORM_L2, the number of matches is less than half of the original number. With NORM_HAMMING, the ratio test is working well and giving us 130 matches. However, when the norm is set to NORM_L2, the ratio test is not working well and giving us only 46 matches, this is because the NORM_L2 is a good option for the SHIFT and SURF methods but not for binary string based methods like ORB.

Also, if we put cross check in the cv2.BFMatcher(), the code gives an error message: "OpenCV(4.5.4) ../modules/core/src/batch_distance.cpp:303: error: (-215:Assertion failed) K == 1 && update == 0 && mask.empty() in function 'batchDistance' ".

ref: https://docs.opencv.org/3.4/dc/dc3/tutorial_py_matcher.html

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.


LSM outline
0
100
200
300
400
500
0
100
200
300
400
500
600
700
BETH COHEN


LSM inliers
0
100
200
300
400
500
0
200
400
600
800
1000
BETH COHEN

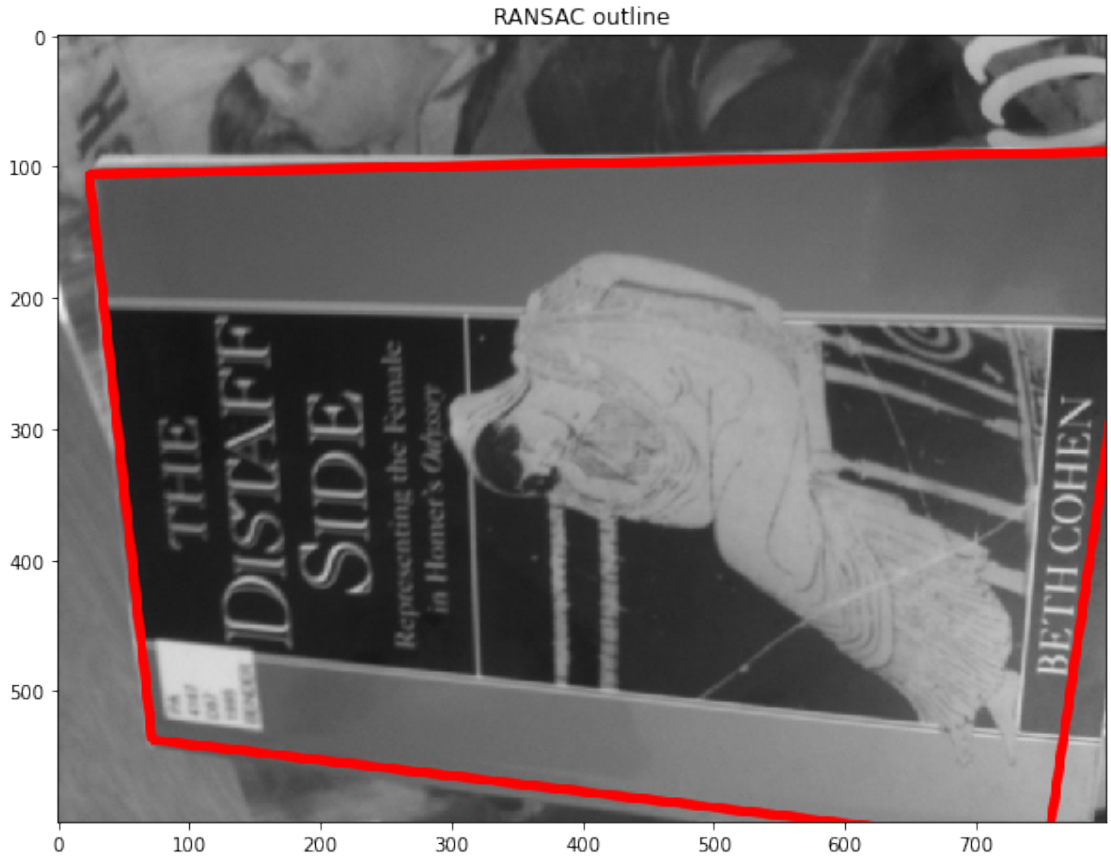
```
LSM inliers:  130
```

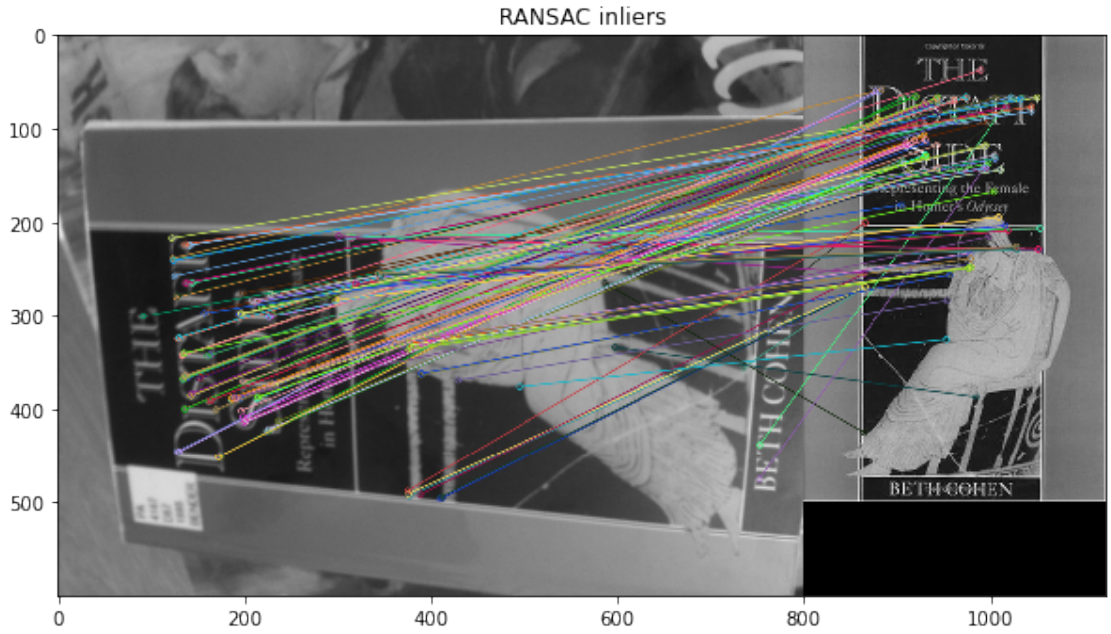
Your explanation of results here

We see that with the give paramenters/good matches, the Least Squared Method is not very good at giving out an accurate outline of the reference image on the query image. This may be because this method uses all the points availabe and hence be affected by the outliers.

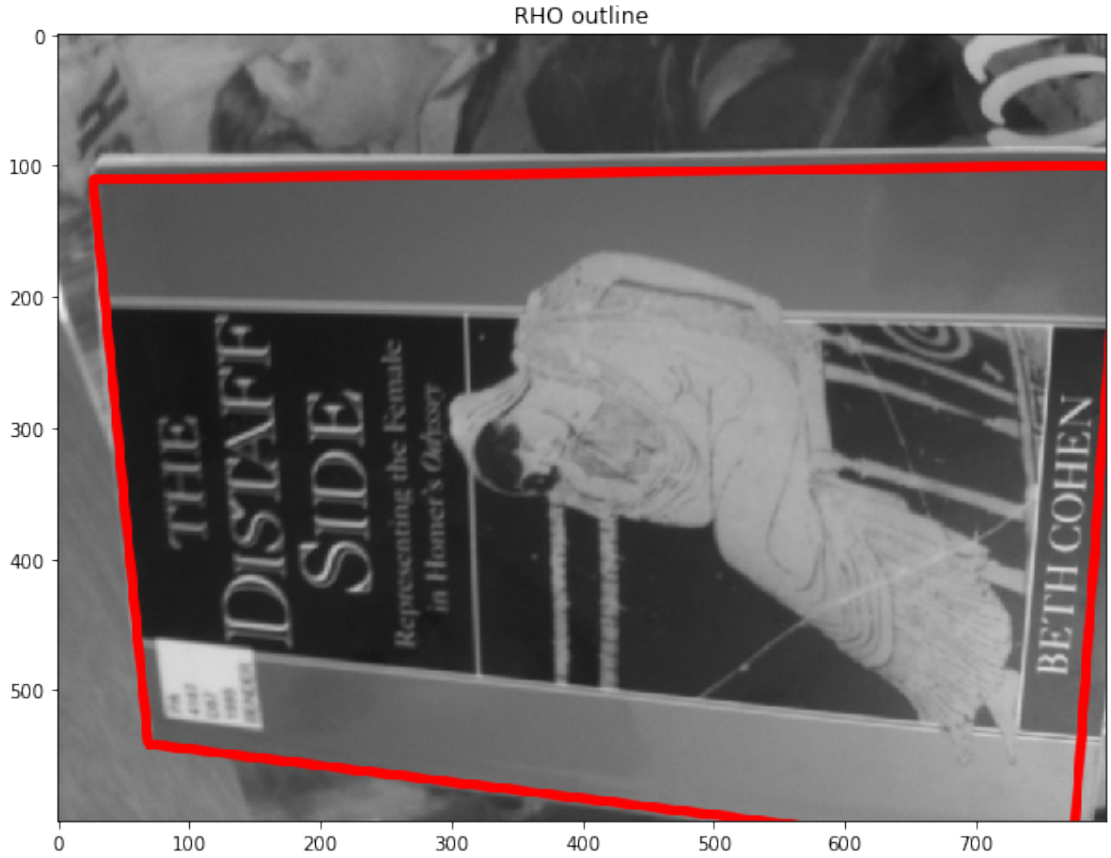
The RANSAC and the RHO methods are good at giving an accurate outline of the reference image on the query image, as they are good at handling outliers.

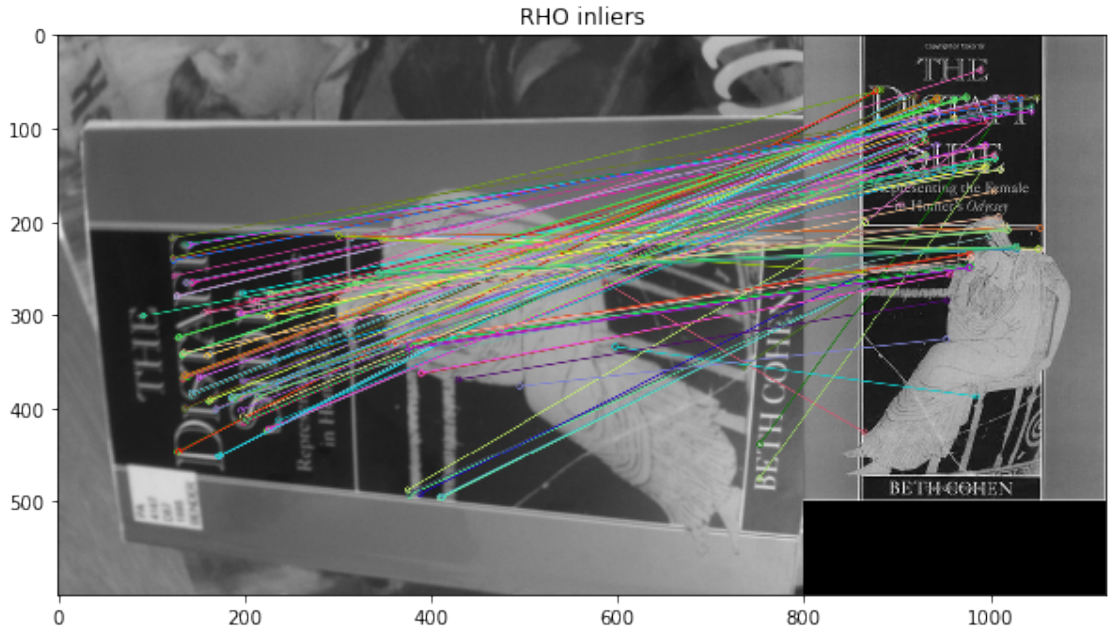
Try the RANSAC option to compute homography. Change the RANSAC parameters, and explain your results. Print and analyze the inlier numbers. Hint: use cv2.RANSCA with cv2.findHomography.






```
RANSAC inliners:  130
```



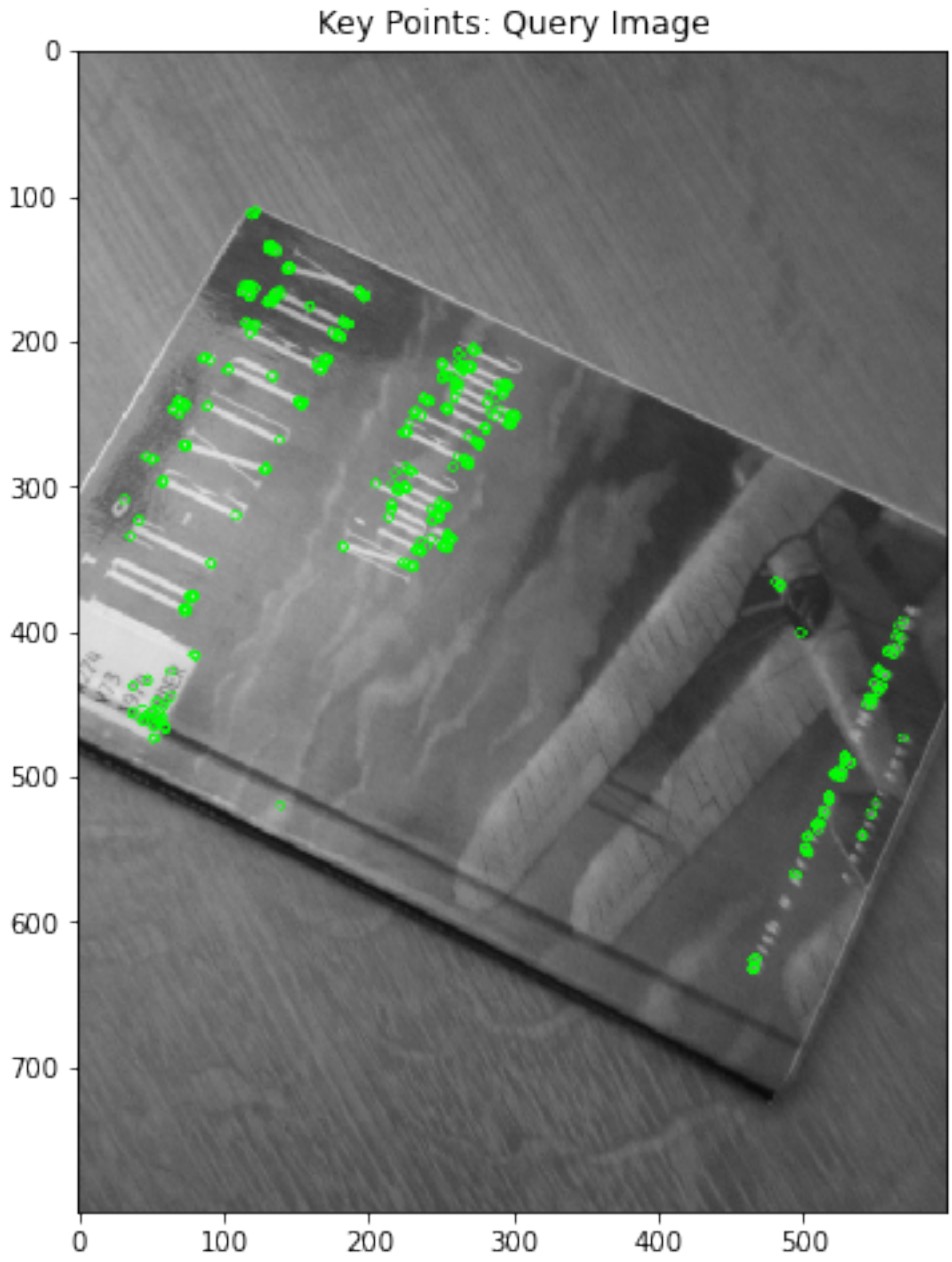


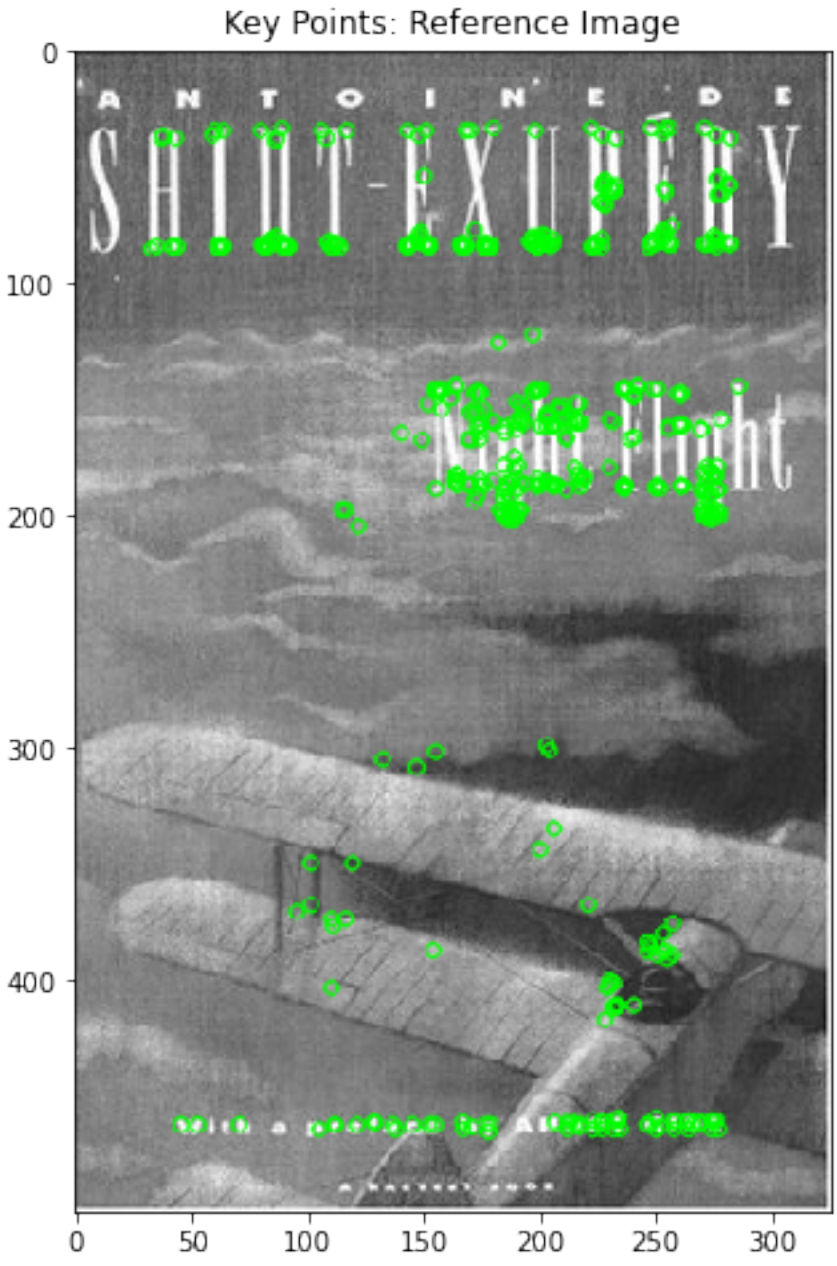

```
RHO inliers:  130
```

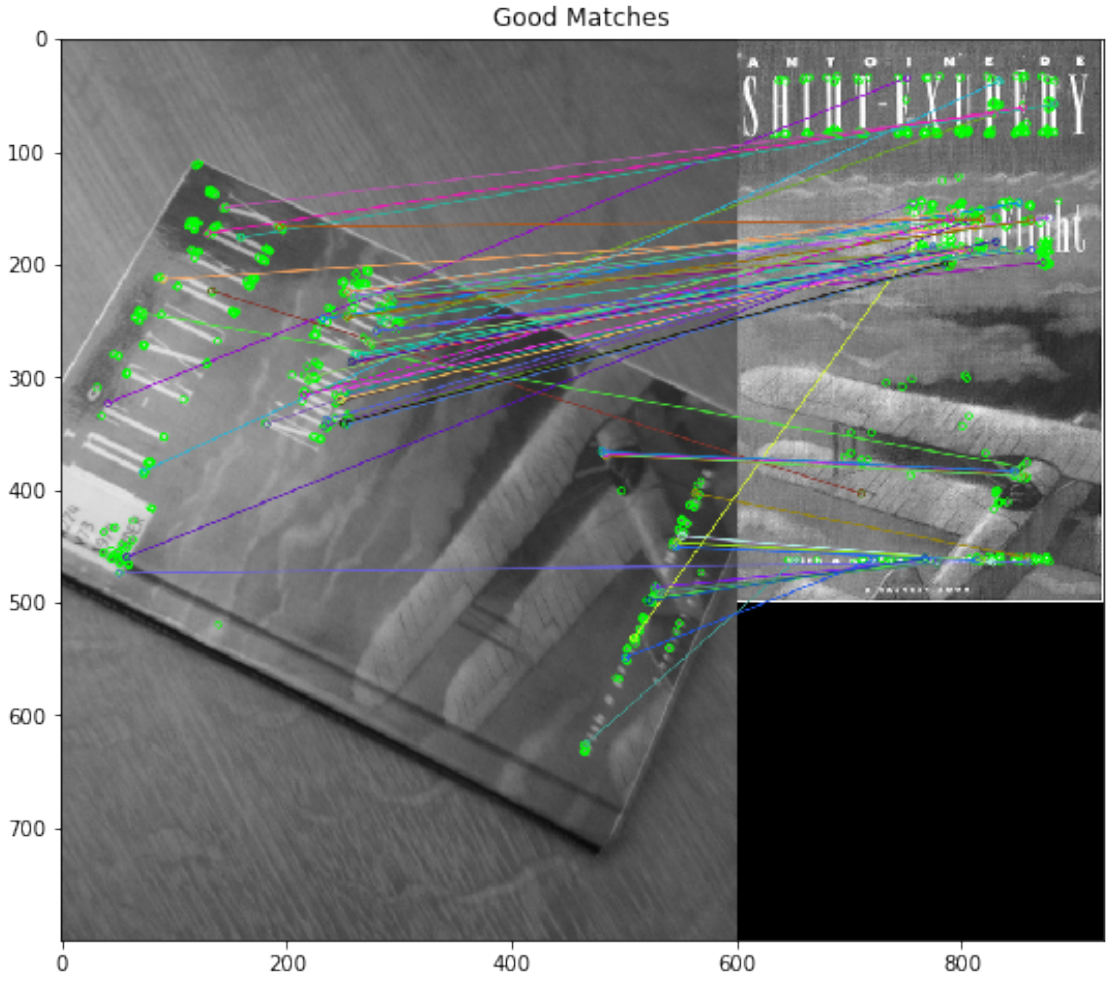
Your explanation of what you have tried, and results here

The RANSAC method gave a good result with the same parameters. The outline of the reference image on the query image is very good. The RANSAC is good at handling outliers therefore it gave us a good result. The same can be said for the RHO method.

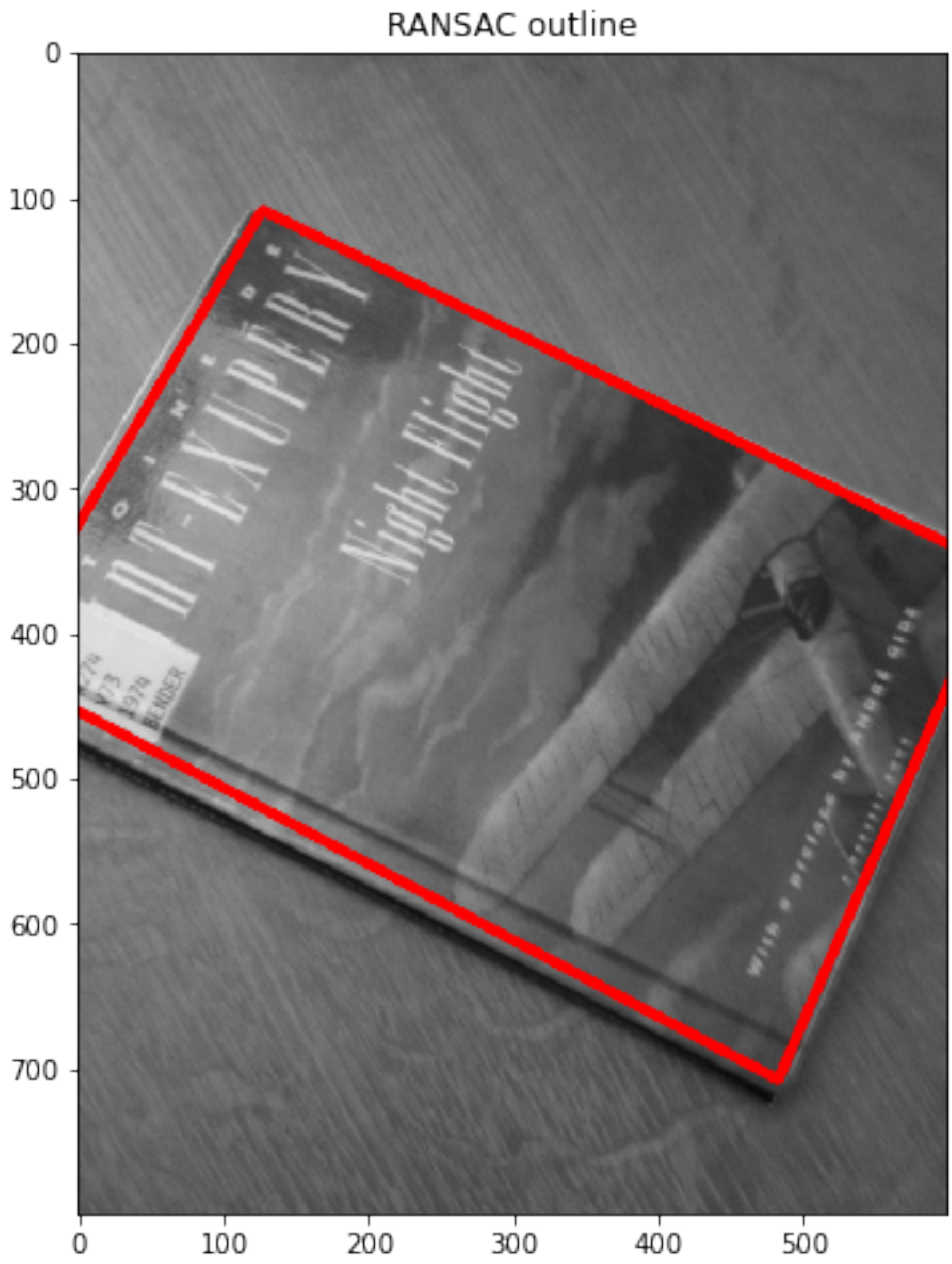
6. Finally, try matching several different image pairs from the data provided, including at least one success and one failure case. For the failure case, test and explain what step in the feature matching has failed, and try to improve it. Display and discuss your findings.
    1. Hint 1: In general, 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.


Key Points: Query Image
0
100
200
300
400
500
600
700
0
100
200
300
400
500


Key Points: Reference Image
0
100
200
300
400
0
50
100
150
200
250
300



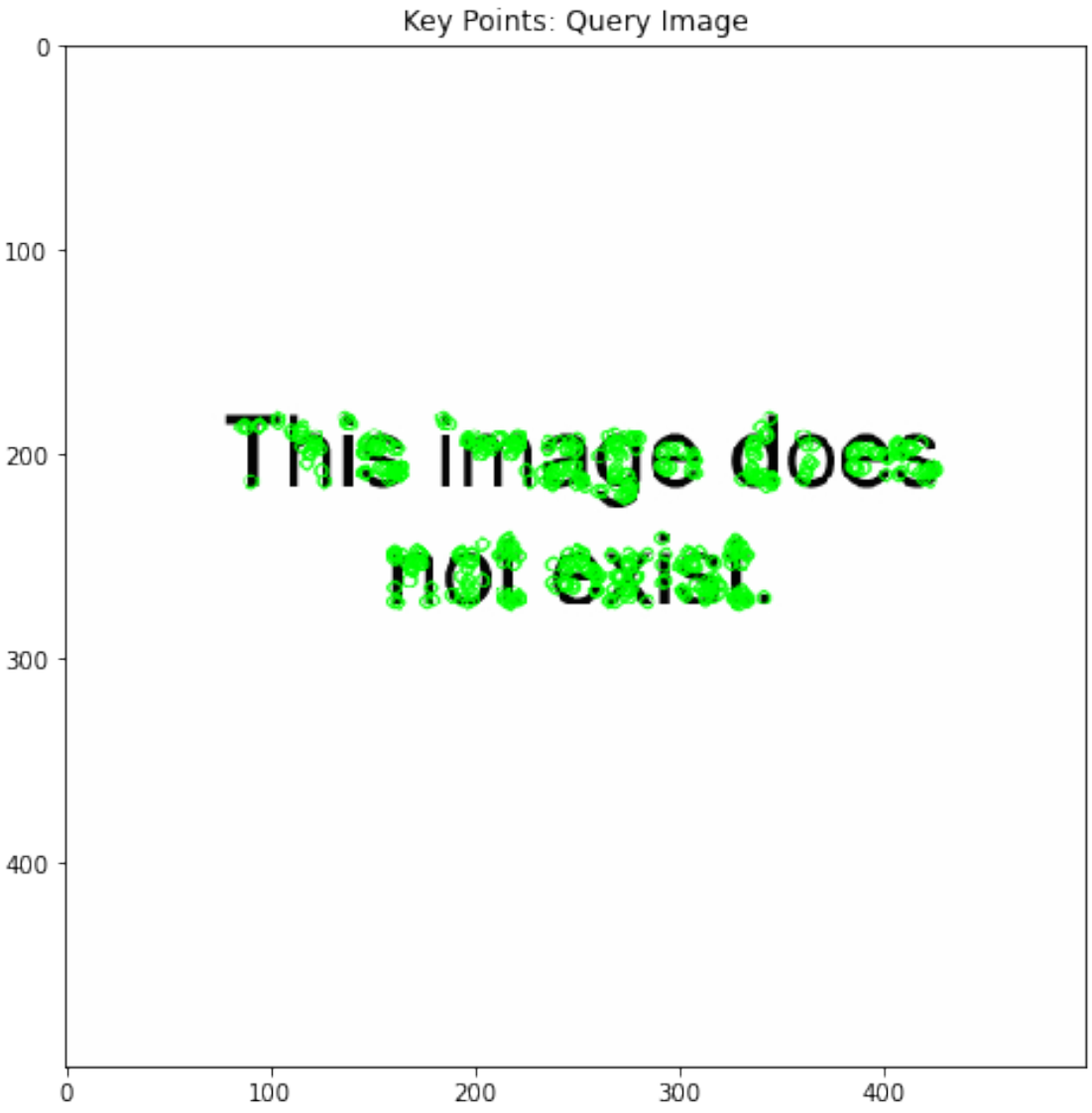

```
NUMBER OF GOOD MATCHES:  69
```

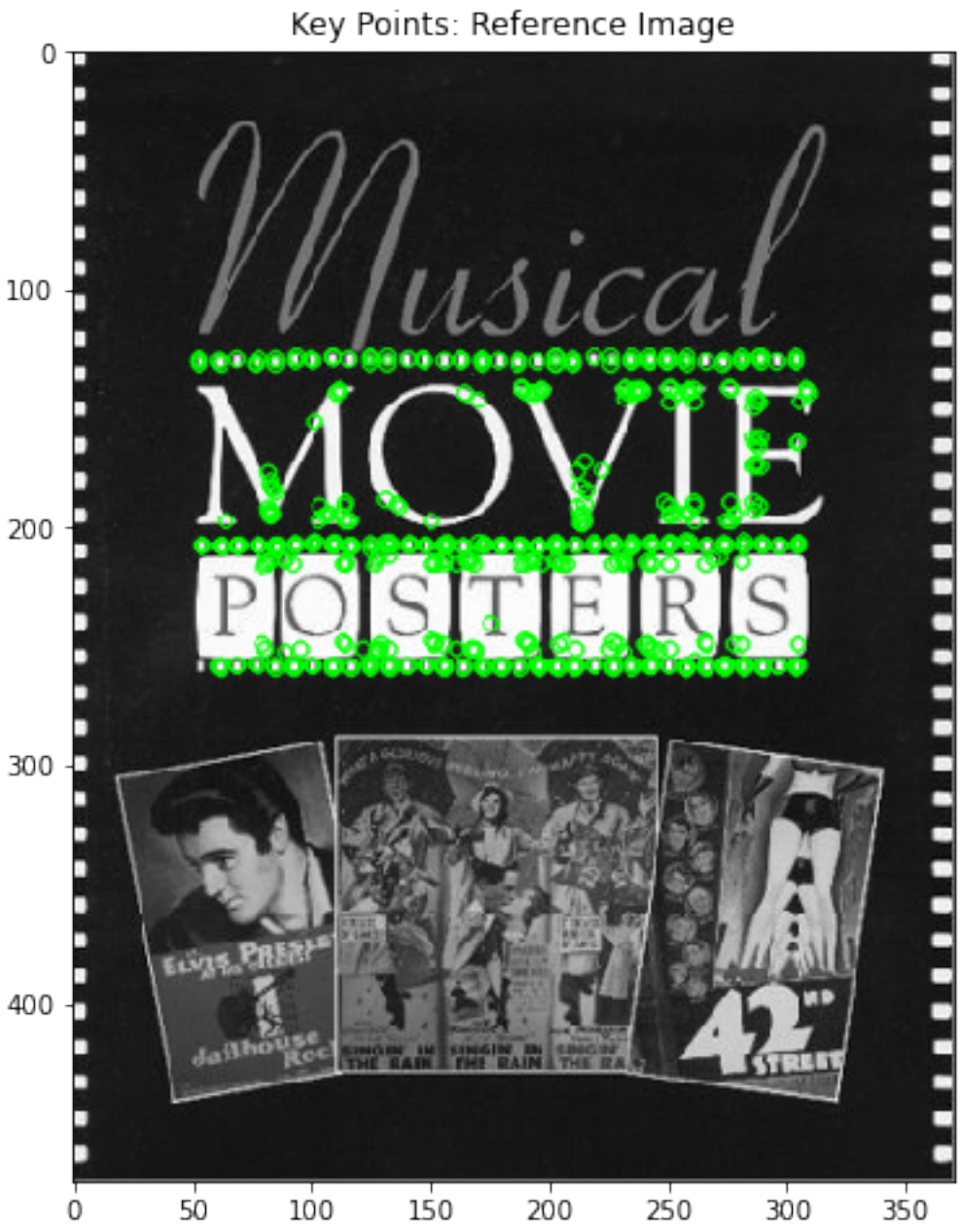


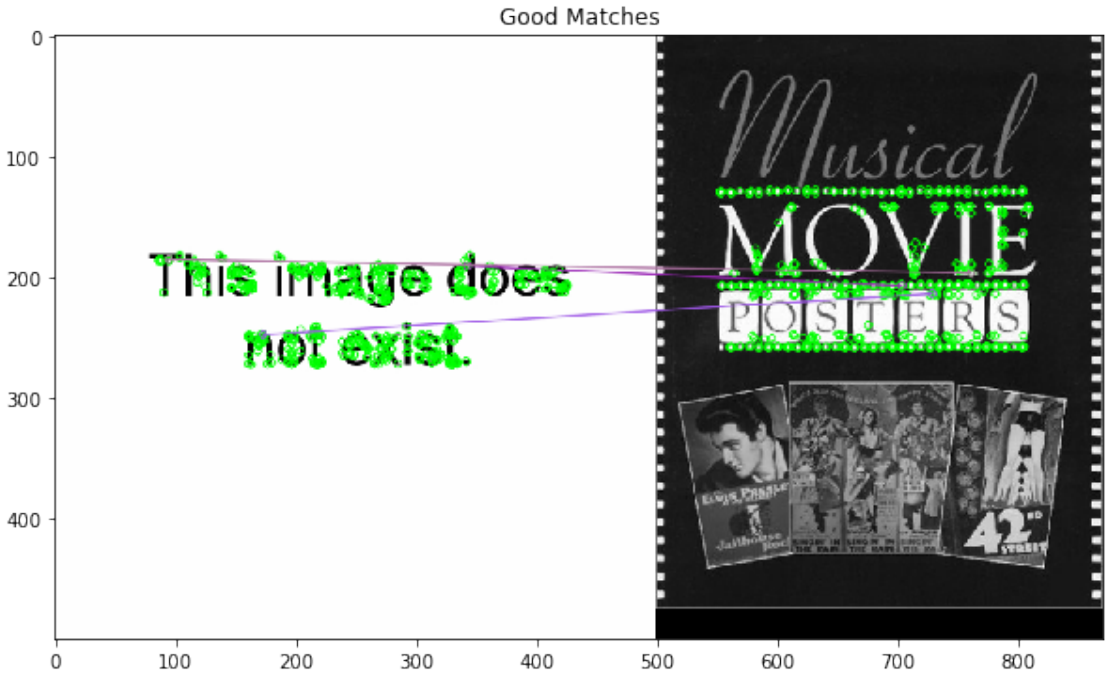
The above example is a great test beacsue the iamges is tilted and a little distorted, however it still gives a good outline.

This shows that RANSAC is a better option that LEASE SQUARE METHOD.

Most of the parameters are set to their default values, with the ratio test at 0.8


Key Points: Query Image
This image does
not exist.
0
100
200
300
400
0
100
200
300
400


Key Points: Reference Image
Musical
MOVIE
POSTERS
0
100
200
300
400
0
50
100
150
200
250
300
350

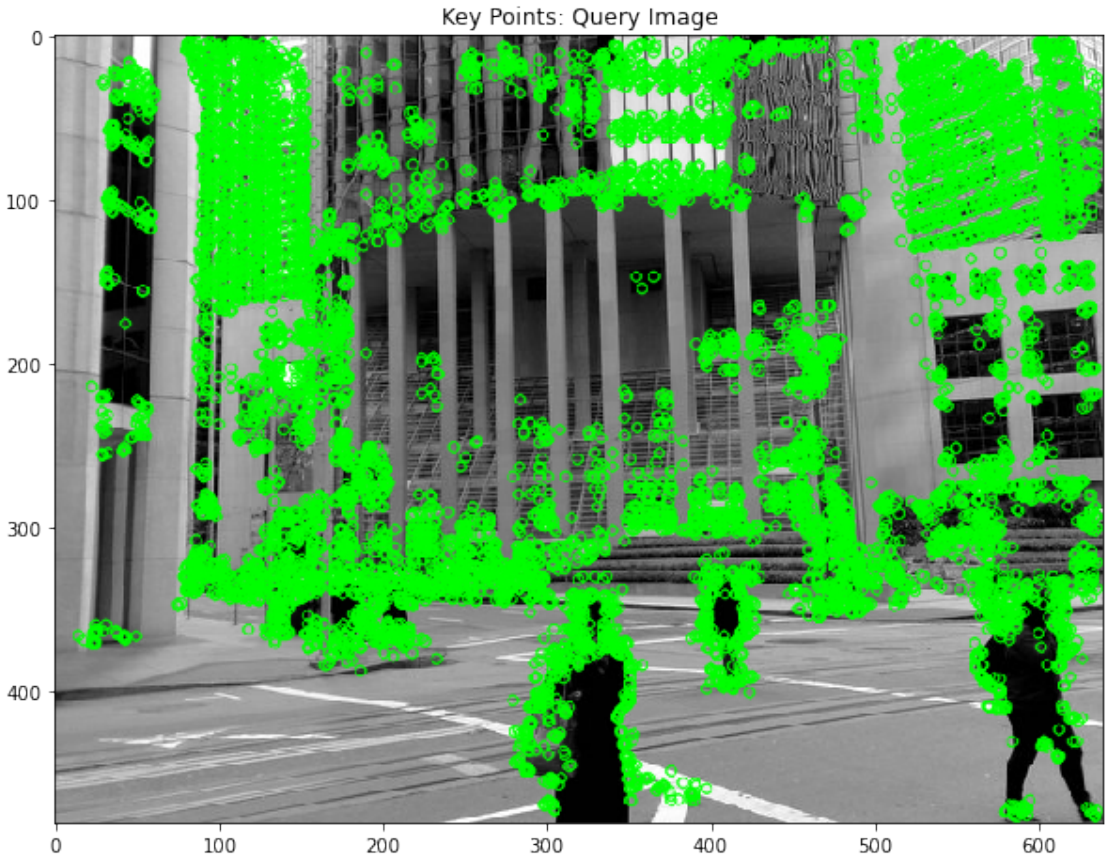


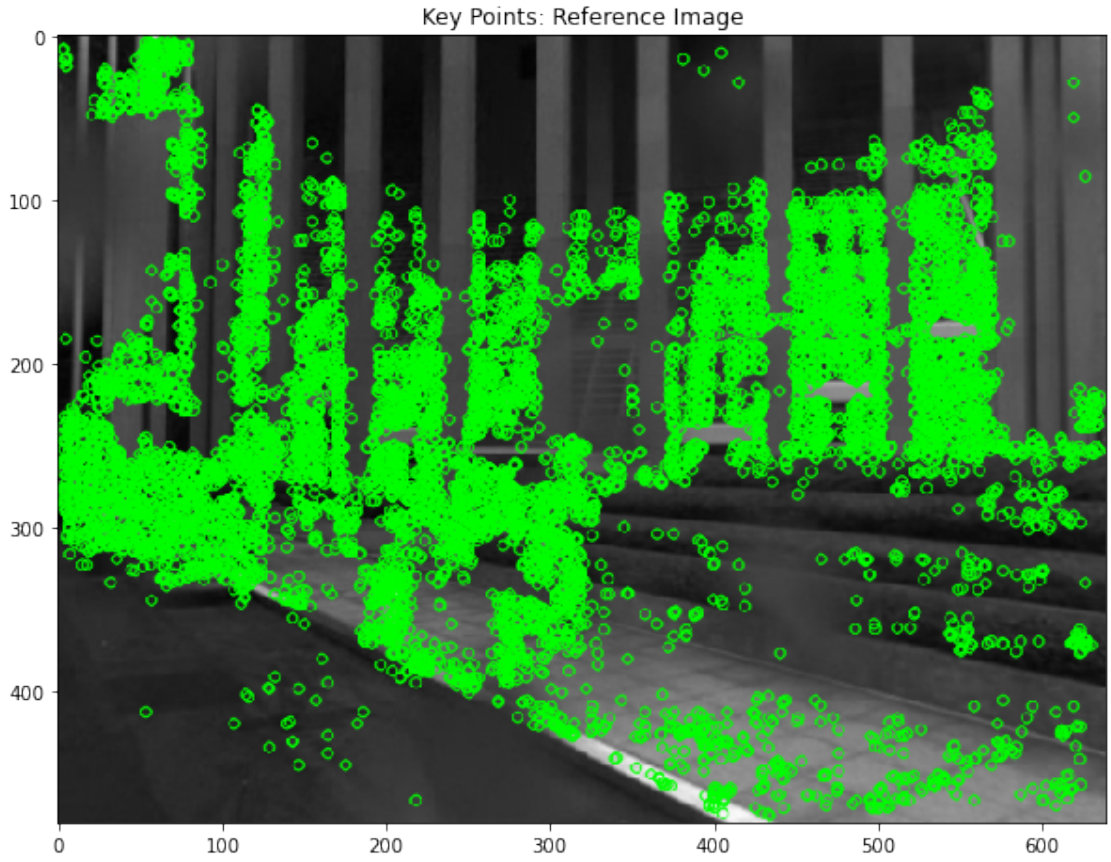

```
NUMBER OF GOOD MATCHES:  3
THE IMAGE DOESNOT EXIST
```

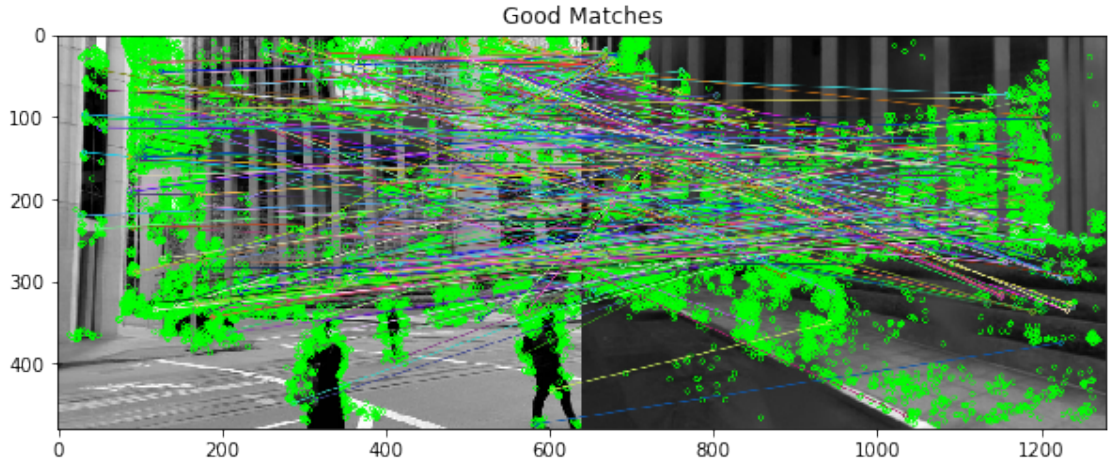
The above image is used bacsuse the query image for this is does not exist and this is a 100% incorrect case.

So the prediction of “not in the dataset” is a correct prediction.

The parameters are still at their default settings.


Key Points: Query Image
0
100
200
300
400
0
100
200
300
400
500
600





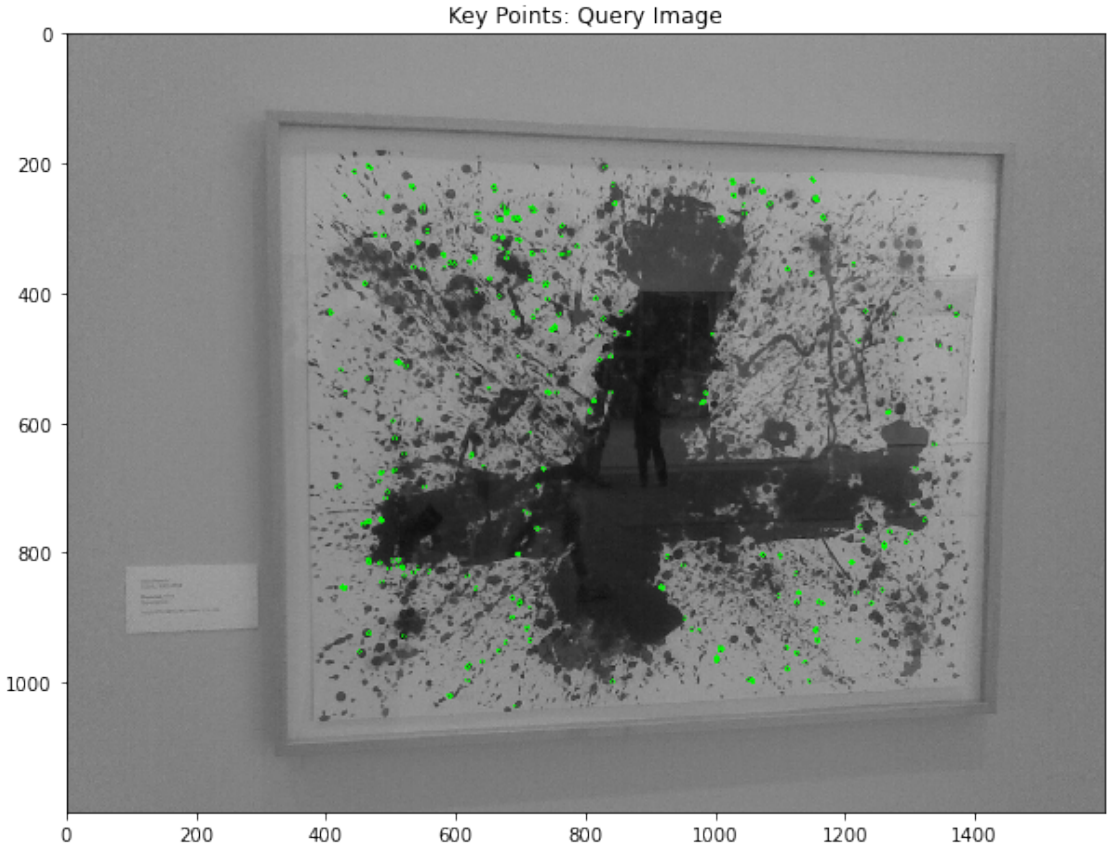

```
NUMBER OF GOOD MATCHES:  242
```

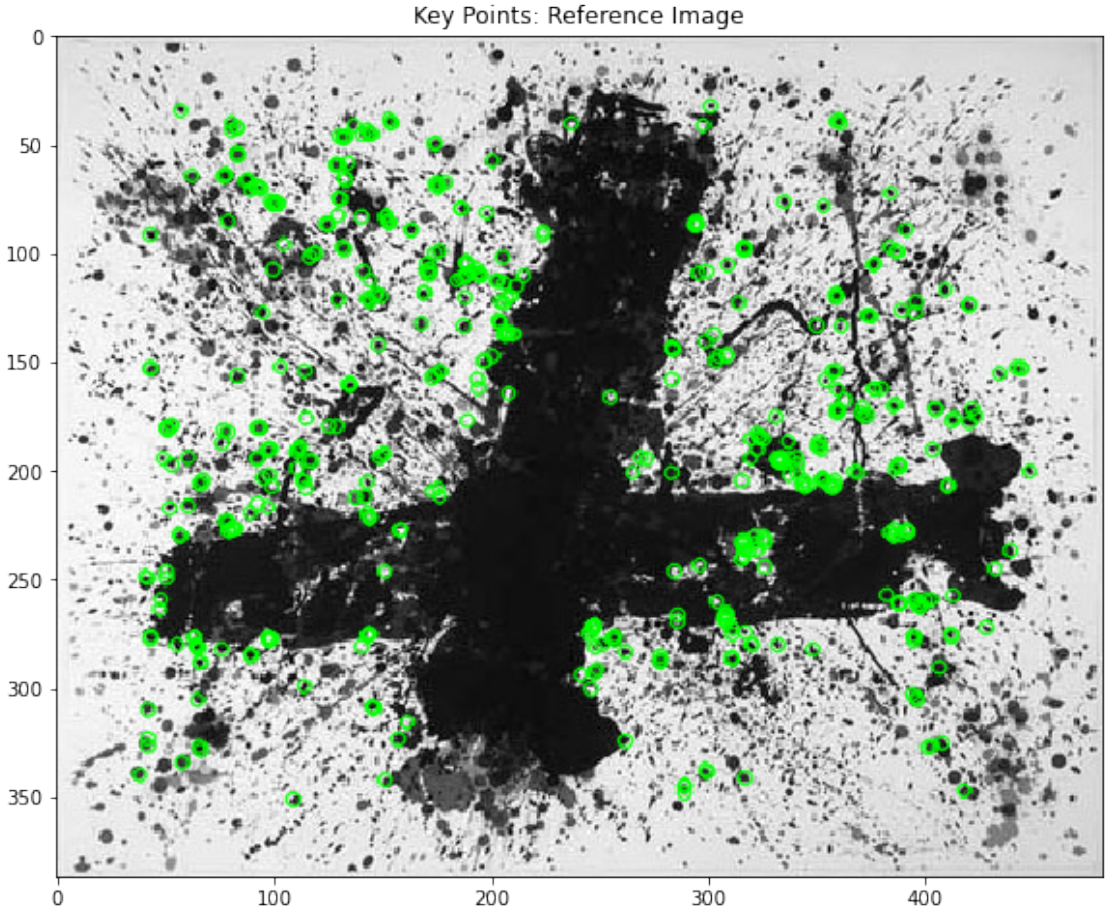
The above test is taken from the landmarks dataset. The specific query was selected because it had a lot of noise and there was a considerable difference of the angle at which the image was taken.

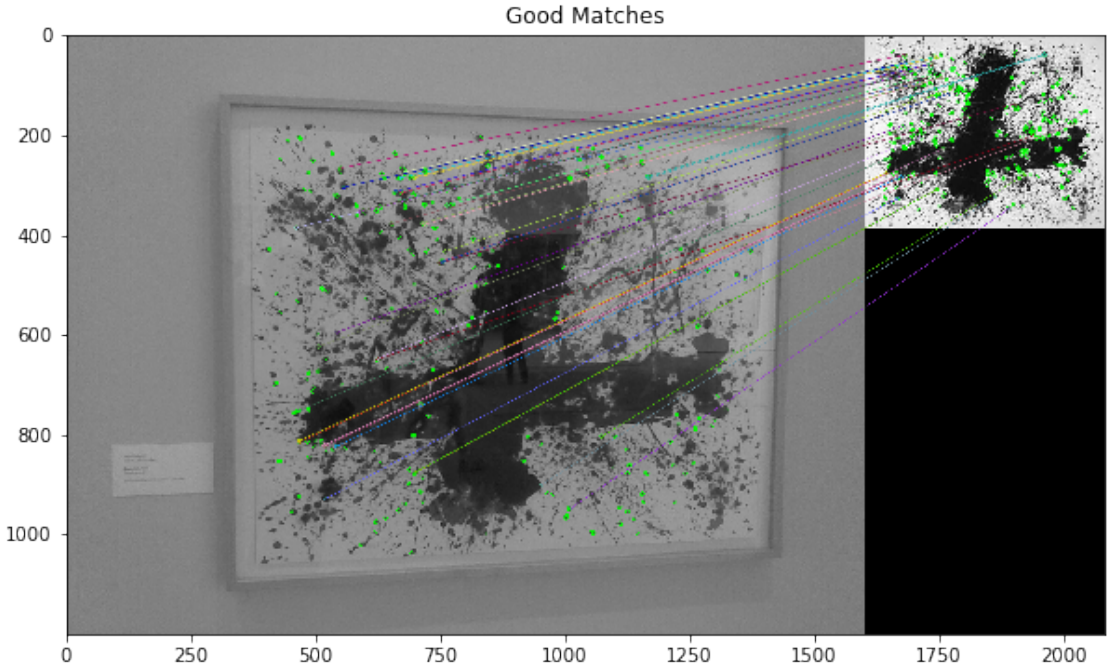
The RANSAC method is very good at handling outliers. However, in this case, the RANSAC method is not able to find a good match with the same parameters as the previous test. (k=2, ratio=0.8, norm=NORM_HAMMING) the total number of good matches is 16, at 0.8 ratio

however, it gets to 453 with no ratio test, this shows the key points are not being detected properly or a lot of key points that are being detetcted are outliers.

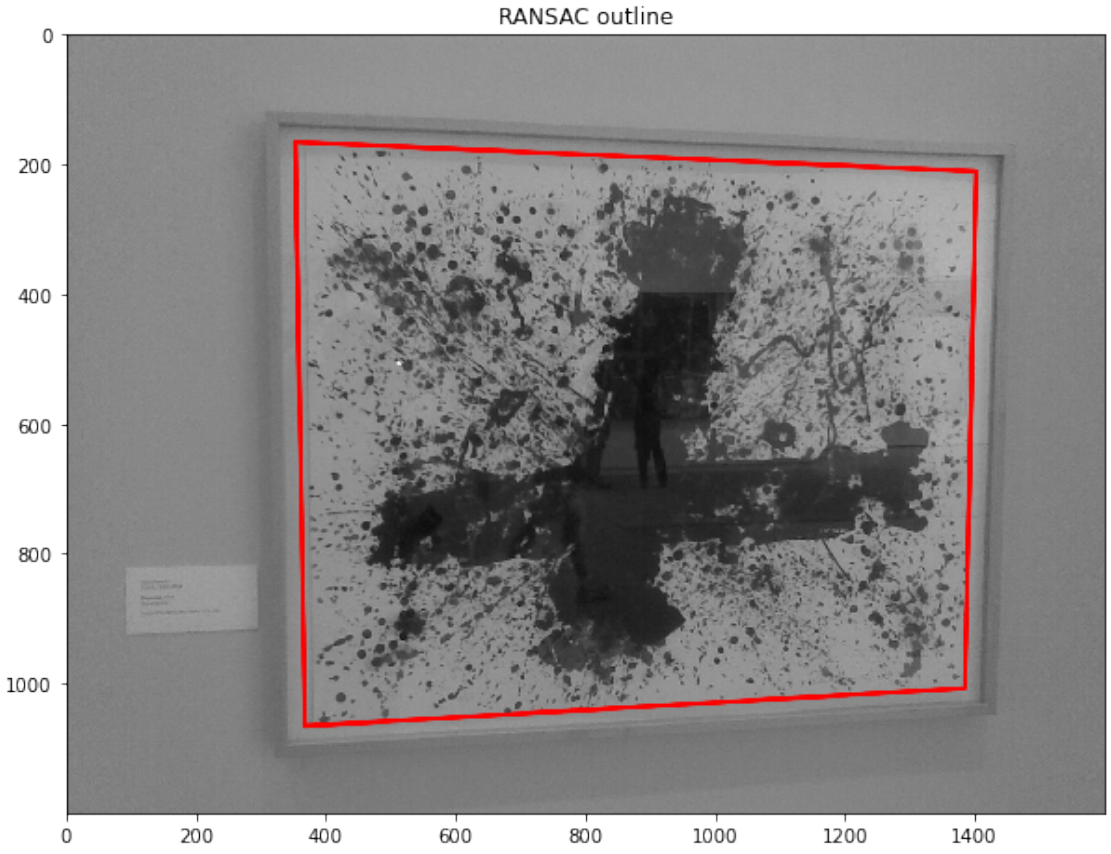
I also changed the ORB parameters to try and get a better result. I set the nParameter to 10000 - to get more key points that would help the match I set the fasthreshold to True, but the results are still not very accurate


Key Points: Query Image
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400


Key Points: Reference Image
0
50
100
150
200
250
300
350
0
100
200
300
400

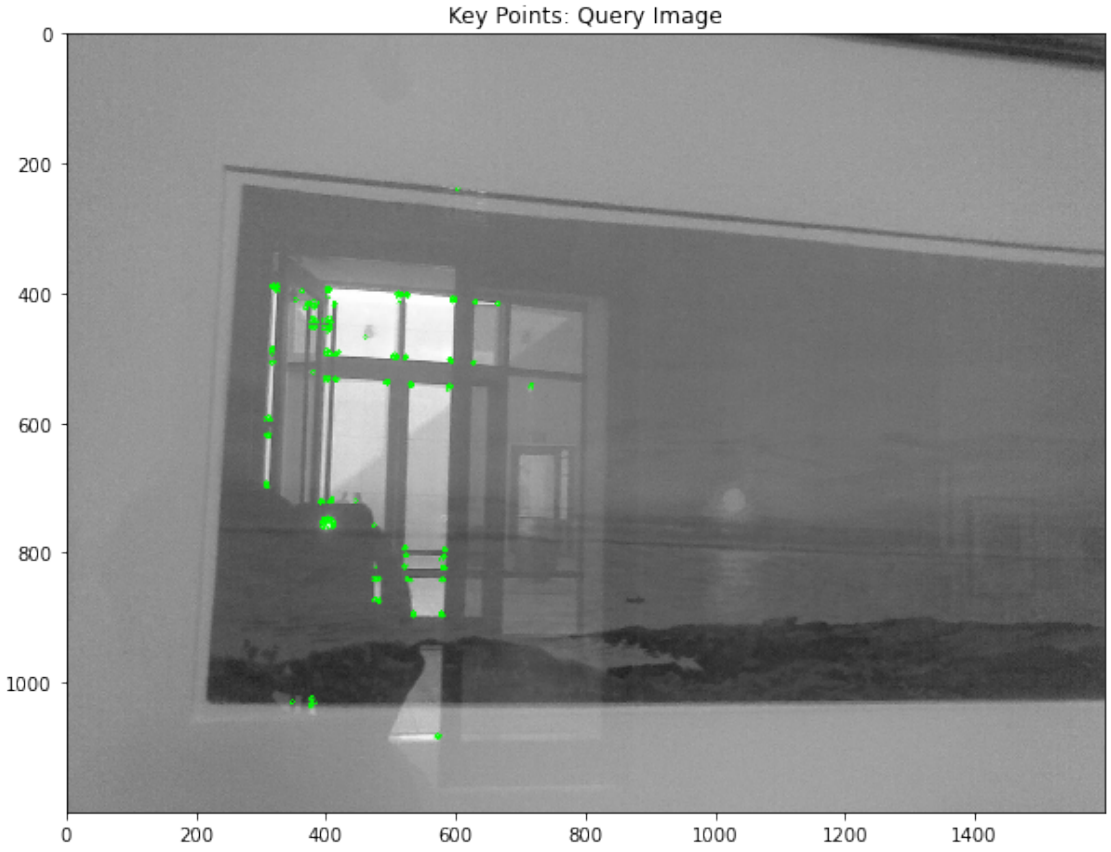

Good Matches
0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750
2000

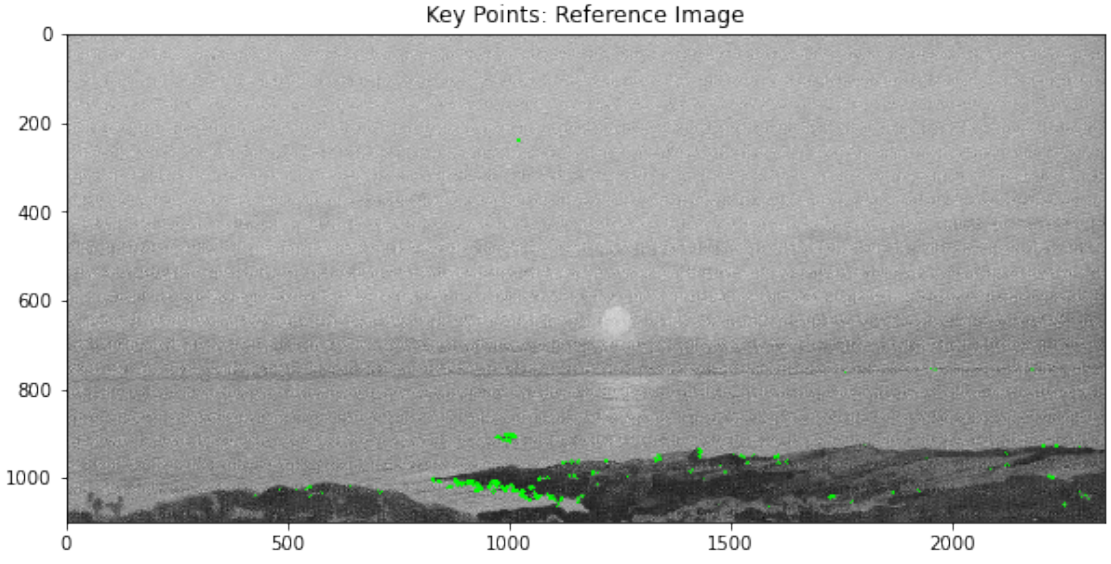
```
NUMBER OF GOOD MATCHES:  44
```

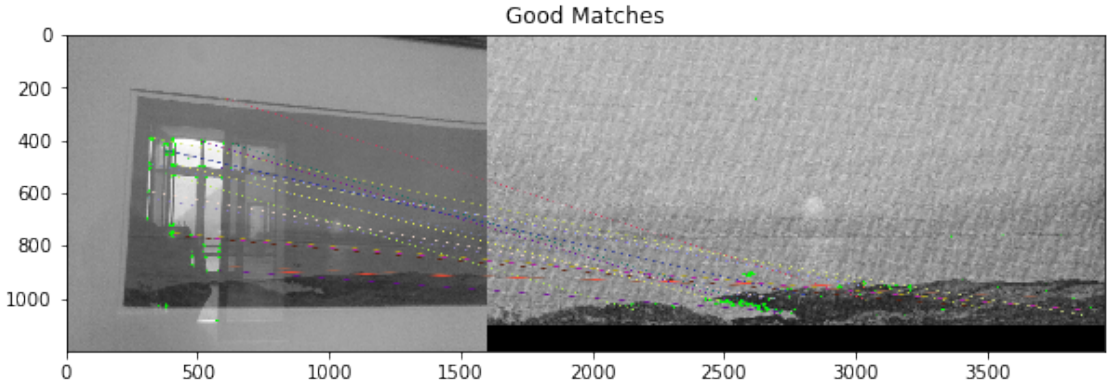


The above test produces good results as we get some good matches that are not outliers. The query image is also very clear with not much of a distortion.

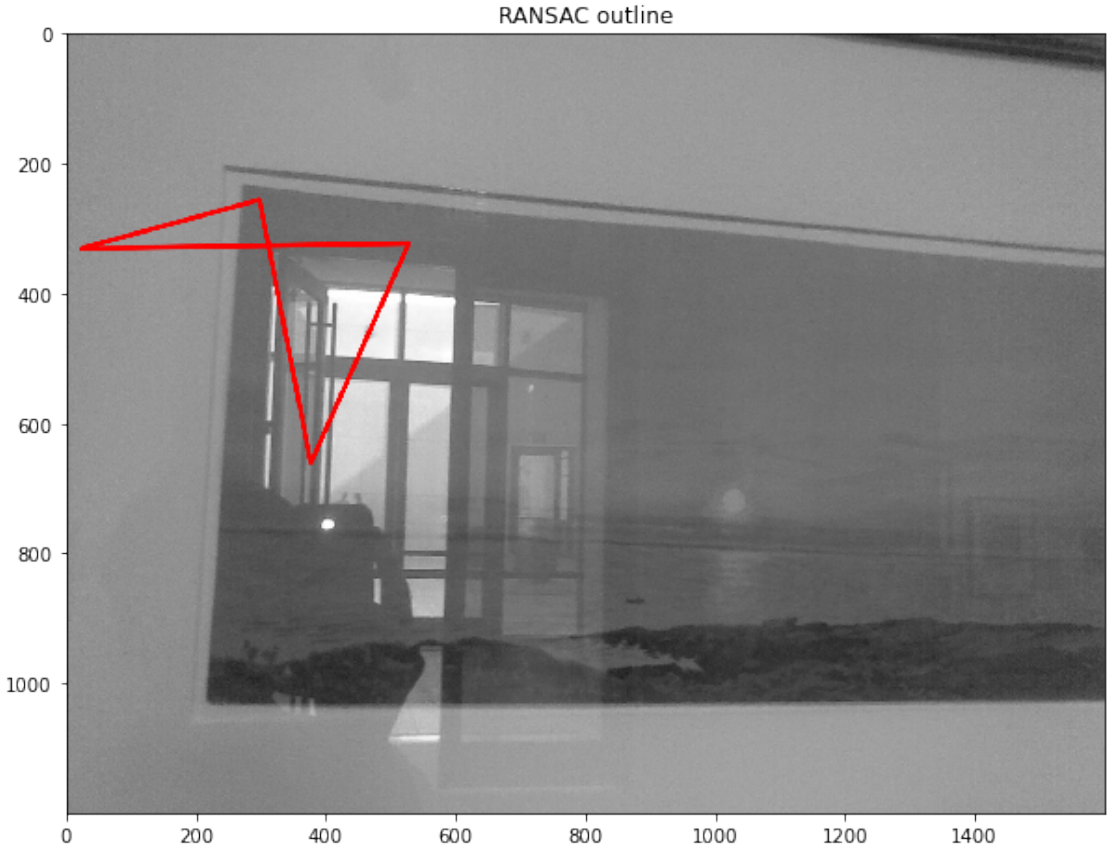
Here , the parameters are again set to default with the ORB to 200 samples and the ratio test at 0.8


Key Points: Query Image
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400


Key Points: Reference Image
0
200
400
600
800
1000
0
500
1000
1500
2000



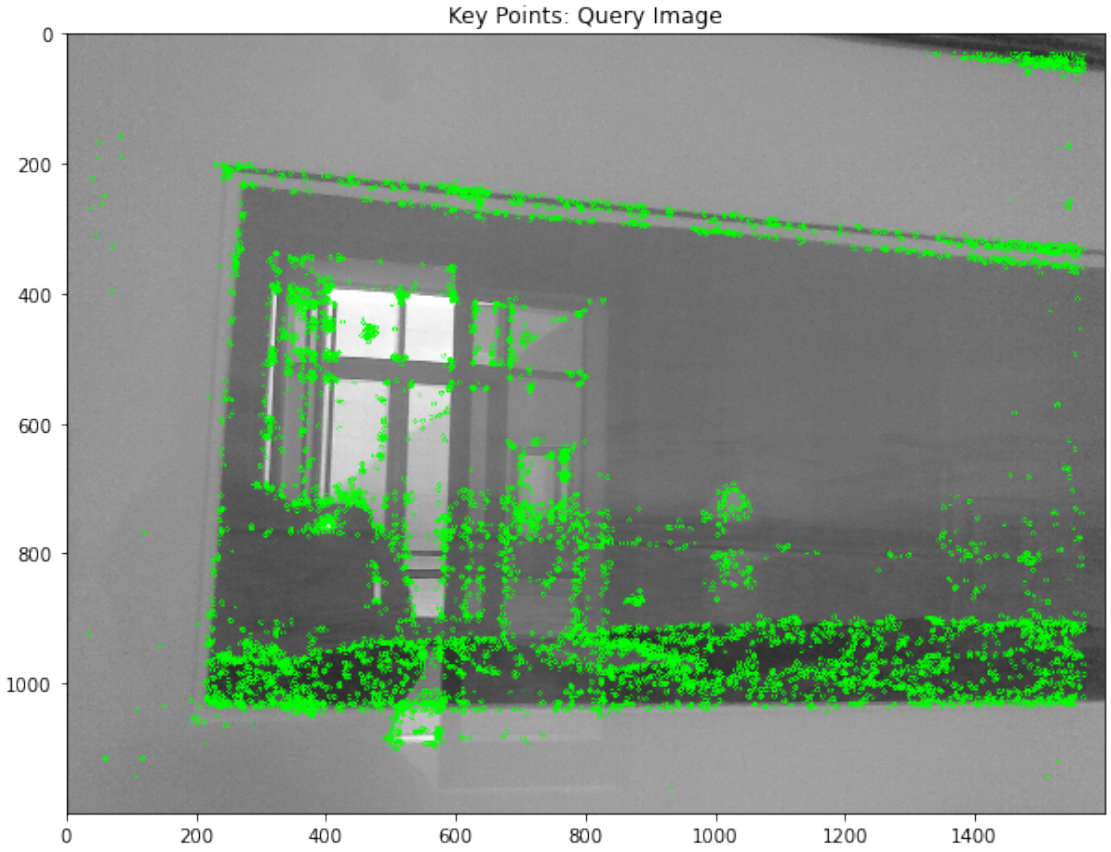

```
NUMBER OF GOOD MATCHES:  25
```

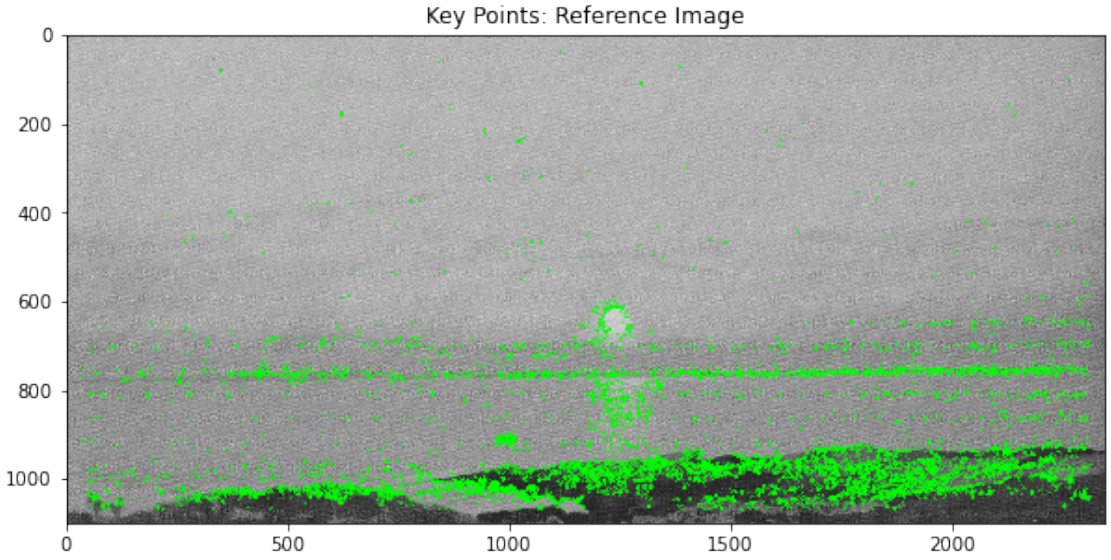


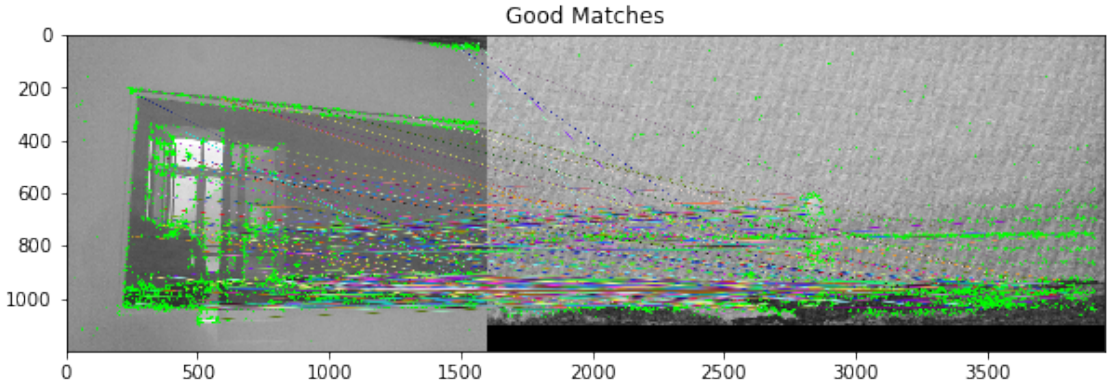
The above image is selected as it is a good test because of the major reflection that it has, it also has a little distortion.

Because of all that, the results with the default setting is not correct at all. The ORB is at 200 samples and the ratio test is at 0.8

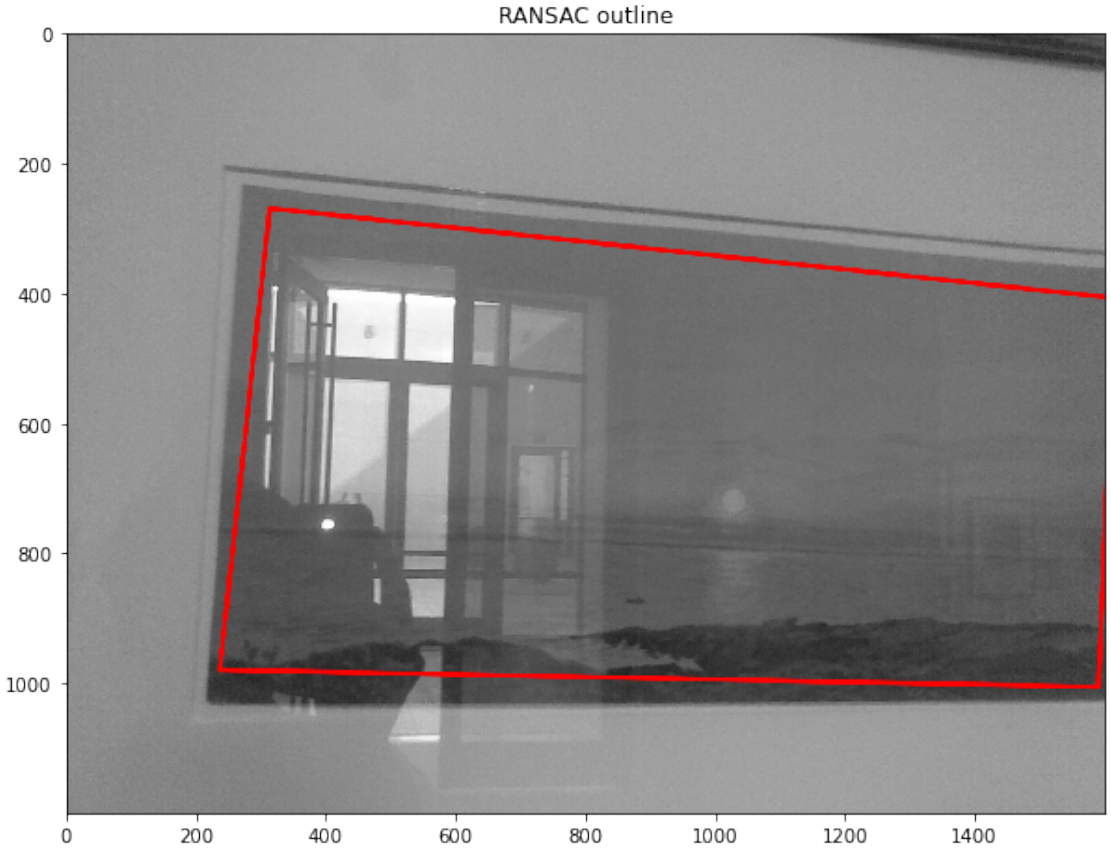
I am plannig to change those parametes for the next run!








```
NUMBER OF GOOD MATCHES:  289
```



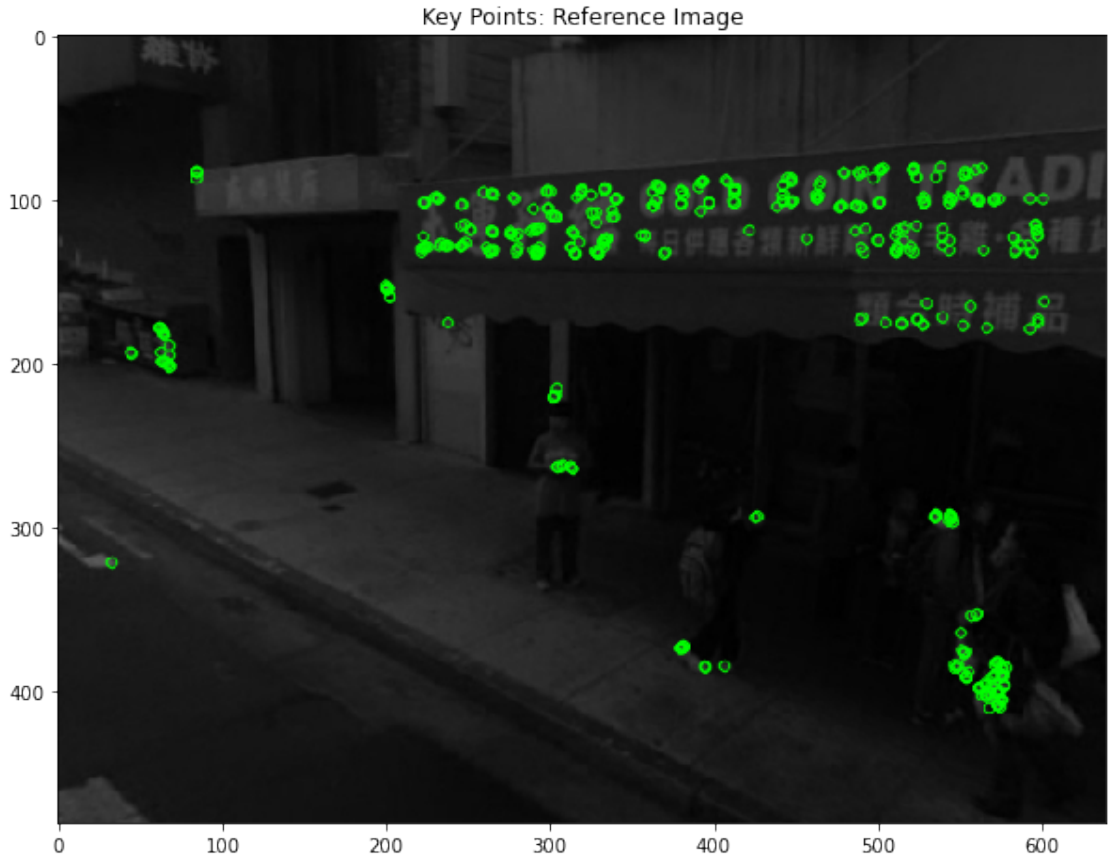
Initial:

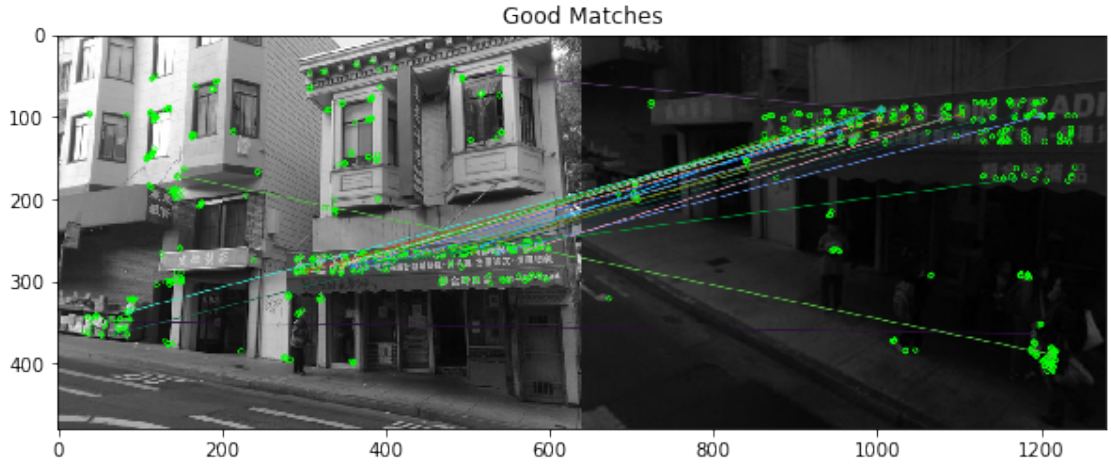
The above test gave a poor result and we only get a few good matches, which are outliers as the drawmatches() function shows that the matches are with a background reflection on the query image.

We could say that most/almost all of the matches are outliers, as with 0.8 the ratio test gives 25 matches, but with ratio = 1 (no penalty) the ratio test gives 447 matches, and ORB is at 200 samples

After:

But after changing the ORB parameter to 10000 samples and setting fast threshold to True, it gave a really good outline.






```
NUMBER OF GOOD MATCHES:  24
```

The above test from the 'landmarks' dataset is also very good. The query image is very clear and the reference image is not distorted. hence, the result is accurate.

Your explanation of results here

The above tests clearly indicate that the test workes very well on all the examples where the query image is not distorted. The RANSAC method is very good at handling outliers, but not when all the key points give incorrect matches. This mostly happens with the images from the 'landmarks' dataset and the 'museum_paintings' dataset.

Changing the ORB parameter 'nsamples' gave the best results when we changes it from 200 samples to 10000 samples. However, it takes more time than the previous methods.

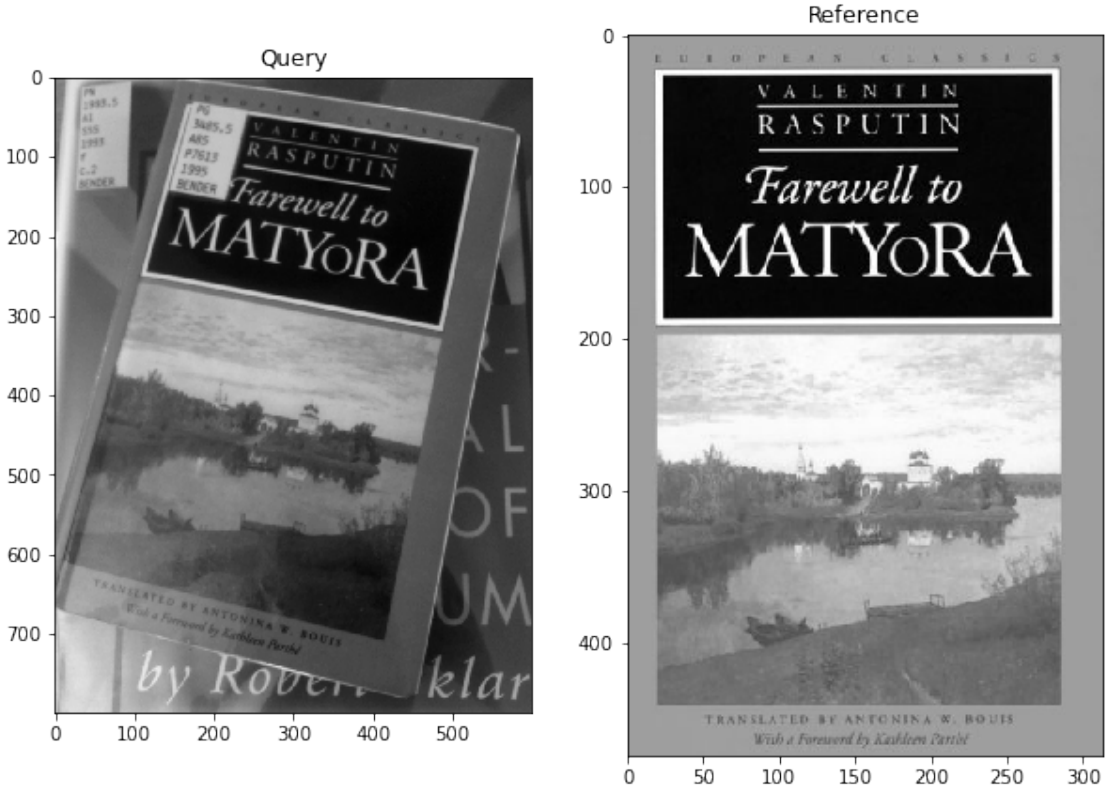
3 Question 2: What am I looking at? (40%)

In this question, the aim is to identify an "unknown" object depicted in a query image, by matching it to multiple reference images, and selecting the highest scoring match. Since we only have one reference image per object, there is at most one correct answer. This is useful for example if you want to automatically identify a book from a picture of its cover, or a painting or a geographic location from an unlabelled photograph of it.

The steps are as follows:

1. Select a set of reference images and their corresponding query images.
    1. Hint 1: Start with the book covers, or just a subset of them.
    2. Hing 2: This question can require a lot of computation to run from start to finish, so cache intermediate results (e.g. feature descriptors) where you can.
2. Choose one query image corresponding to one of your reference images. Use RANSAC to match your query image to each reference image, and count the number of inlier matches found in each case. This will be the matching score for that image.
3. Identify the query object. This is the identity of the reference image with the highest match score, or "not in dataset" if the maximum score is below a threshold.
4. Repeat steps 2-3 for every query image and report the overall accuracy of your method (that is, the percentage of query images that were correctly matched in the dataset). Discussion of results should include both overall accuracy and individual failure cases.
    1. Hint 1: In case of failure, what ranking did the actual match receive? If we used a "top-k" accuracy measure, where a match is considered correct if it appears in the top k match scores, would that change the result?

```
The book is in the reference library at the position: 40
```



The above test gives a correct output for the image number 55 ("This image does not exist")

The above test gives an incorrect prediction for image 67 (matching it to ref no.39)

The above test gives an incorrect prediction for image 70 (matching it to ref no.97)

The above test gives an incorrect prediction for image 76 (matching it to ref no.11)

```
50 correct predictions from 50 images
The accuracy of the prediction is : 100.0
```

The first 50 images gives a 100% result

using the current parameters of nsamples of 10000 and threshold of 15

```
97 correct predictions from 101 images
The accuracy of the prediction is : 96.03960396039604
```

```
Query : Predictions
54      96
67      39
70      97
76      11
```

The above test gives a correct output for the image number 55 (“This image does not exist”)

The above test gives an incorrect prediction for image 67 (matching it to ref no.39)

The above test gives an incorrect prediction for image 70 (matching it to ref no.97)

The above test gives an incorrect prediction for image 76 (matching it to ref no.11)

```
98 correct predictions from 101 images
The accuracy of the prediction is : 97.02970297029702
```

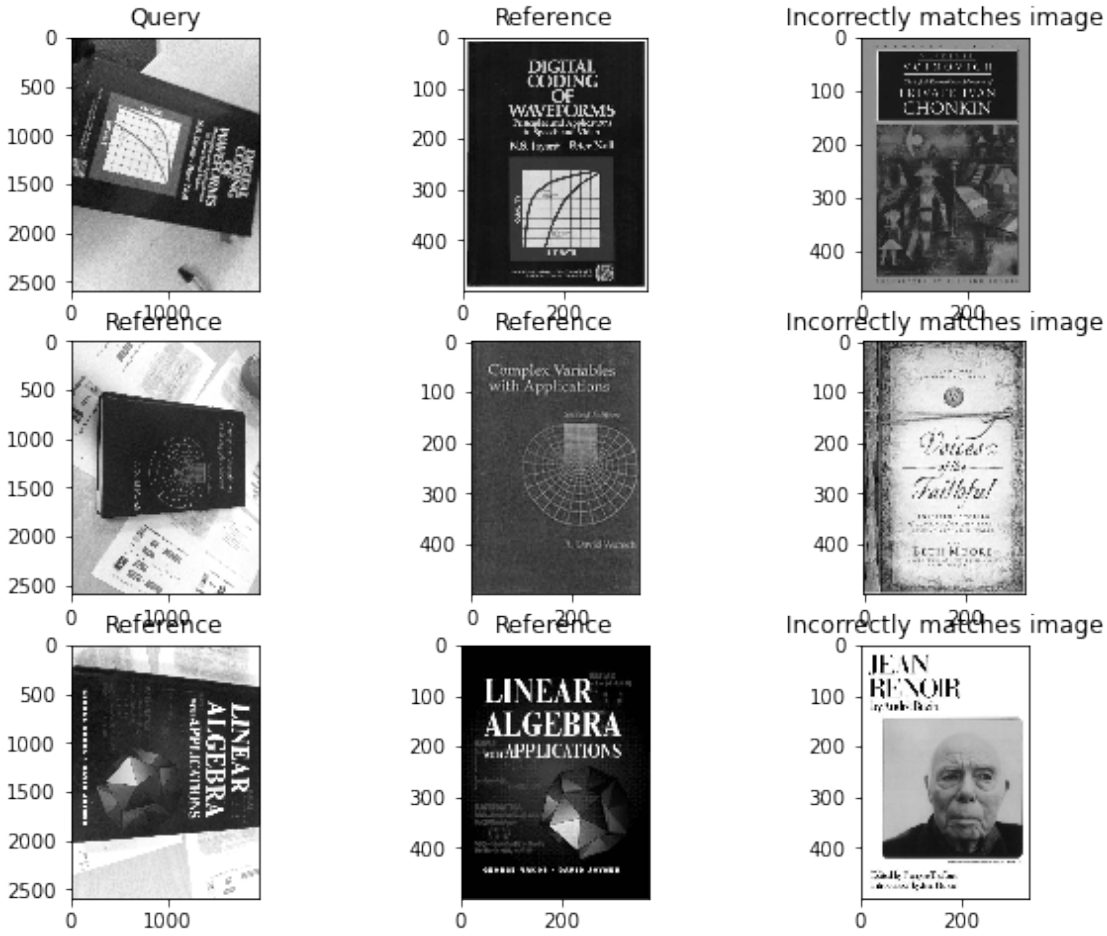
```
################################
Predicted      Actual
39              67
99              67
11              67
22              67
13              67

################################
Predicted      Actual
97              70
24              70
87              70
0               70
8               70

################################
Predicted      Actual
11              76
16              76
17              76
```

```
30                76
84                76
```

```
Text(0.5, 1.0, 'Incorrectly matches image')
```



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

For the overall accuracy I got 96.03% with 3 incorrectly matched cases.

I used the top-k matches but still got nothing with 3, 5, 10 as k.

I have individually listed the query, the reference images above and the incorrectly matches image.

To change and improve the code for part 4, which was originally just a for loop with all the steps, I made two functions the 'get_good_matches' and the 'get_inline_matches'. These functions are used to get the good matches and the inline matches respectively.

After predicting all the query images and getting an overall accuracy of 97%, there were 3 cases that were incorrect.

So I used the top - 5 predictions for each of them and from the above output we can see that the

cases did not give correct answer even after that.

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.

```
101
101

94.05940594059405

number of correct predictions: 95
96 54 20
-1 67 14
-1 70 12
-1 74 15
11 76 16
-1 92 14
-1 93 11
-1 94 14
-1 95 14
16 96 18
-1 97 9
-1 98 11
-1 99 11
-1 100 13
```

The dataset with extra images gave a 95 correct predictions out of 101 images. However, initially it was a lower accuracy and I had to increase the threshold value from 6 to 15 to match the extra images, so that it will consider them out of the dataset. The ORB is set to 10000 samples and fast threshold is set to True

thus, after that the accuracy went up to about 95%

Your explanation of results and any changes made here

To change the code for Q4 was originally just a for loop with all the steps, but to improve it, I made two functions the 'get_good_matches' and the 'get_inline_matches'. These functions are used to get the good matches and the inline matches respectively.

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.

```
91
91

the code took 1178.9475951194763 seconds
accuracy: 79.12087912087912 %
```

```
Query : Pred : max Score
12      -1      6
21      12      12
24      12      11
29      19      10
40      50      11
41      2       12
42      49      11
44      12      12
48      20      9
49      20      13
50      12      16
53      47      9
55      12      13
58      12      14
59      57      16
68      1       10
73      20      12
79      21      12
85      12      9
```

The initial run with the same **threshold** as the book covers = 15 gave a poor accuracy of around 51%.

Checking the average scores for the images, the threshold is set to a lower level, the ORB parameter was also changed.

I used the whole of the museum paintings dataset for the part above, the accuracy is lower than the one I got on the book covers, this was resolved my using a lower **Threshold** = 6 rather than 15 (on the book_covers) and by tweaking the ORB parameter **fast threshold** = 10 using the above mentioned changes the accuracy went up to around 80% from the previous 58%.

However, the time it takes to compute is much more than the previous one:

1180 seconds compared to the 557 seconds on the previous run

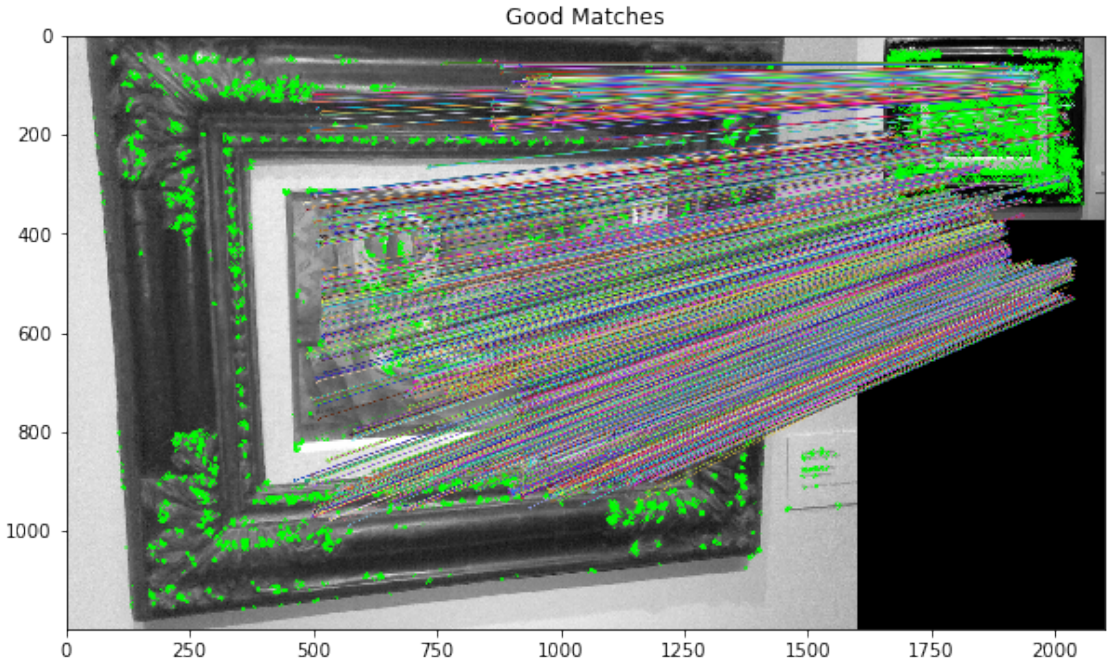
4 Question 3 (10%)

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'.

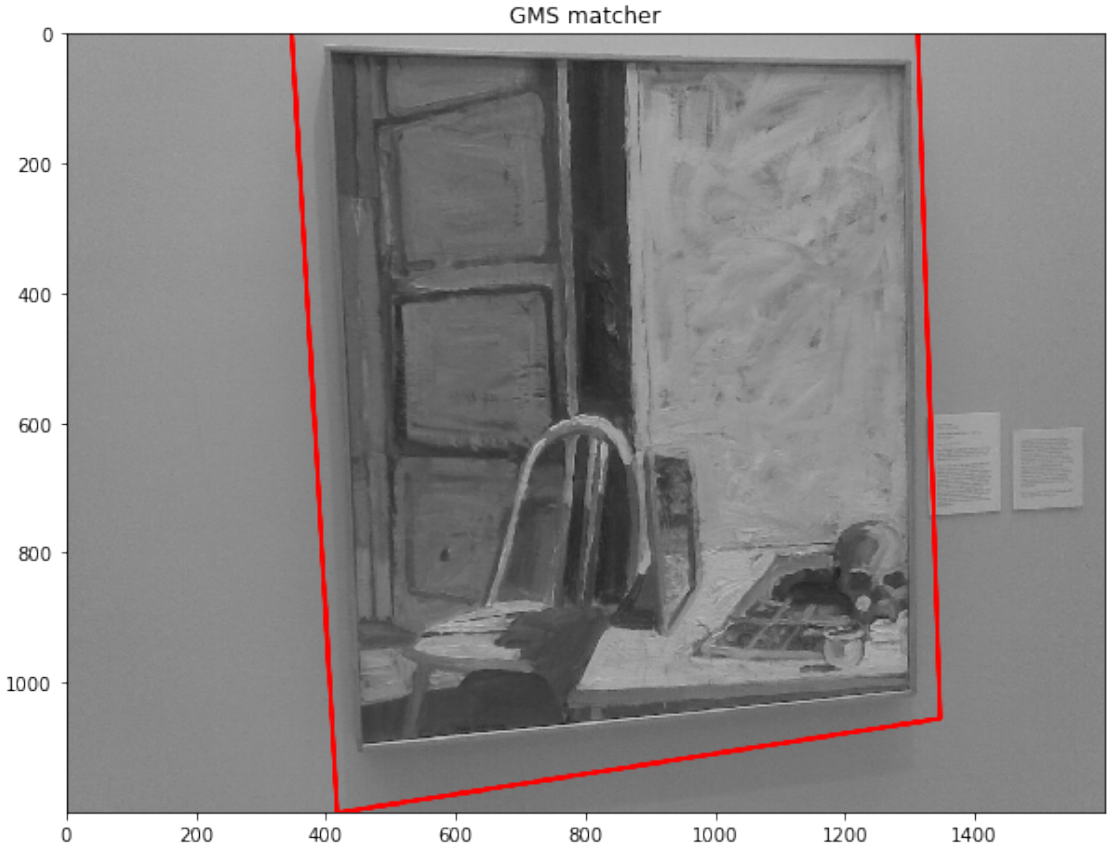
1. Hint 1: `cv2.xfeatures2d.matchGMS()` can be used, but you need to install the opencv-contrib by `pip install opencv-contrib-python`
2. Hint 2: You do not need use KNN matching, because GMS does not require second nearest neighbor.
3. Hint 3: You need to change the parameters in `cv2.ORB_create()` 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".

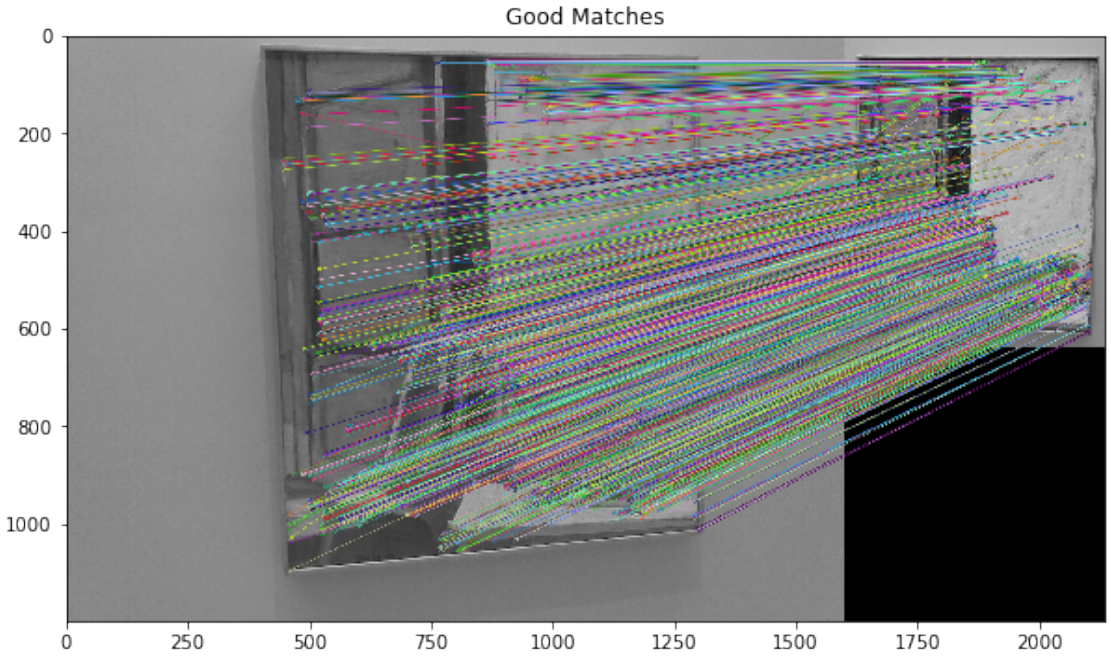
```
The number of matches is : 3043
```



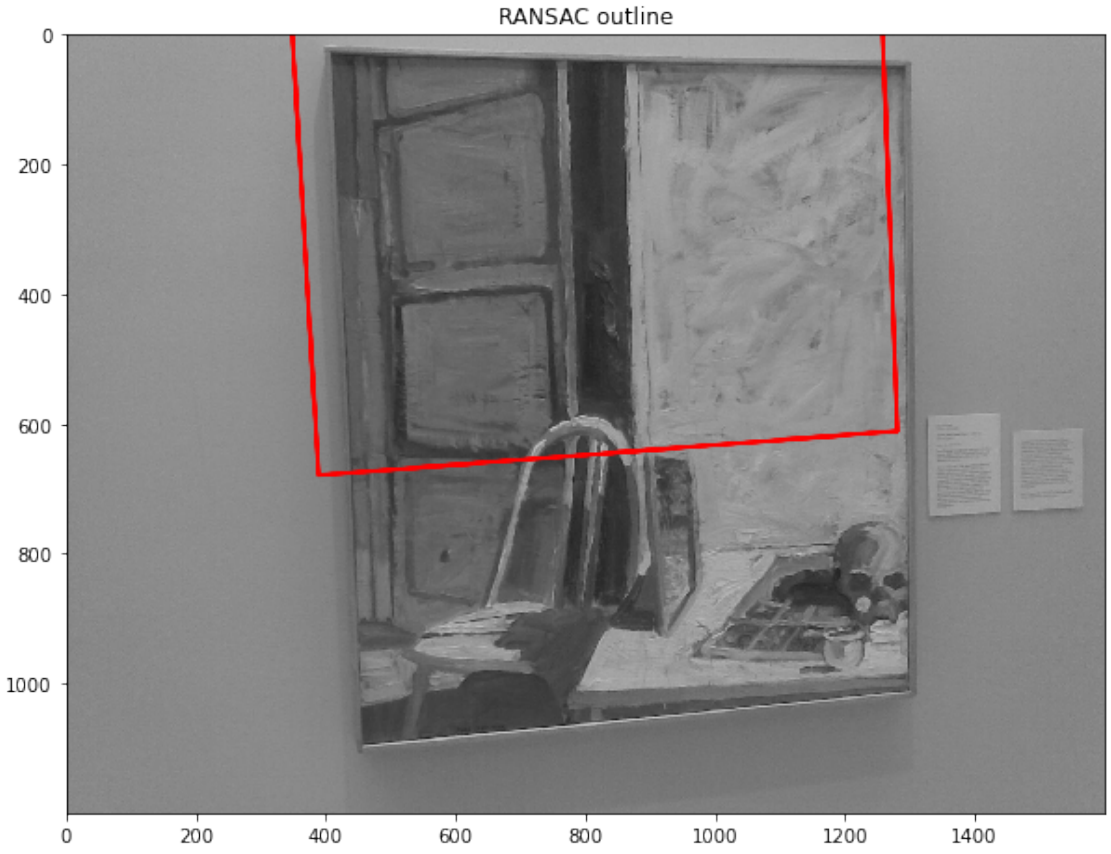

```
The number of inliers: 1706
```



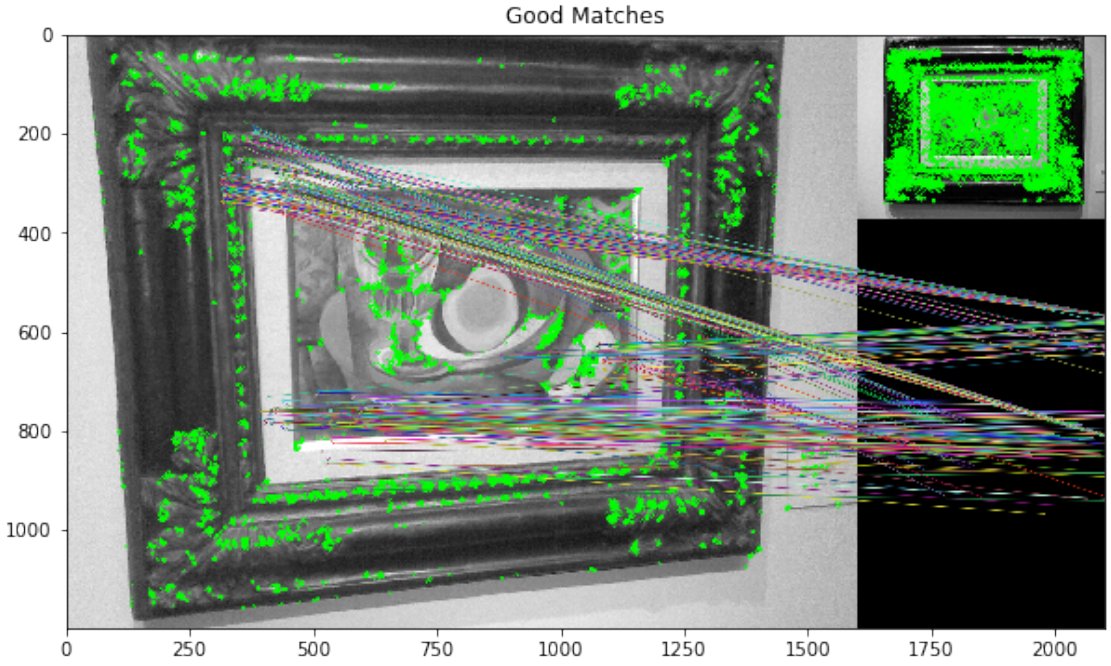

```
The number of ratio test matches is : 2293
```



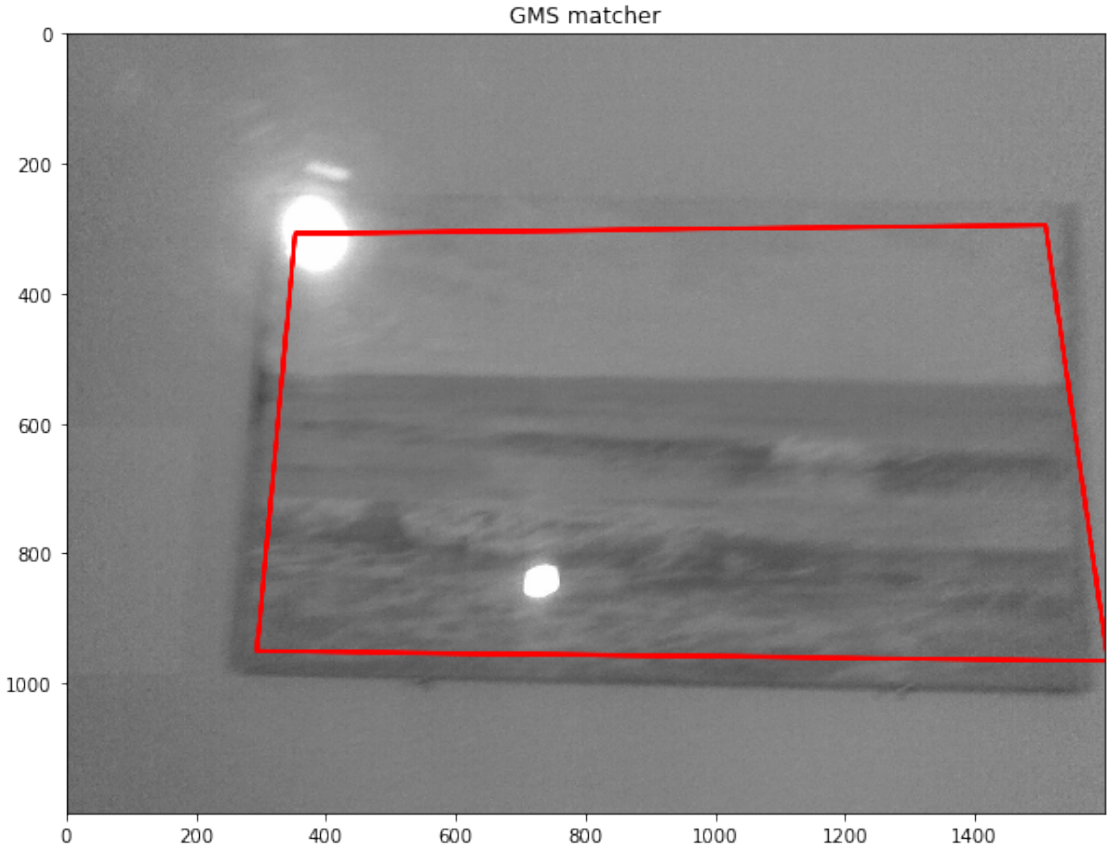
```
The number of inliers: 1366
```



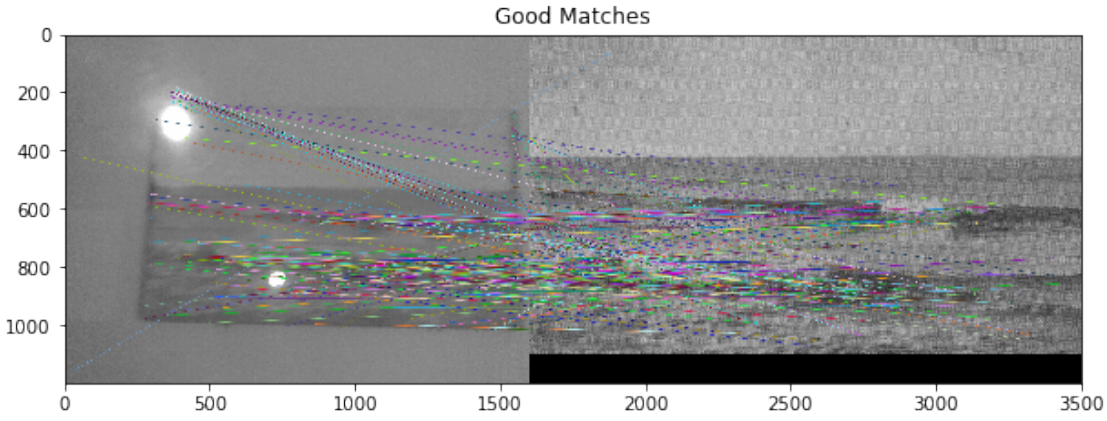

```
The number of matches is : 379
```



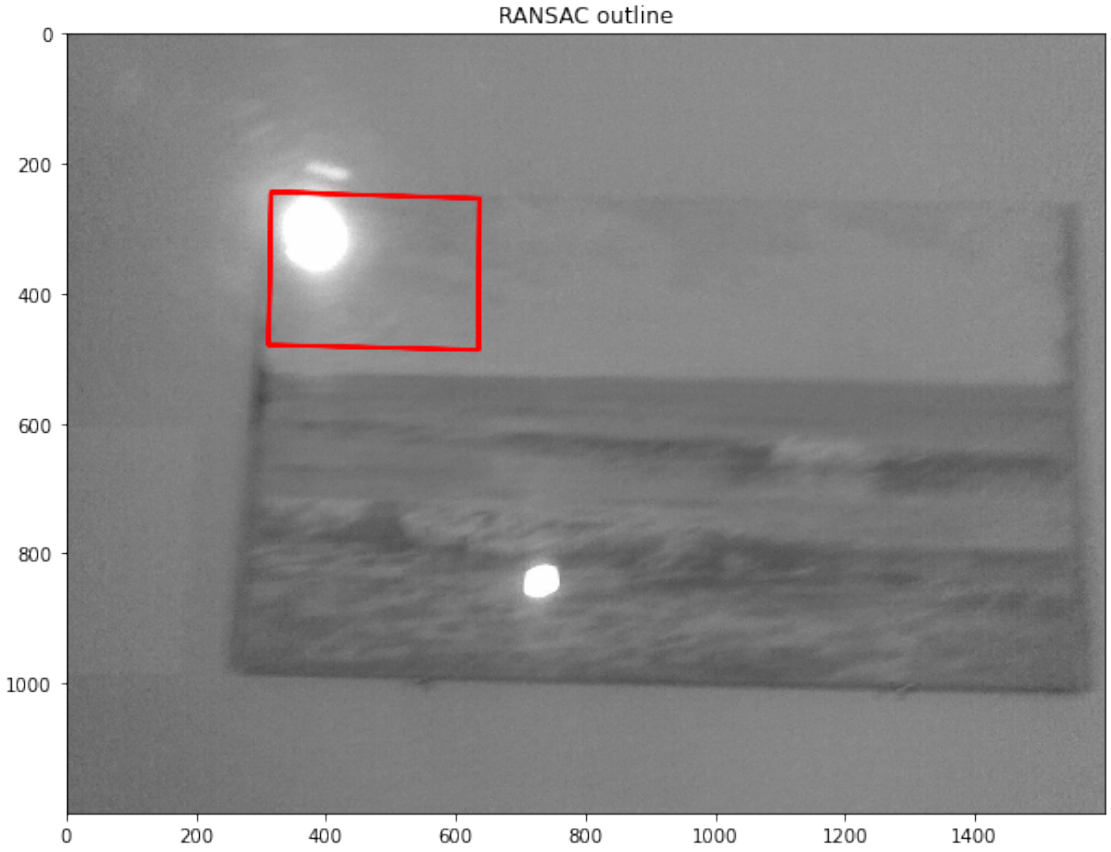

```
The number of inliers: 42
```



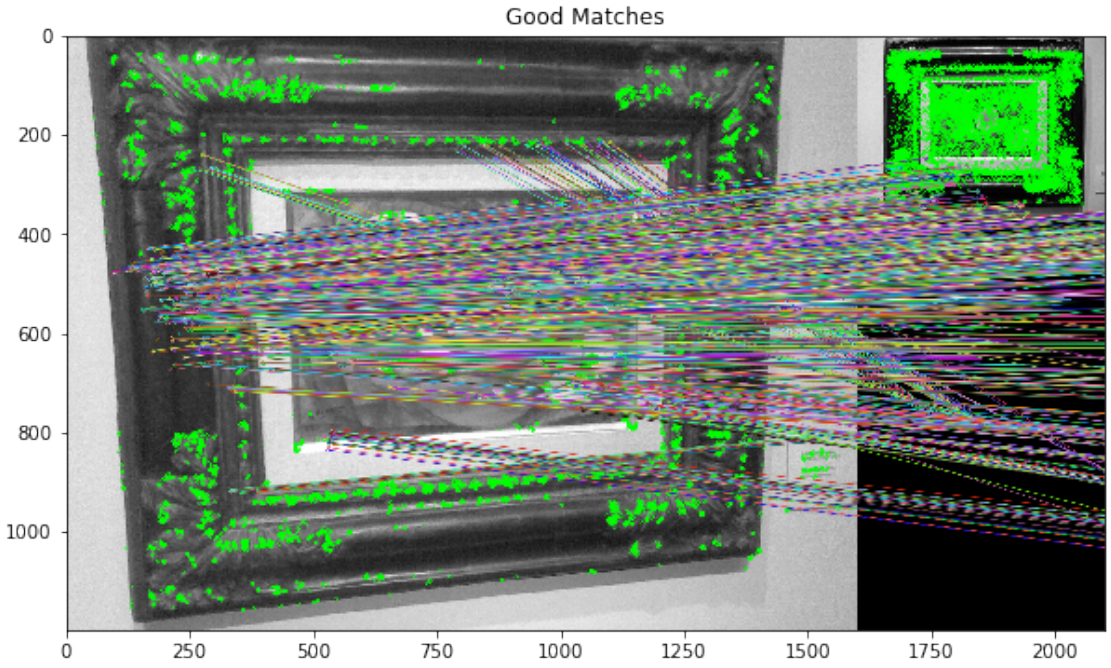
```
The number of ratio test matches is : 207
```



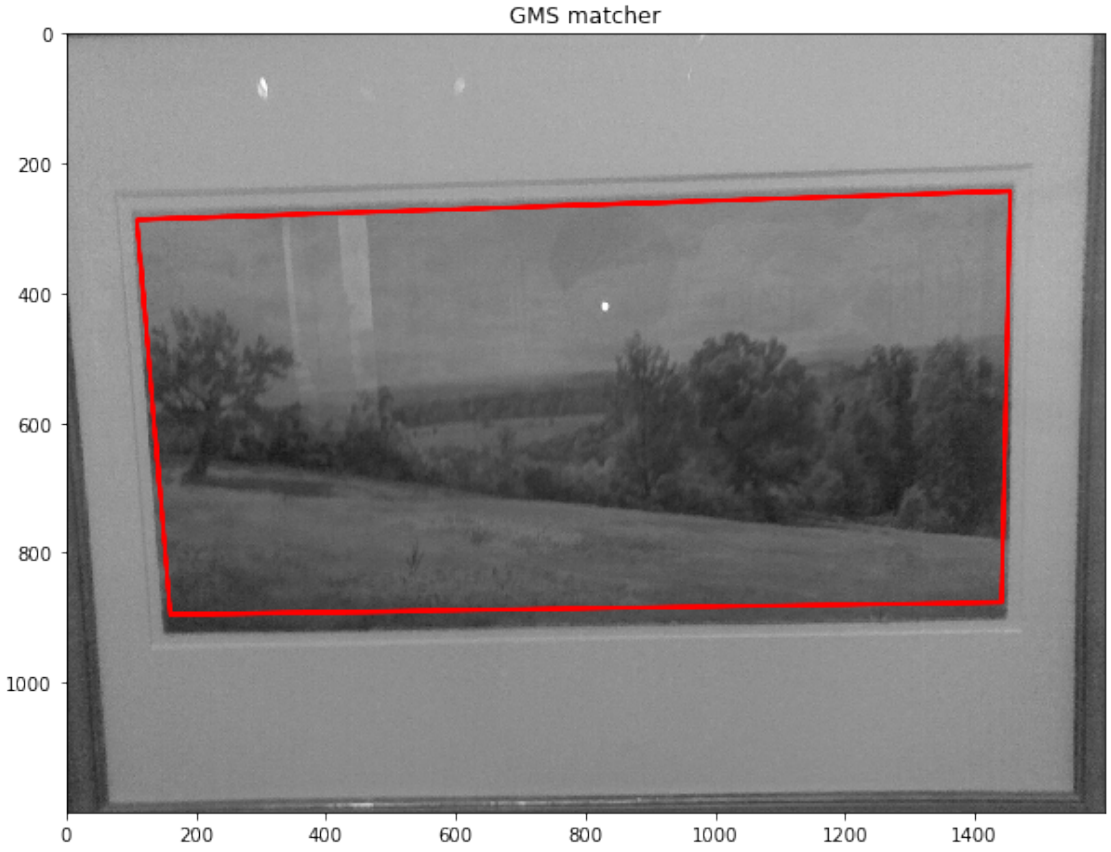

```
The number of inliers: 27
```



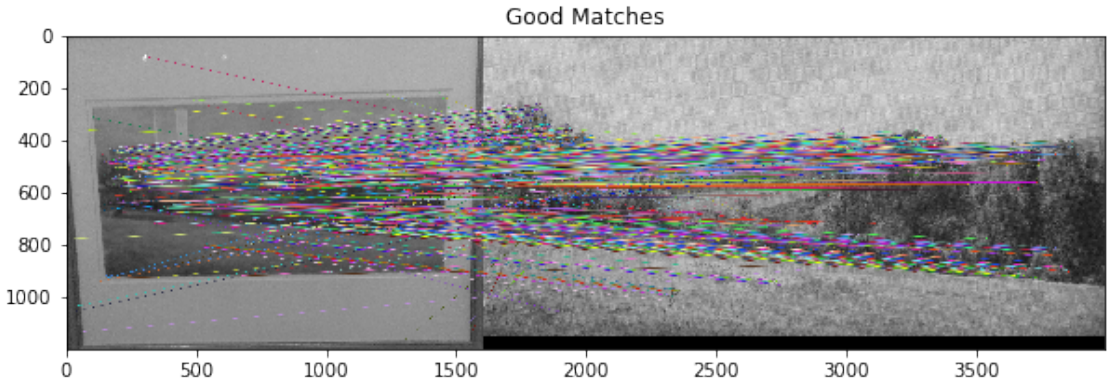
```
The number of matches is : 1493
```




```
The number of inliers: 645
```




```
The number of ratio test matches is : 705
```




```
The number of inliers: 404
```

Your results here

As we can clearly see from the above three tests that the GMS matcher gave a much better and accurate result than the regular ratio test.

By using the GMS matcher, the number of **ovarall matches** and the **number of inliers** are much more that the ones we get with the ragular ratio test.

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%)

1.

Like every project there was a profusion of challenges that we encountered while completing the project, for starters, in Q2, of Assignment two, the biggest challenge that I encountered was to develop a model that gave an acceptable accuracy. At the start of this, the models that I developed had an accuracy was about 56 %, and while persistence was a key, reading the documentation, and conducting ORB, optimising the parameters and trying sample datasets. With the help of

experimenting this algorithm on sample datasets, I could successfully devise and efficient model that produced an accuracy of about 80%.

Additionally, I remember encountering problems in my first assignment, with regards to time management, and not submitting an assignment at the level which intended it to be, and this problem was rectified in the assignment this time, I tried to complete my assignment to at least a satisfactory level, a significant time before it was due.

2.

Due to missing out on the deadline for the first assignment by about an hour I changed on how I worked on the second assignment: - I started working on assignemnt 2 very early on. - I realised that I underestimeted the ammount of work for the first assignment and I tried to improve on it by re-watching the lectures and starting as soon as ppossible.

- Along with that, instead of being stuck on one question for too long, I engaged more on the 'discussion board'. I found it extremly helpful in giving me hints in tricky situations.

All of these things helped me a lot in finishing the assignment on time