CSCI 677: Advanced Computer Vision, HW3

Matin Barekatain 1385073716

October 4, 2021

• Data

There are 3 source images and 2 destination images in our dataset that have been shown below. Our goal is to find the matches of source and destination objects through homography by benefiting from the SIFT features and by applying the RANSAC method.

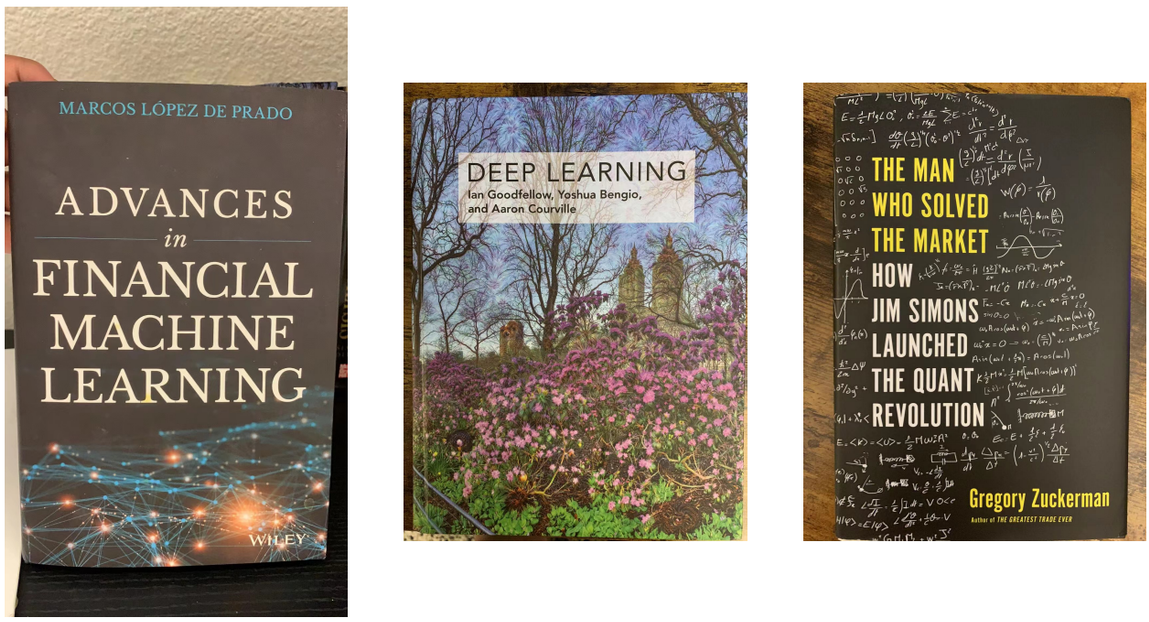


Figure 1: Source Images

Figure 2: Destination Images

• SIFT Extraction

What we initially extract are the SIFT features of all these images after we made them gray scale.

SIFT Extraction

```
def sift_generator(img, img_id):
```

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
sift = cv2.SIFT_create()
keypoints, descriptors = sift.detectAndCompute(img, None)
print("Number of SIFT features in {}: {}".format(img_id, len(keypoints)))

# sketching the detected key points
sift_image = cv2.drawKeypoints(gray, keypoints, np.array([]),
flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

# save the image
cv2.imwrite("Results/"+img_id+"_general_sift.jpg", sift_image)

return keypoints, descriptors
```

This gives us both the locations and directions of the detected SIFT features overlaid on the image. The total number of detected features for each image has been mentioned in the captions.

Figure 3: SIFT Features of Source Images, Total Number of Findings (left to right): **5252, 42472, 13002**.

Figure 4: SIFT Features of Destination Images, Total Number of Findings: **49978, 10638**.

• Homogeneous Matching

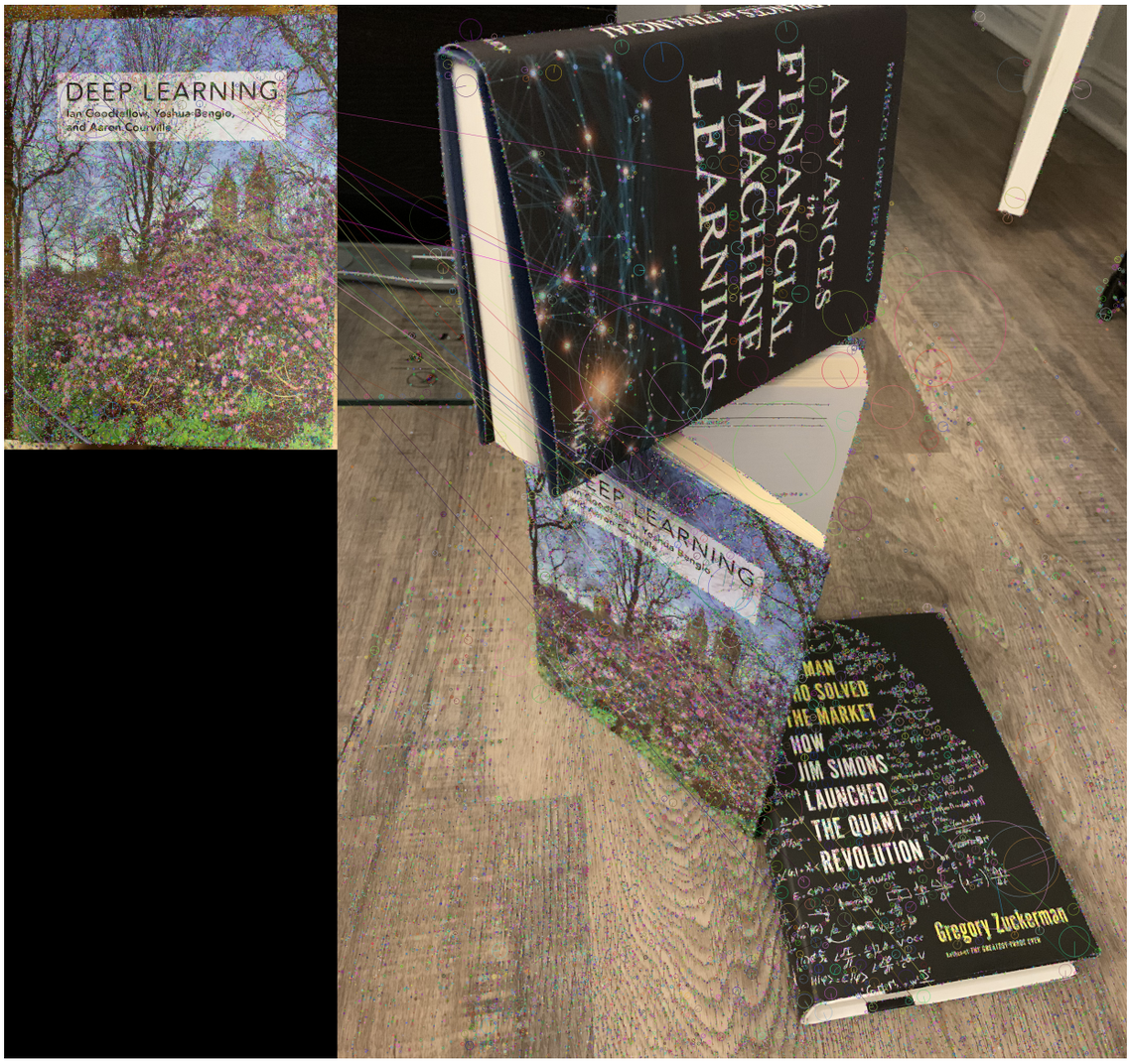
At this step, we run our brute-force matcher to find matched features between source and destination images. We find the good matches with respect to the 70% distance rule inspired by OpenCV documents. We sort the "good" items based on the distances of the elements in the "good" list as shown in the code below.

```
def matcher(desc_s, desc_d):
    bf = cv2.BFMatcher()

    matches = bf.knnMatch(desc_s, desc_d, k=2)
    print("Number of Matches: {}".format(len(matches)))

    # Apply ratio test
    good = []
    for m,n in matches:
        if m.distance < 0.7*n.distance:
            good.append(m)

    print("Number of Good Matches: {}".format(len(good)))
    good = sorted(good, key = lambda x:x.distance)

    return good
```

After collecting the "good" matches and prioritizing the top-20 values, the following result is obtained before performing homography through RANSAC. We also show statistics regarding the number of matches that we consider as "good". The number of matches have been mentioned in the captions.

src_0 and dst_0: 132 Matches

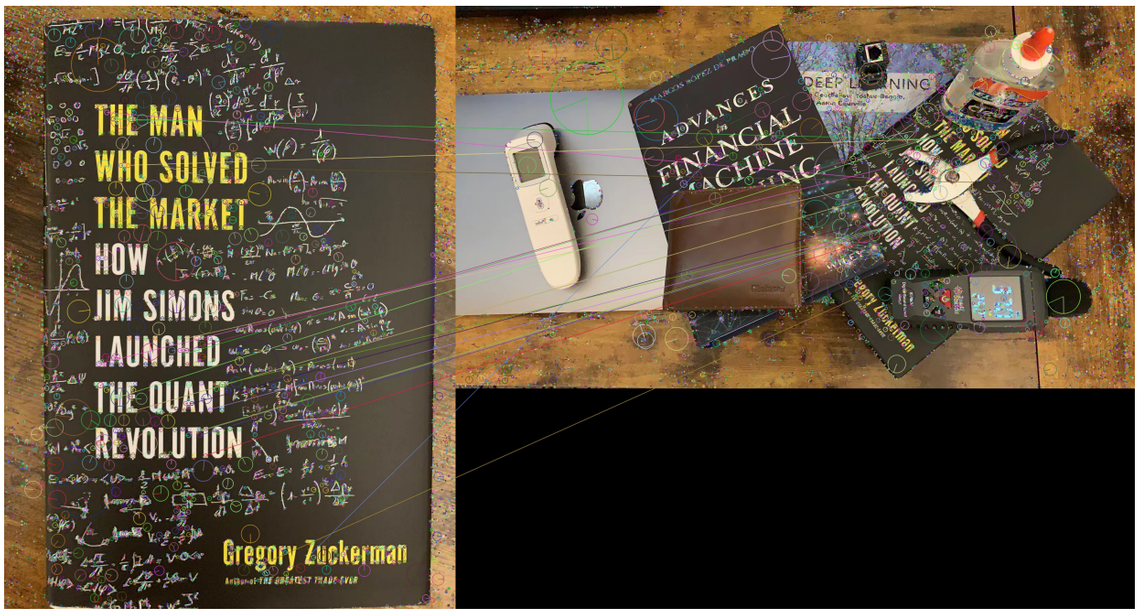
src_0 and dst_1: 388 Matches



src_1 and dst_0: 53 Matches

src_1 and dst_1: 484 Matches

src_2 and dst_0: 912 Matches



src_2 and dst_1: 843 Matches

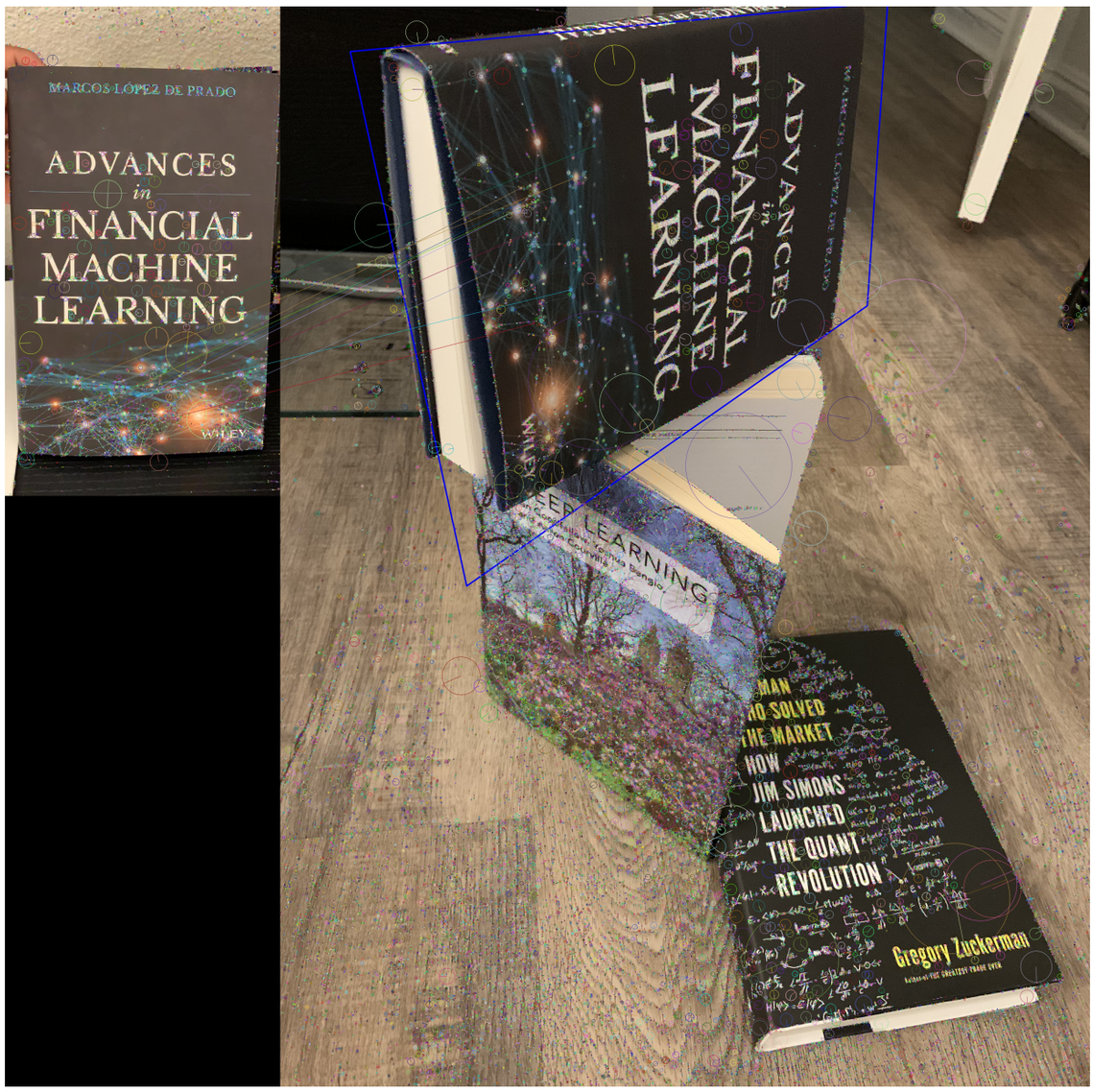
• Homography with RANSAC

We bring our "homographer" function in the following that is capable of performing the RANSAC method to find suitable masks for the keypoints previously detected for each pair of images (i.e., source and destination), and matching them.

Applying Strategies (e.g. Color)

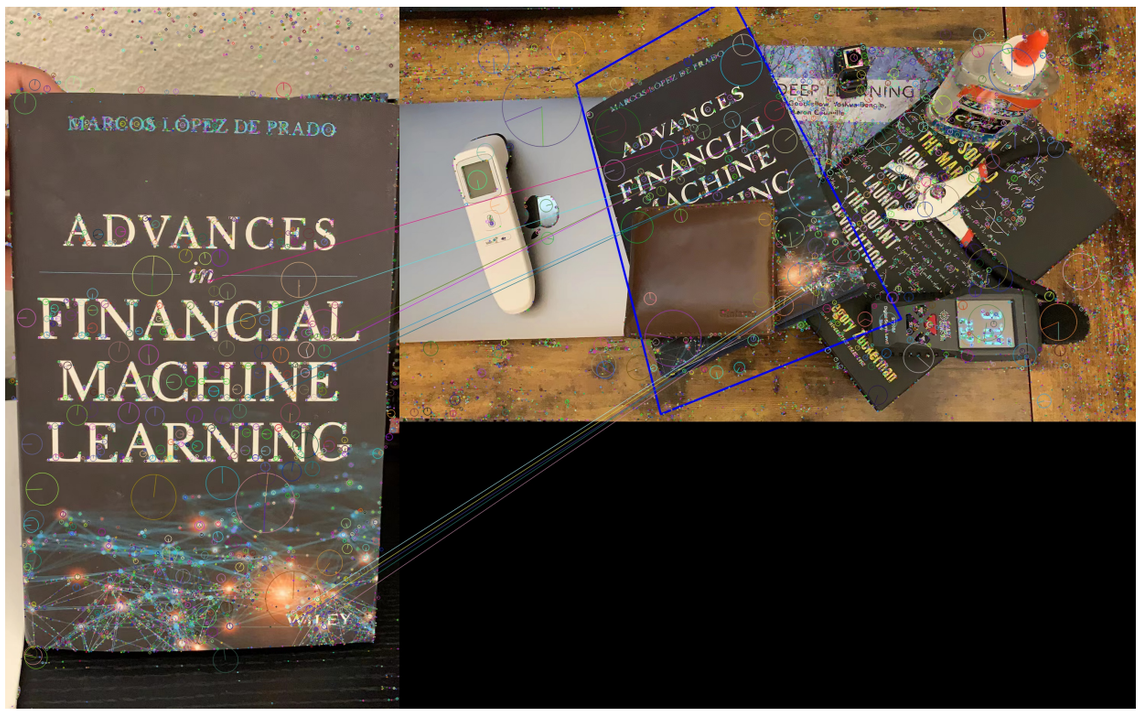
```
def homographer(kp1, kp2, src_img, dst_img, good):
  MIN_MATCH_COUNT = 1

  if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([kp1[m.queryIdx].pt for m in good]).reshape(-1,1,2)
    dst_pts = np.float32([kp2[m.trainIdx].pt for m in good]).reshape(-1,1,2)

    M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC,5.0)
    matchesMask = mask.ravel().tolist()
    print("Number of RANSAC Matches: {}".format(sum(matchesMask)))

    h,w,d = src_img.shape
    pts = np.float32([ [0,0],[0,h-1],[w-1,h-1],[w-1,0] ]).reshape(-1,1,2)
    dst = cv2.perspectiveTransform(pts,M)
    dst_img = cv2.polylines(dst_img,[np.int32(dst)],True,255,3, cv2.LINE_AA)

  else:
      print( "Not enough matches are found - {}/{}"
                                  .format(len(good), MIN_MATCH_COUNT))
      matchesMask = None
      sys.exit(0)

  return dst_img, matchesMask
```

The followings are the top-10 results obtained after calling the "homographer" function based on the mask values. The total number of matches has been shown in the caption of each image. These numbers obtained by summing the values of inliers, i.e., masks with values equal to one. The homography matrices have also been mentioned.



src_0 and dst_0: 30 Matches

$$M_{Homography} = \begin{bmatrix} 3.53240111e-01 & -1.20040855e+00 & 2.32732028e+03 \\ 1.38327585e+00 & 7.95813499e-02 & -5.74441769e+01 \\ 2.01640843e-04 & -2.73885390e-04 & 1.00000000e+00 \end{bmatrix}$$



src_0 and dst_1: 215 Matches

$$M_{Homography} = \begin{bmatrix} 3.30675837e-01 & 7.82275666e-03 & 4.61252183e+02 \\ -2.22041901e-01 & 2.91667614e-01 & 2.01915243e+02 \\ -7.20971283e-05 & -1.66082545e-04 & 1.00000000e+00 \end{bmatrix}$$

src_1 and dst_0: 8 Matches

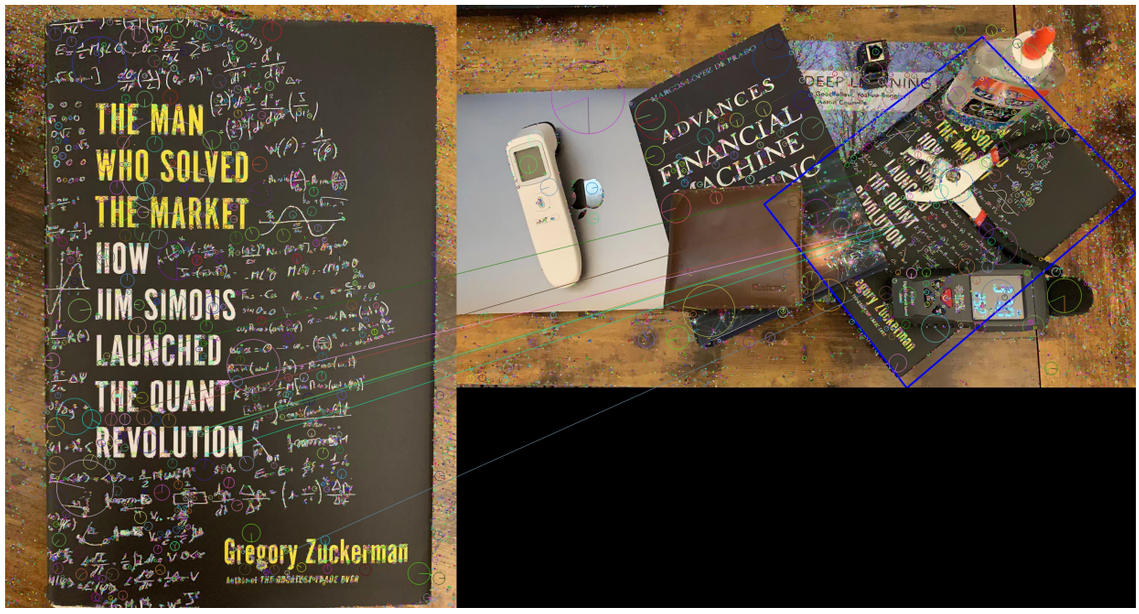
$$M_{Homography} = \begin{bmatrix} 7.43926380e-01 & 1.10002552e-01 & 6.88932735e+02 \\ 3.10514444e-01 & 9.90614301e-01 & 1.41457421e+03 \\ -9.66285161e-05 & 1.21749536e-04 & 1.00000000e+00 \end{bmatrix}$$

src_1 and dst_1: 394 Matches

$$M_{Homography} = \begin{bmatrix} 4.52496703e-01 & -8.85293698e-02 & 8.84580180e+02 \\ 1.45880486e-02 & 3.94252627e-01 & 7.54852062e+01 \\ 1.34863243e-05 & -6.88664684e-05 & 1.00000000e+00 \end{bmatrix}$$

src_2 and dst_0: 540 Matches

$$M_{Homography} = \begin{bmatrix} 8.08395751e-01 & -1.34757221e-01 & 1.42129187e+03 \\ 2.83246653e-02 & 1.69778730e-01 & 2.37060442e+03 \\ 3.86153969e-05 & -1.79704478e-04 & 1.00000000e+00 \end{bmatrix}$$



src_2 and dst_1: 378 Matches

$$M_{Homography} = \begin{bmatrix} 2.11967790e-01 & -3.92626947e-01 & 1.48599630e+03 \\ 3.10450815e-01 & 2.58174836e-01 & 9.00260746e+01 \\ -6.68827402e-05 & -3.49535751e-05 & 1.00000000e+00 \end{bmatrix}$$

• Analysis

• Overall, the method works with impressive results. The SIFT extraction helped finding the features inn each of the images, the brute-force matcher obtained the homogeneous matches between the source and destination images, and finally the homographer function selected best inlier mask matches through RANSAC so that the source objects were detected in the destination images.

• On the other hand, it would be of great importance to mention that the method did not work identically for all input samples. The main factors of the differences between the results we obtained for any of the cases demonstrated above include angle of observation, texture and geometric features of the objects in each of the images, location of objects in destination images, etc.

• The number of SIFT features increase when we have a wide spectrum of geometrical features including intersection, corners, lines, curves, etc. For instance, the Deep Learning book's cover has 42k features while the Financial Machine learning Book's cover has only 5k.

• For the general homogeneous matching cases, the angle of observation is one of the most important factors to obtain the maximum number of "good" matches. The more oblique the angle is, the less amount of matches would be founded. For example, as can be seen for the Deep Learning book, although in the second destination image, the most area of the book is covered by other objects, the number of good matches (i.e., 484) is much higher than that of the first destination image (i.e., 53). In contrast, it is not the same case for the Zuckerman book, because the mismatches caused by the obliqueness of the angle in the first destination image (that gives us a total number of 912 matches), is almost in the same order of the loss we face due to the area covered by other object in the second destination image (that leads to 843 matches).

• It is meaningless to mention the total number of all homogeneous matches without filtering the outliers, because the BFmatcher creates an object for each SIFT feature in the source image even though the distance is very large. So we reported the number of "good" matches.

• The wood patterns in the first destination image extraordinarily increased the number of SIFT features, however, since there was no similar patterns in any of the source images, they easily were filtered out as outliers even through the homogeneous match findings.

• SIFT features that belong to texts are usually better discriminators in finding matches. This again can be inspected for the objects most of which are covered by other objects in destination images. For instance, text SIFT features helped finding the Deep Learning book in the second destination image neglecting the fact the majority portion of the book is covered.

• The more homogeneous matches we find using the brute-force matcher, the easier the RANSAC method can detect the source object in the destination image. To exemplify, the Zuckerman book has 912 homogeneous good matches in the second destination image. After operating the RANSAC method, this number was shrunk to 540, yet made it easy for the algorithm to find the book object.