Ankit Nitinkumar Bhawsar
USC ID: 4889451132
abhawsar@usc.edu

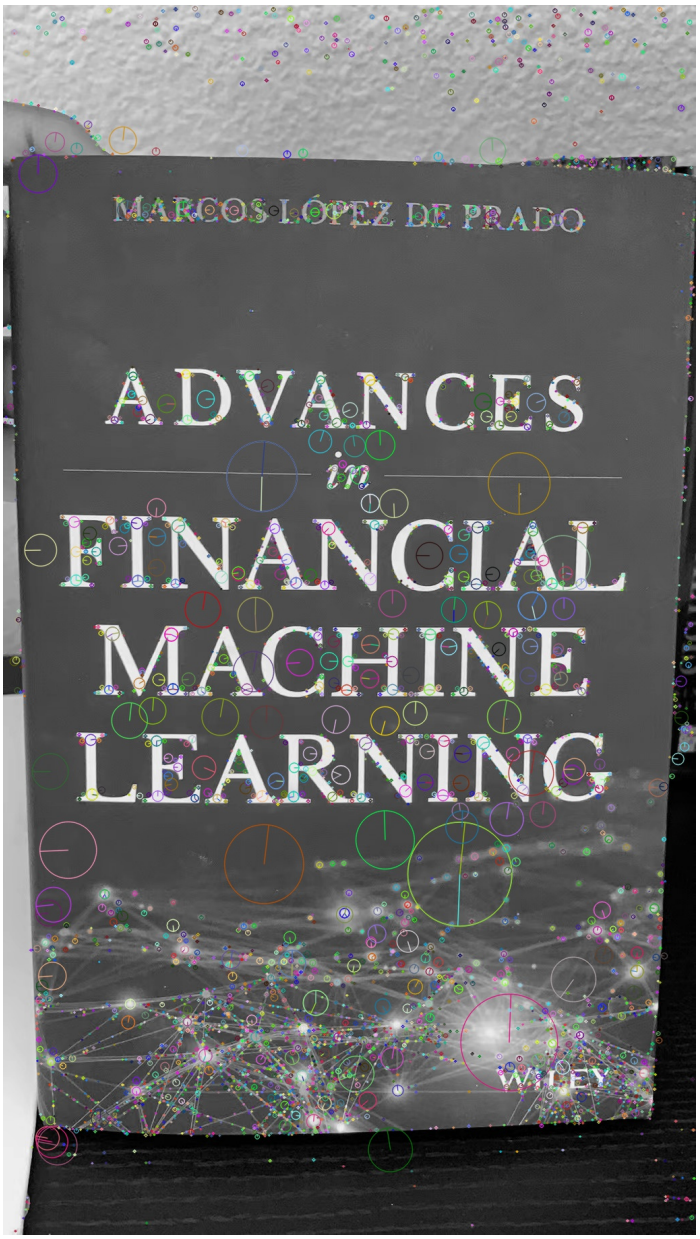
Assignment 3
CSCI 677: Advanced Computer Vision

2021-10-07

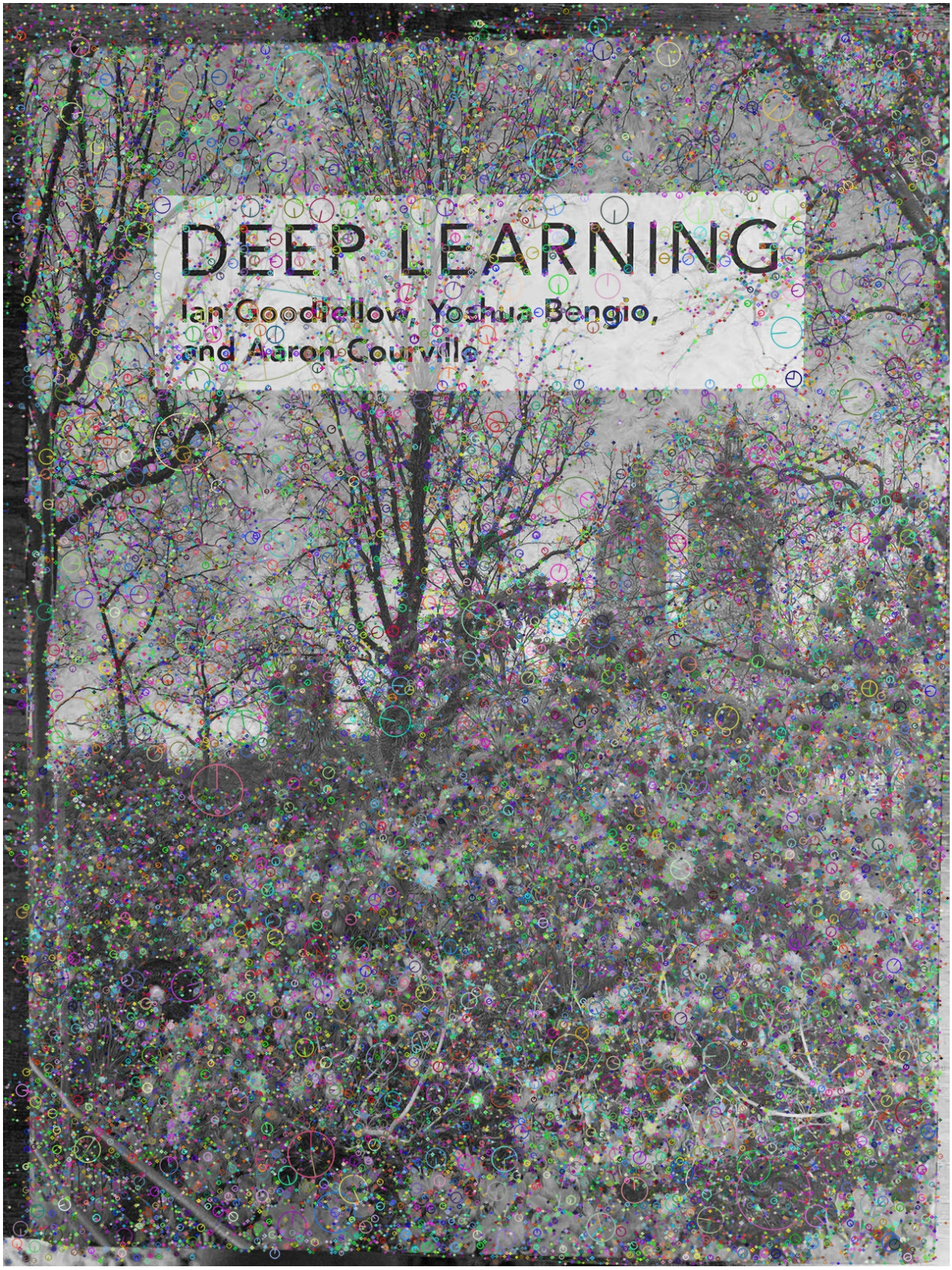
# 1 Program Description and Outputs

(a) **Creating and Computing SIFT Features**

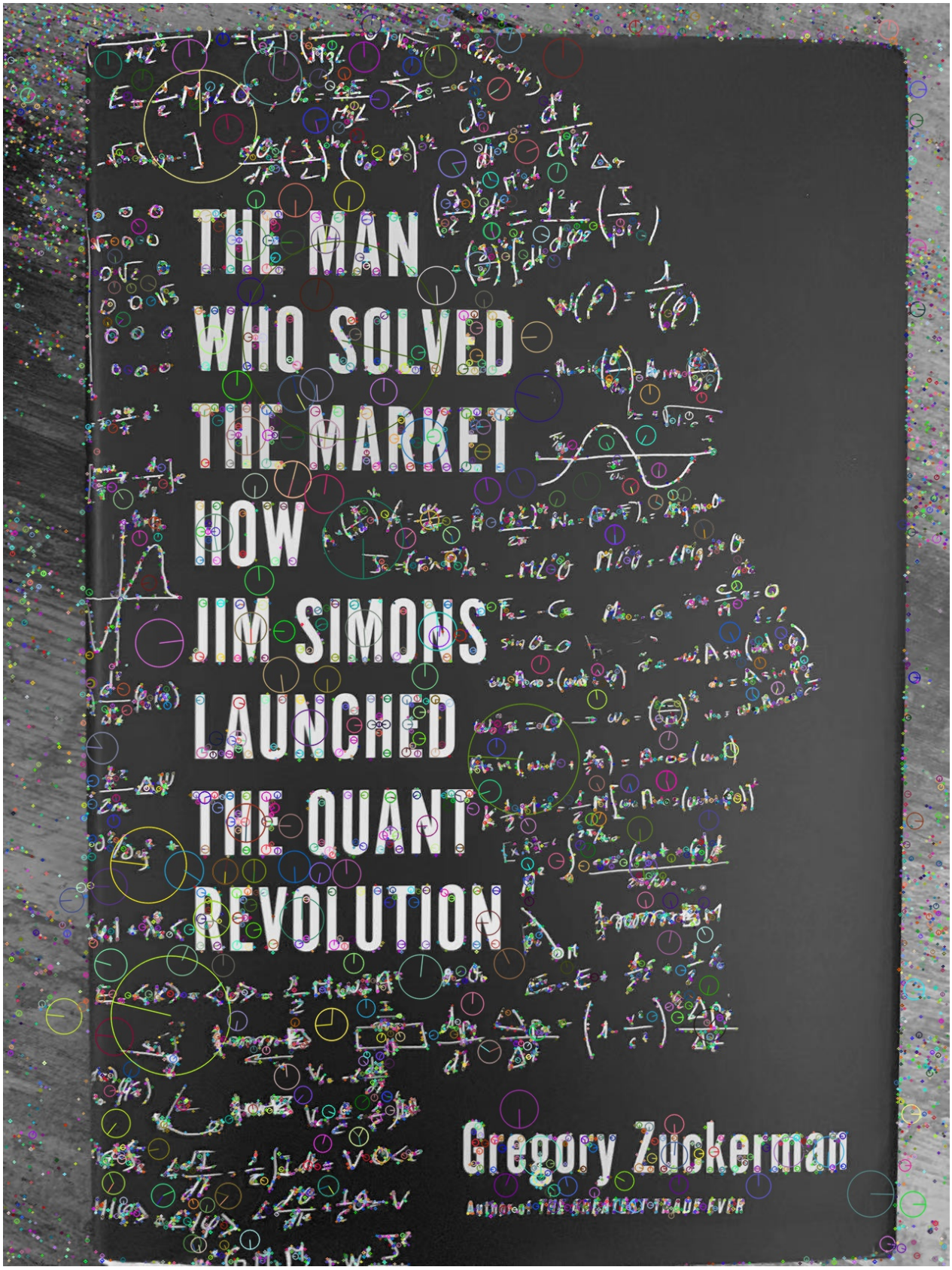
First step in locating objects in images is to detect SIFT keypoints and descriptors of all source objects images and target images. We do this via the $detectAndCompute$ function of class $SIFT\_create$. We print the keypoints with their directions and magnitudes in color, over the image in grayscale for better identification.
**Images:**



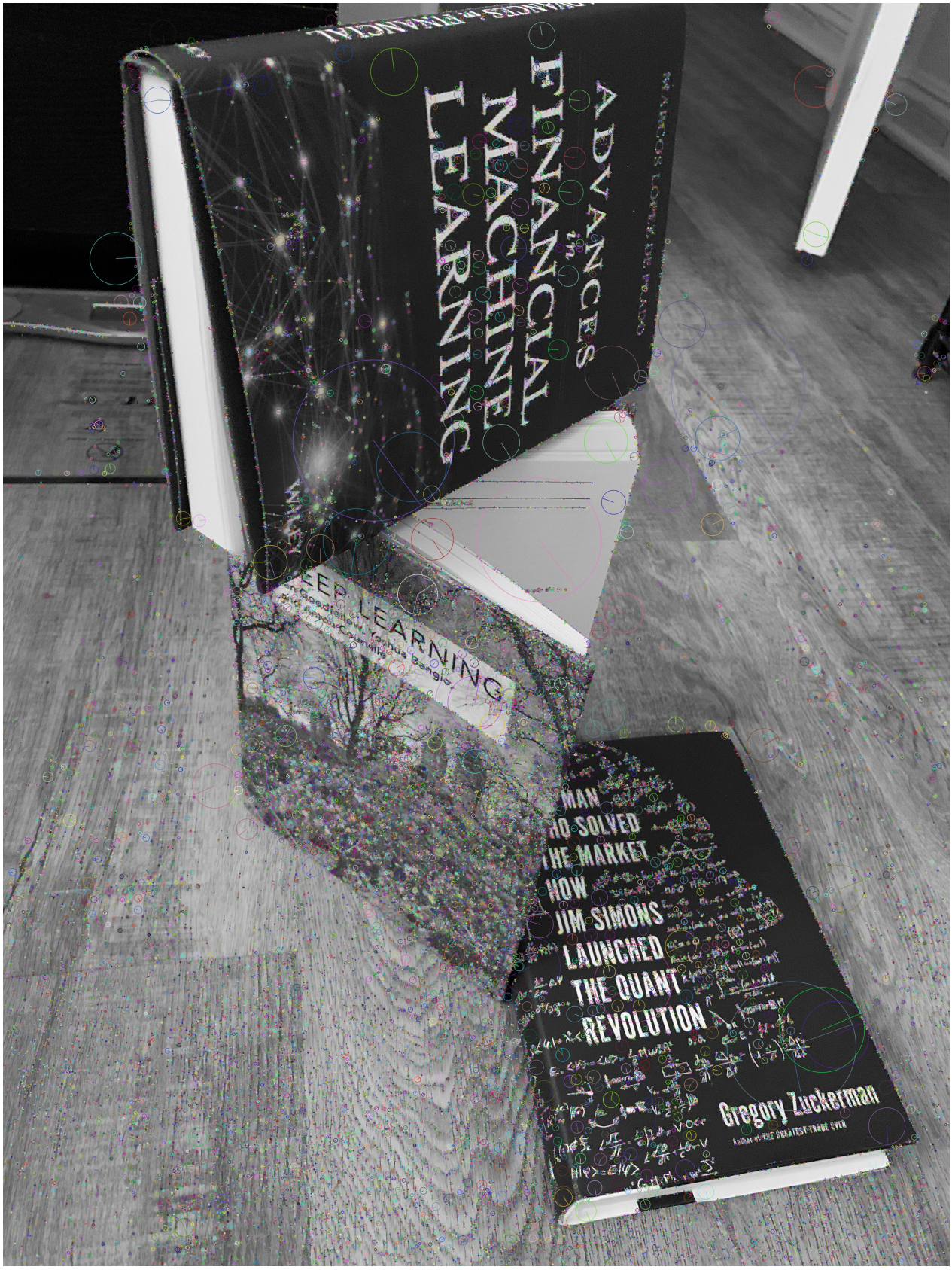

SIFT features for src_0.jpg




SIFT features for src_1.jpg




SIFT features for src_2.jpg




SIFT features for dst_0.jpg


SIFT features for dst_1.jpg

**Total Number of Features Detected:**

```
Anaconda Prompt

(cv_is_fun) C:\Users\admin\CS 677 ACV>python hw3.py
--------------------------------------------------------------------------------
Features found in image src_0.jpg=  5256
Features found in image src_1.jpg=  42472
Features found in image src_2.jpg=  13002
Features found in image dst_0.jpg=  49978
Features found in image dst_1.jpg=  10638
```

Number of Features.jpg

(b) **Displaying Top 20 Matches:**

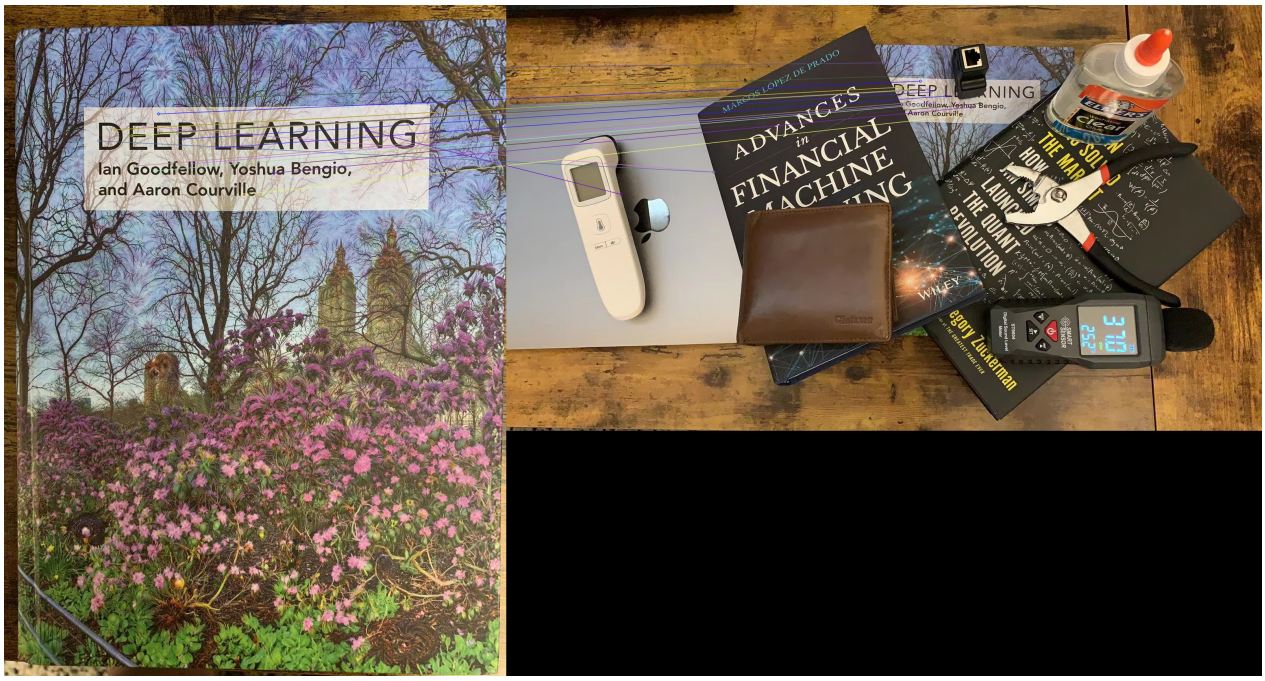
Once we have our feature points, we can begin creating matches. We use $knnMatch$ function of brute force matcher class $bfMatcher$. This functions takes descriptors of one image and finds the k most similar keypoints in the second image.
We take k=2 so that we can run the ratio test on the images. The ratio test disregards matches where the 2 points returned have similar distances to the original descriptor. When distance is similar, it suggests that there are repetitive patterns in the images and that these keypoints can be ignored.
After filtering out the good matches, we sort them and display the top 20 matches.
**The images showing top 20 matches:**


Top 20 matches between dst_0.jpg and src_0.jpg

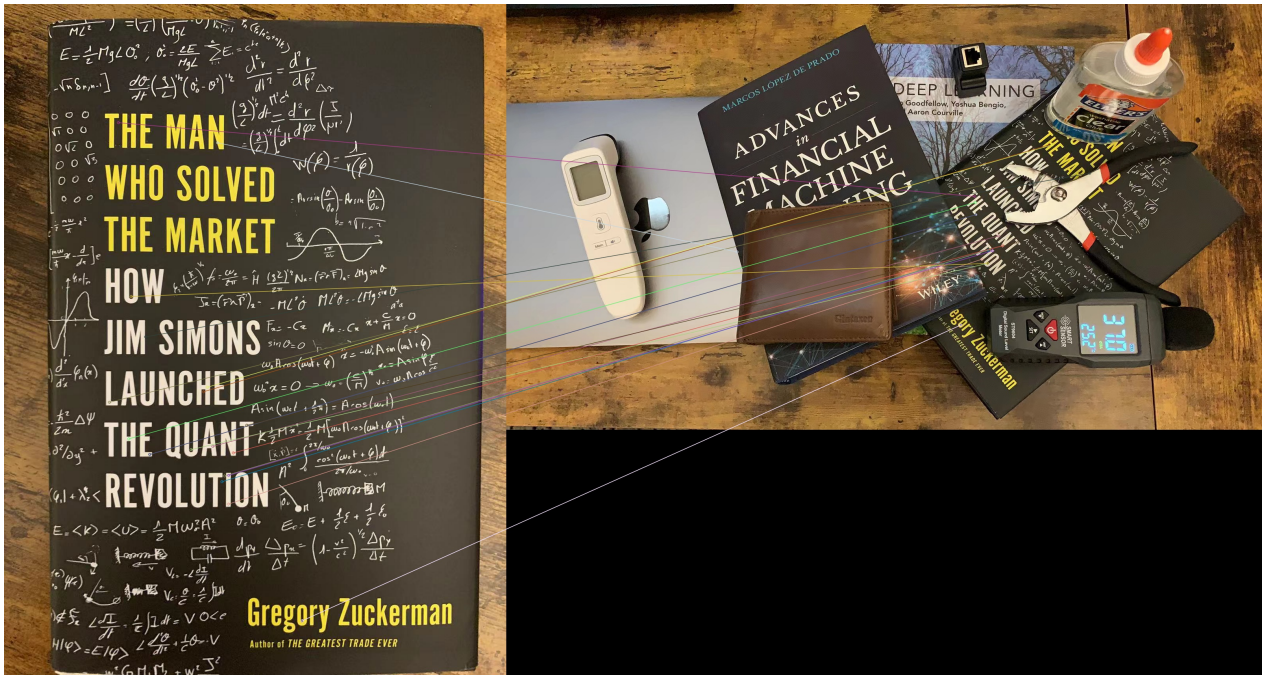

Top 20 matches between dst_0.jpg and src_1.jpg


Top 20 matches between dst_0.jpg and src_2.jpg


Top 20 matches between dst_1.jpg and src_0.jpg




Top 20 matches between dst_1.jpg and src_1.jpg



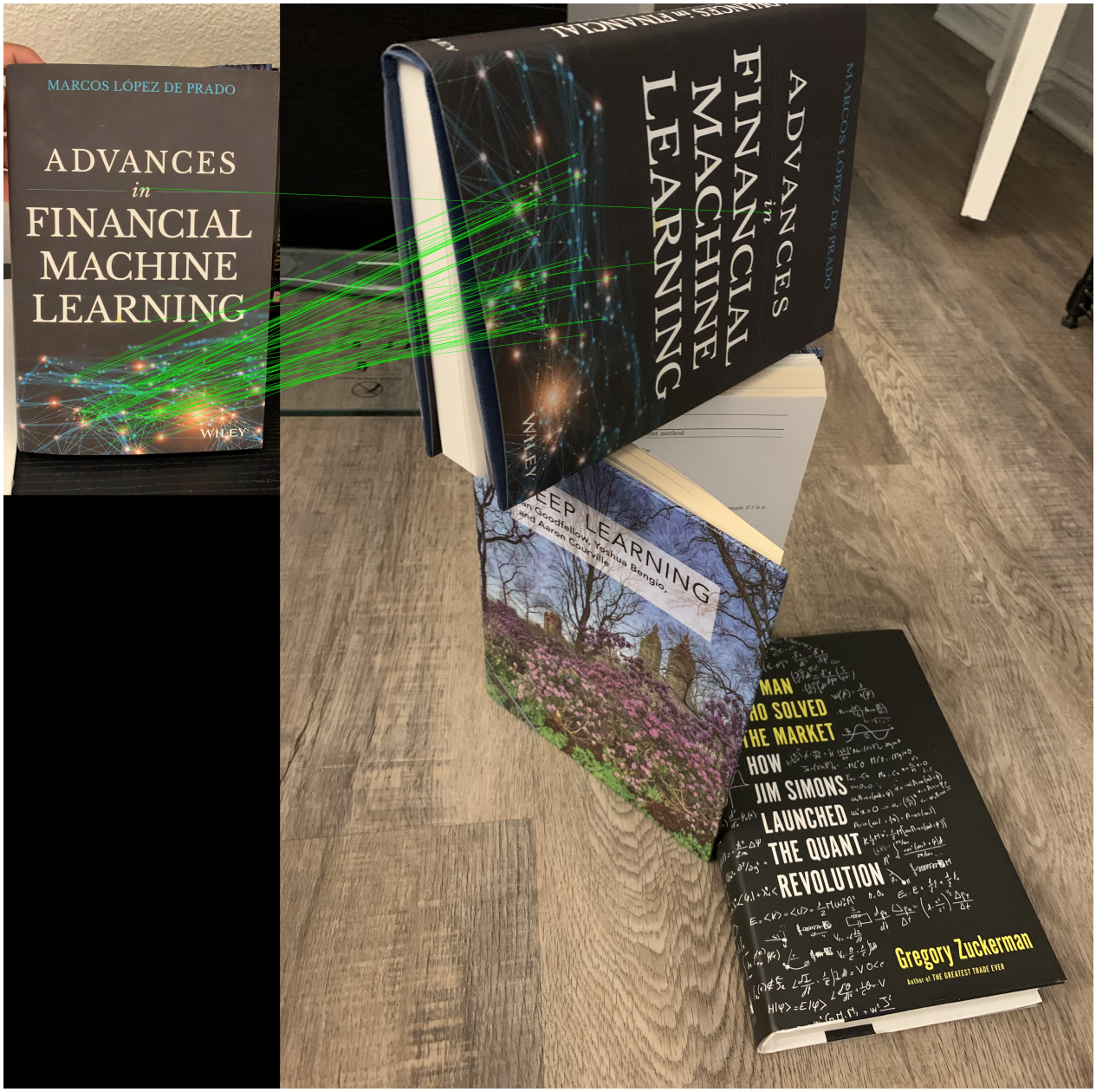

Top 20 matches between dst_1.jpg and src_2.jpg

**Total Number of matches found for each pair:**

```
Anaconda Prompt

(cv_is_fun) C:\Users\admin\CS 677 ACV>python hw3.py
--------------------------------------------------------------------------
Features found in image src_0.jpg=  5256
Features found in image src_1.jpg=  42472
Features found in image src_2.jpg=  13002
Features found in image dst_0.jpg=  49978
Features found in image dst_1.jpg=  10638

------------------------------Target Image: dst_0.jpg | Object Image: src_0.jpg------------------------------
Total Matches found = 398

------------------------------Target Image: dst_1.jpg | Object Image: src_0.jpg------------------------------
Total Matches found = 721

------------------------------Target Image: dst_0.jpg | Object Image: src_1.jpg------------------------------
Total Matches found = 416

------------------------------Target Image: dst_1.jpg | Object Image: src_1.jpg------------------------------
Total Matches found = 936

------------------------------Target Image: dst_0.jpg | Object Image: src_2.jpg------------------------------
Total Matches found = 1784

------------------------------Target Image: dst_1.jpg | Object Image: src_2.jpg------------------------------
Total Matches found = 1485

--------------------------------------------------------------------------
```

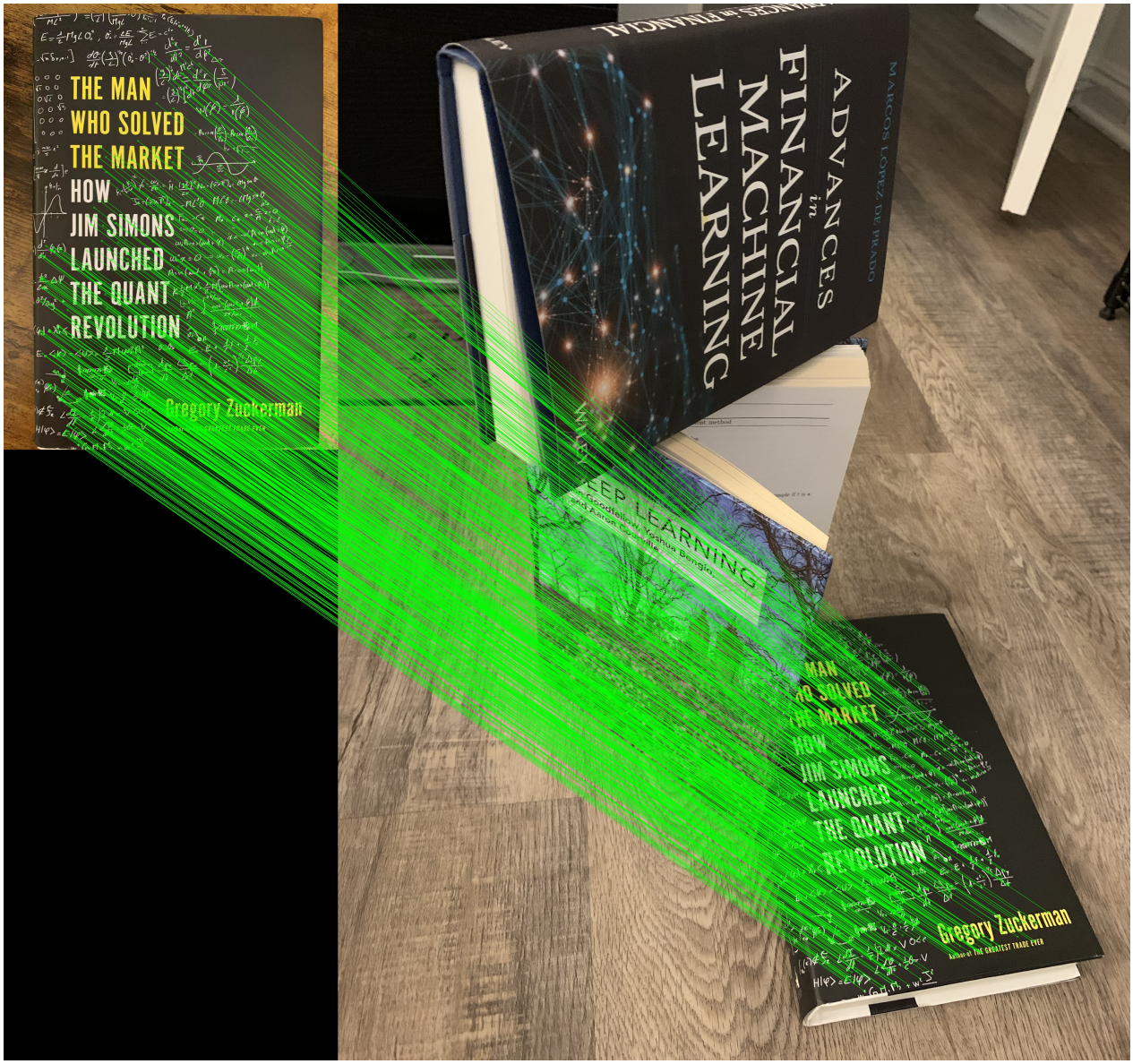
Total number of matches

(c) After we have a sufficient number of good matches between a pair of images, we can use the $findHomography$ function with RANSAC method to estimate the values of homography matrix, which can transform points from one image to the other.
We use this homography matrix to plot the corner points of the objects in the target image. The $findHomography$ function also returns a mask telling us which of the points in good matches are inliner matches.
**Image pairs showing all inliner matches (in green) in target image:**




Inliner matches between dst_0.jpg and src_0.jpg

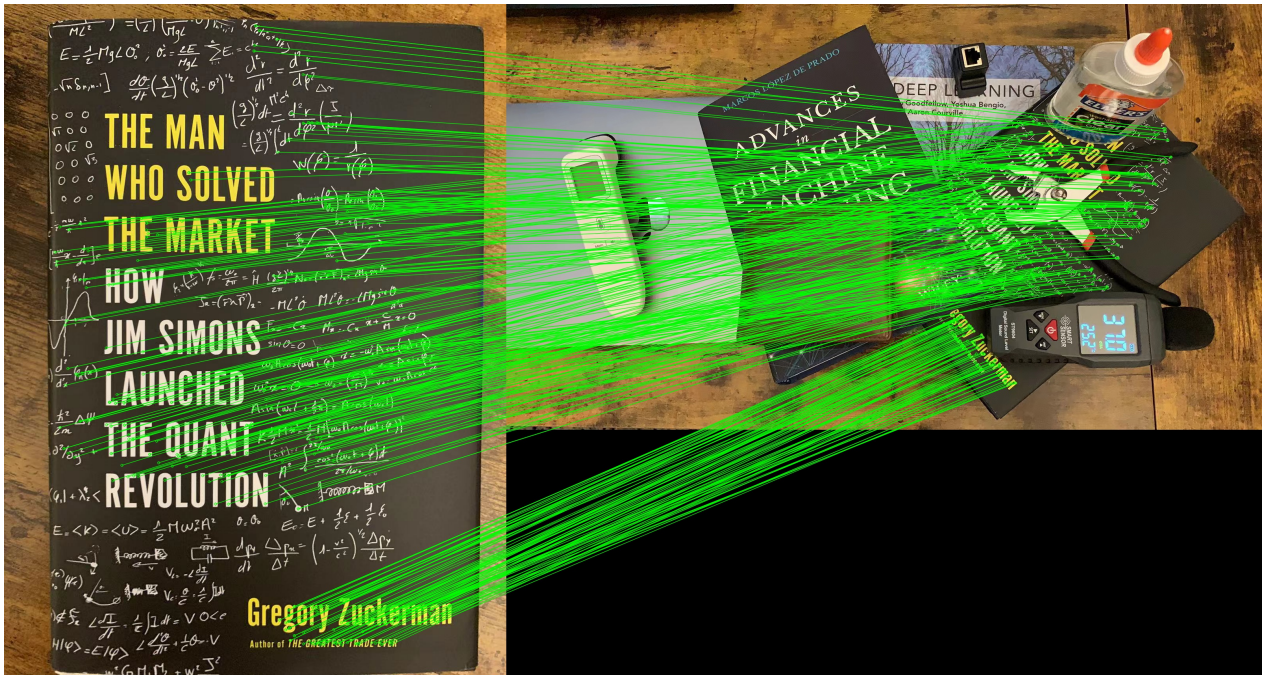

Inliner matches between dst_0.jpg and src_1.jpg




Inliner matches between dst_0.jpg and src_2.jpg


Inliner matches between dst_1.jpg and src_0.jpg


Inliner matches between dst_1.jpg and src_1.jpg




Inliner matches between dst_1.jpg and src_2.jpg

**Total Number of inliner matches for each pair:**

```
Anaconda Prompt

(cv_is_fun) C:\Users\admin\CS 677 ACV>python hw3.py
--------------------------------------------------------------------------
Features found in image src_0.jpg=  5256
Features found in image src_1.jpg=  42472
Features found in image src_2.jpg=  13002
Features found in image dst_0.jpg=  49978
Features found in image dst_1.jpg=  10638

------------------------------Target Image: dst_0.jpg | Object Image: src_0.jpg------------------------------
Total Matches found = 398
Inliner matches found =  63

------------------------------Target Image: dst_1.jpg | Object Image: src_0.jpg------------------------------
Total Matches found = 721
Inliner matches found =  267

------------------------------Target Image: dst_0.jpg | Object Image: src_1.jpg------------------------------
Total Matches found = 416
Inliner matches found =  45

------------------------------Target Image: dst_1.jpg | Object Image: src_1.jpg------------------------------
Total Matches found = 936
Inliner matches found =  467

------------------------------Target Image: dst_0.jpg | Object Image: src_2.jpg------------------------------
Total Matches found = 1784
Inliner matches found =  791

------------------------------Target Image: dst_1.jpg | Object Image: src_2.jpg------------------------------
Total Matches found = 1485
Inliner matches found =  429

--------------------------------------------------------------------------
```

Total number of inliner matches

We also use the homography matrix to project all the descriptors in the object images to create predicted points. We then compute the distance between these predicted points and the destination keypoints. We then plot the matches with the top 10 lowest error. I have used L2 norm distance to calculate distance.

**Image pairs with top 10 matches with lowest error predictions:**

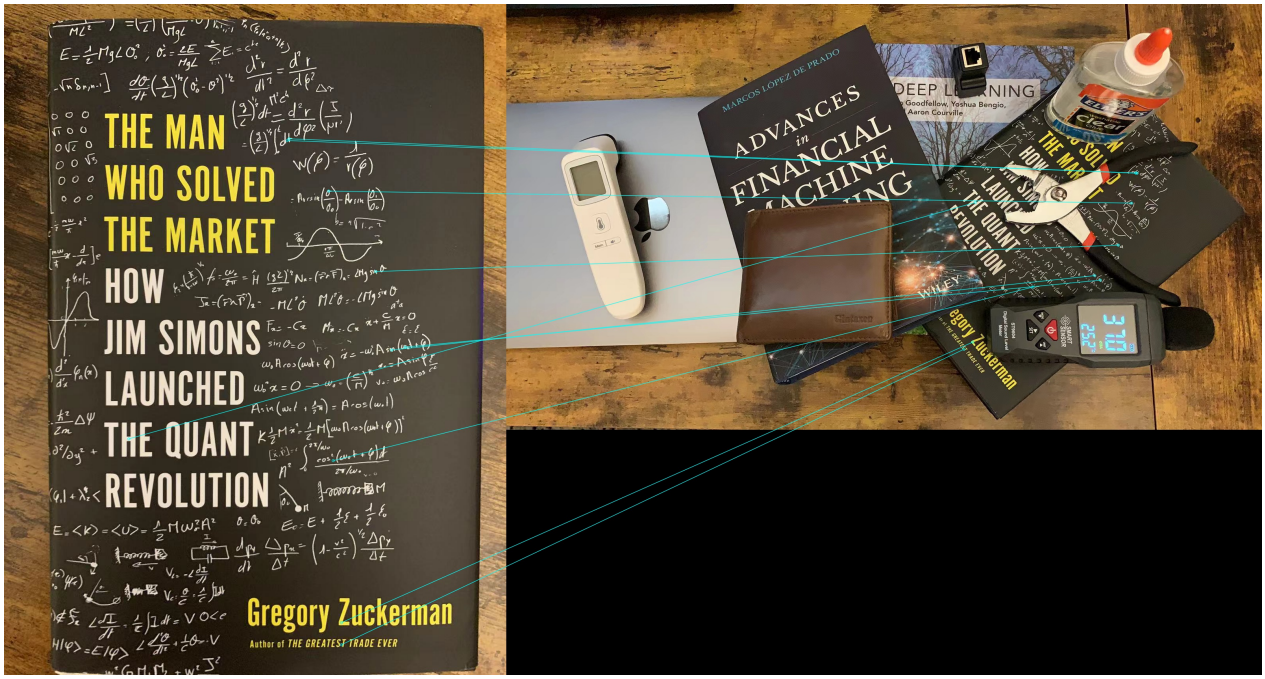

Top 10 low error matches between dst_0.jpg and src_0.jpg


Top 10 low error matches between dst_0.jpg and src_1.jpg


Top 10 low error matches between dst_0.jpg and src_2.jpg


Top 10 low error matches between dst_1.jpg and src_0.jpg


Top 10 low error matches between dst_1.jpg and src_1.jpg




Top 10 low error matches between dst_1.jpg and src_2.jpg

(d) **Finally we print the Homography matrices:**

```
------------------------------Target Image: dst_0.jpg | Object Image: src_0.jpg------------------------------
Homography Matrix=
[[ 3.15835656e-01 -1.17183536e+00  2.26511079e+03]
 [ 1.29630797e+00  4.94667920e-02 -1.03835320e+01]
 [ 1.75849702e-04 -2.90437325e-04  1.00000000e+00]]

------------------------------Target Image: dst_1.jpg | Object Image: src_0.jpg------------------------------
Homography Matrix=
[[ 3.44232095e-01  3.65555723e-03  4.60027387e+02]
 [-2.17788827e-01  2.91519090e-01  2.00871025e+02]
 [-5.77197068e-05 -1.71671774e-04  1.00000000e+00]]

------------------------------Target Image: dst_0.jpg | Object Image: src_1.jpg------------------------------
Homography Matrix=
[[ 7.39033116e-01  1.47318074e-01  7.04147854e+02]
 [ 3.81015525e-01  1.12858294e+00  1.34817433e+03]
 [-9.25106606e-05  1.66377211e-04  1.00000000e+00]]

------------------------------Target Image: dst_1.jpg | Object Image: src_1.jpg------------------------------
Homography Matrix=
[[ 4.52668896e-01 -8.94384698e-02  8.84474806e+02]
 [ 1.46577266e-02  3.93945457e-01  7.54689954e+01]
 [ 1.36103533e-05 -6.98324975e-05  1.00000000e+00]]

------------------------------Target Image: dst_0.jpg | Object Image: src_2.jpg------------------------------
Homography Matrix=
[[ 8.26791162e-01 -1.34892249e-01  1.41729499e+03]
 [ 4.52960623e-02  1.70658512e-01  2.36970856e+03]
 [ 4.51033790e-05 -1.80326010e-04  1.00000000e+00]]

------------------------------Target Image: dst_1.jpg | Object Image: src_2.jpg------------------------------
Homography Matrix=
[[ 2.12750881e-01 -3.92792248e-01  1.48607985e+03]
 [ 3.10834871e-01  2.58270148e-01  8.98876562e+01]
 [-6.63825064e-05 -3.49581089e-05  1.00000000e+00]]

--------------------------------------------------------------------------------
```

Homography Matrices between image pairs

# 2 Conclusion

- Using this method, we can successfully identify objects in the target images.
- This method should work equally well on all examples. This is because the SIFT features that we generate are invariant of scale, rotation or translation of images, making it a robust method for calculating features.
- SIFT descriptors are based on histogram of gradients. Images need to be pre-processed and gradients need to be calculated. All these processes take time and thus this method is very time intensive.
- We could transform the entire object image and overlay it over the target image using the homography matrix.
  One example can be as follows:
  **Example:**


Example showing inliner points and object boundary created using object corners overlaid over the target image.