CSCI677 Homework Assignment #3
Ye Joo Park (USC# 1128-6851-51)

Brief Description of the Program

The python script takes two images (a target object and the object within other objects) at a time and first finds keypoints and descriptors with OpenCV's SIFT module in xfeatures2d (I'm using OpenCV 3 with opencv_contrib).

Using the keypoints, the program matches keypoints between two images using brute-force matcher. The matches are then sorted by distance (smaller is better). We will find tomography using OpenCV's findHomography() and apply perspective transform to get better match results.

The intermediate steps are self-explanatory through the screenshots below.

Python/OpenCV Version:

```
Python version: 3.5.4
OpenCV version: 3.0.0-dev
```

1. Detected features overlaid in images

image_1.jpg (3238 keypoints)

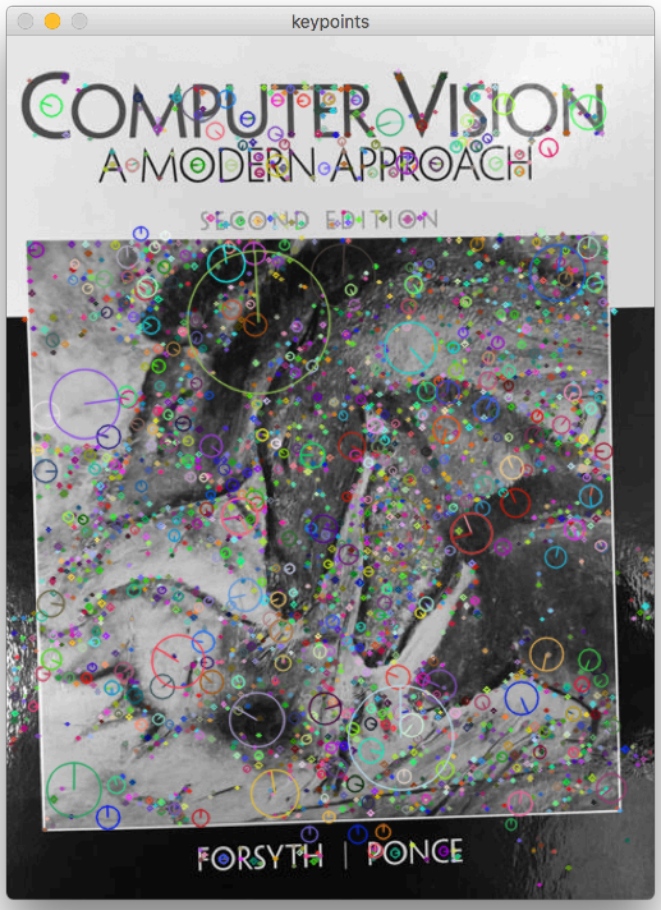


image_2.jpg (2830 keypoints)

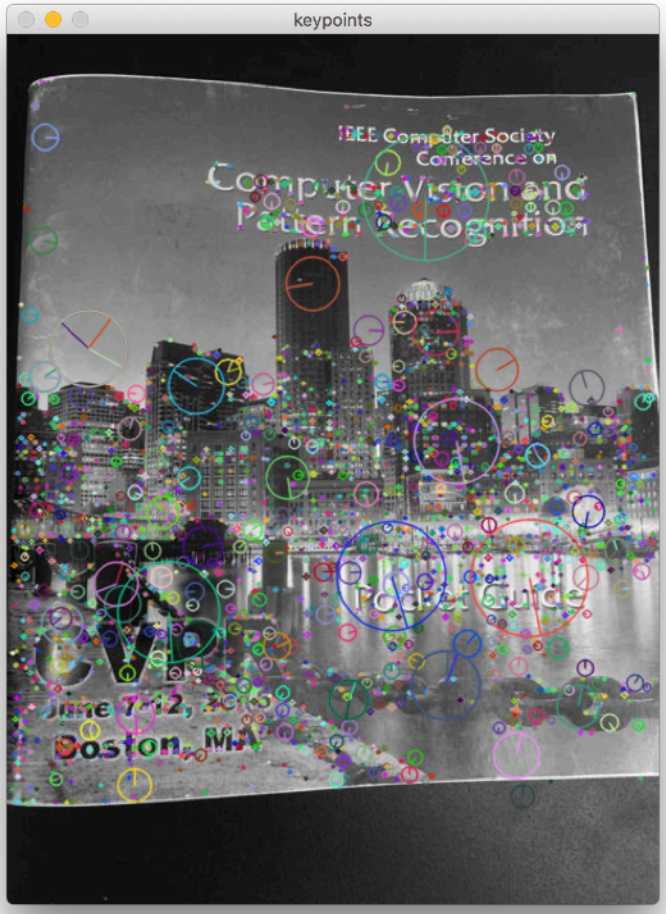


image_3.jpg (1721 keypoints)

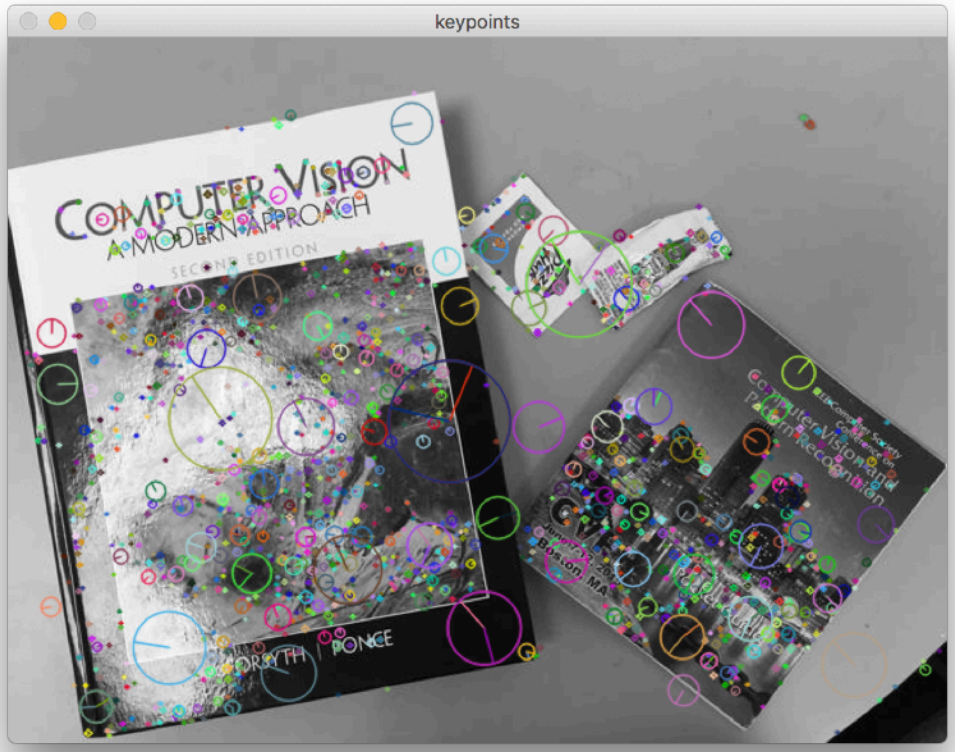


image_4.jpg (1612 keypoints)

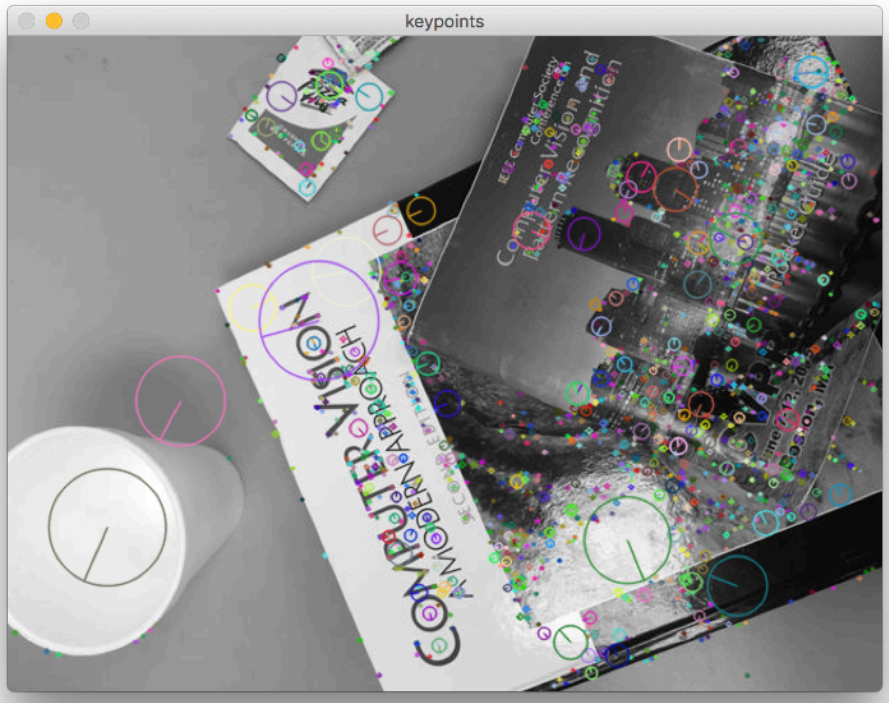
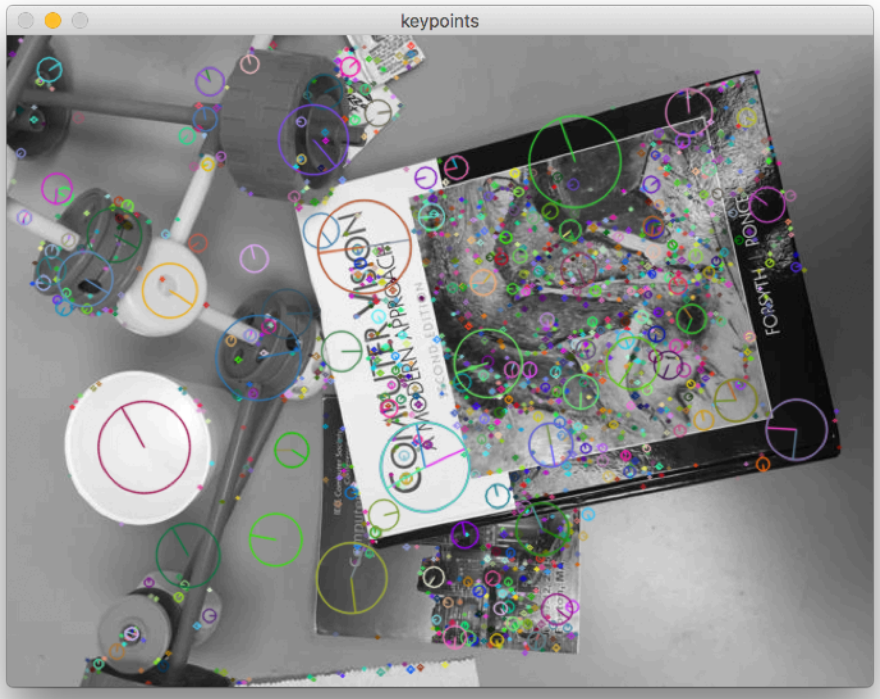


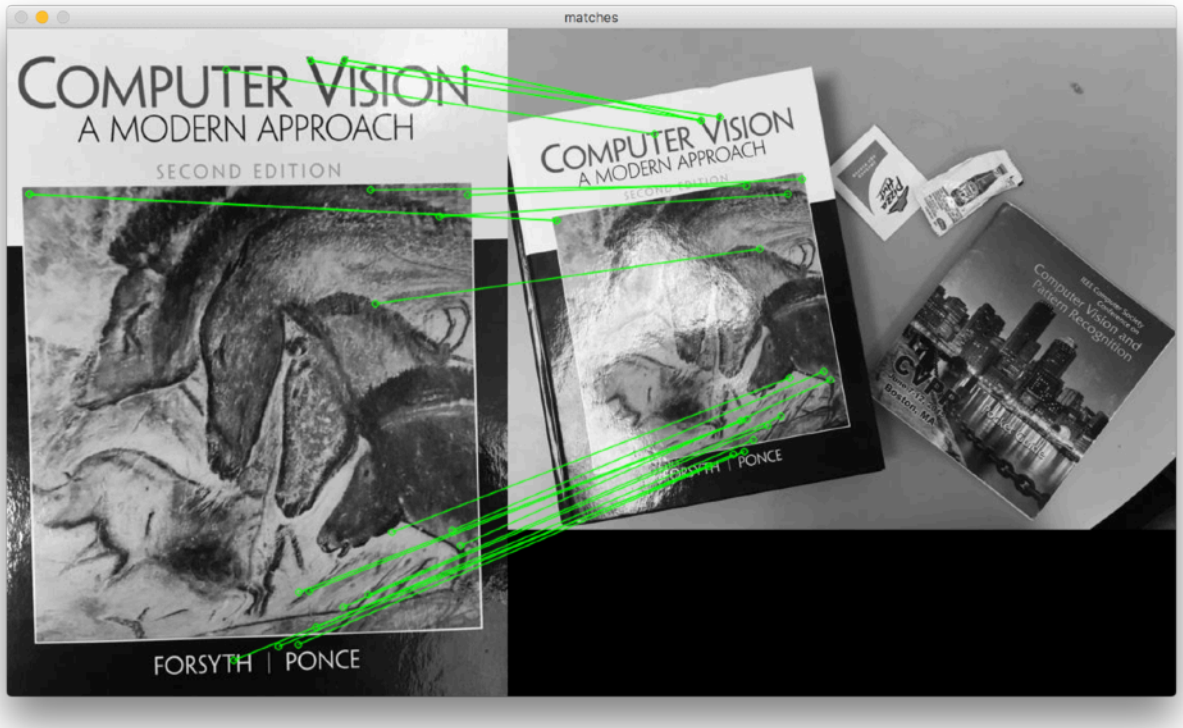
image_5.jpg (1660 keypoints)



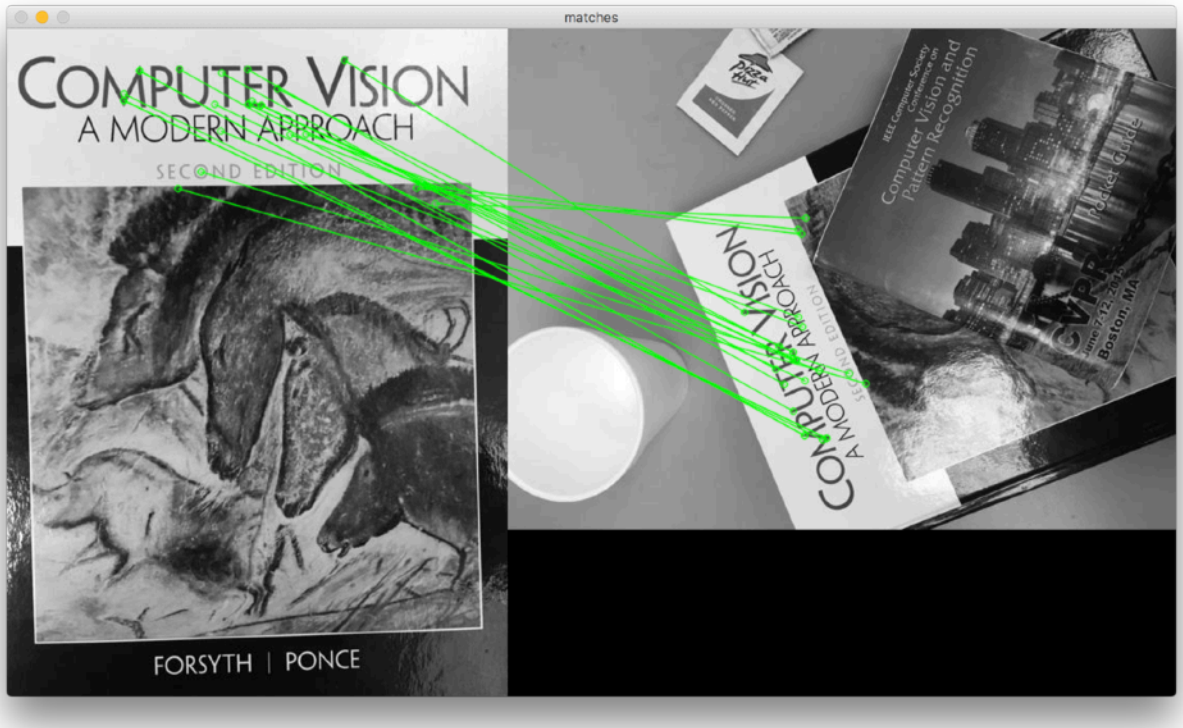
2. Top 20 scoring matches found by the matcher before RANSAC

A “good” match is defined by the number of matches filtered by distance shorter than 0.7 (p1.distance < 0.7 * p2.distance).

Between image_1.jpg and image_3.jpg (409 “good” matches)

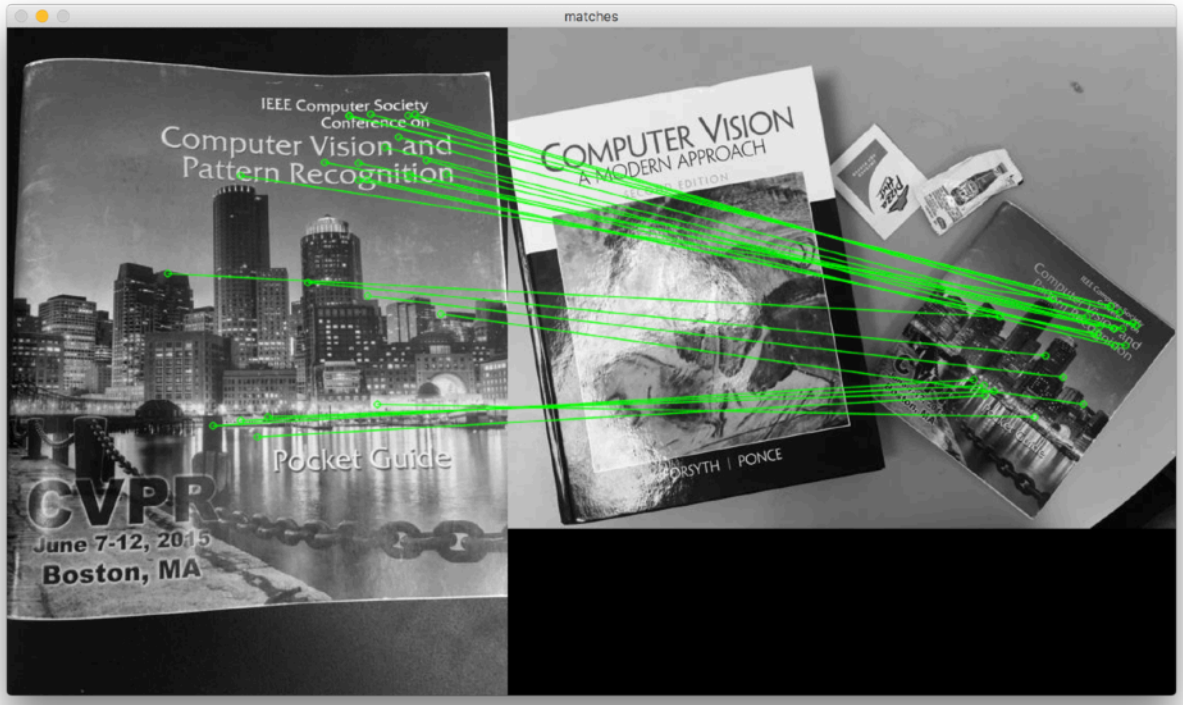


Between image_1.jpg and image_4.jpg (235 “good” matches)



Between image_1.jpg and image_5.jpg (559 “good” matches)

Between image_2.jpg and image_3.jpg (199 “good” matches)

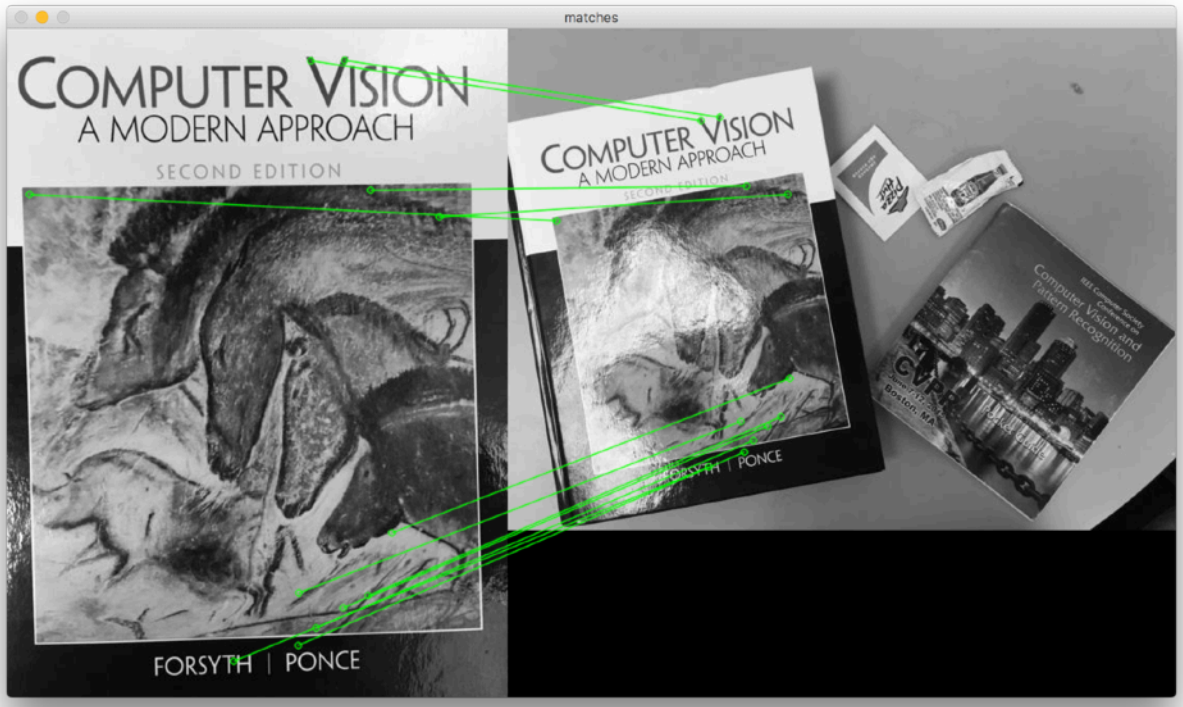


Between image_2.jpg and image_4.jpg (213 “good” matches)

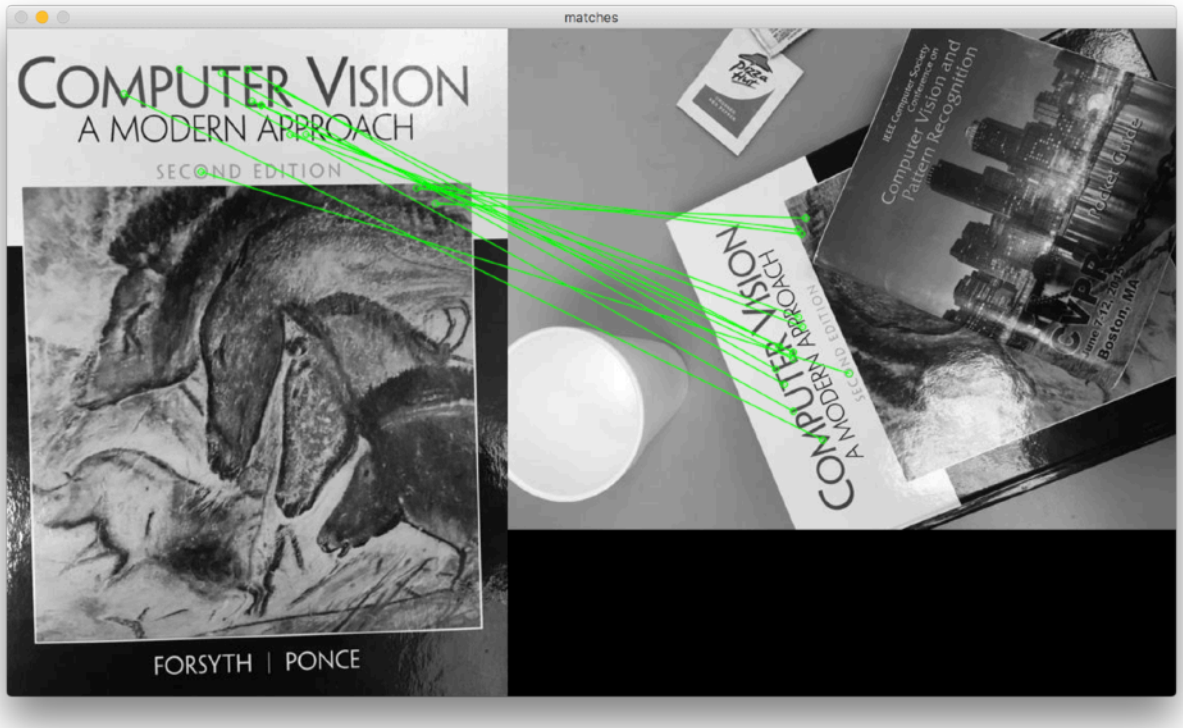
Between image_2.jpg and image_5.jpg (48 “good” matches)

3. **<u>Top 10 (or more) matches that are found after homography the has been computed Total numbers consistent with the computed homography</u>**

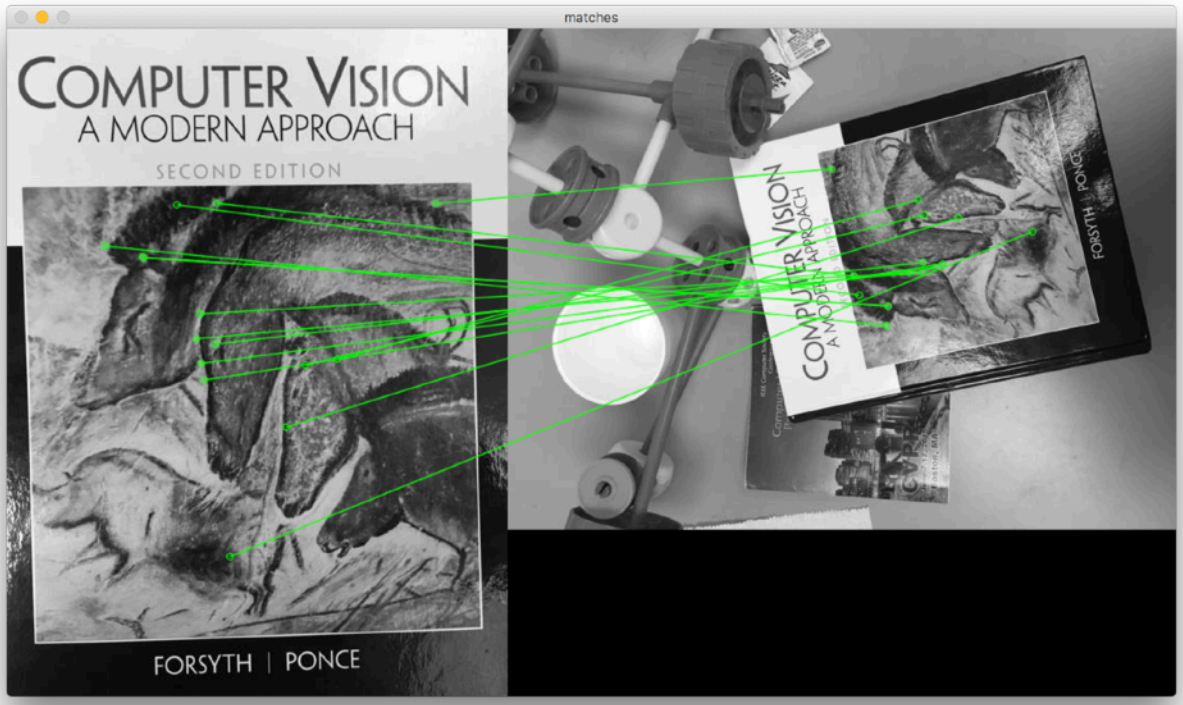
Between image_1.jpg and image_3.jpg (471 matches consistent with the computed homography)



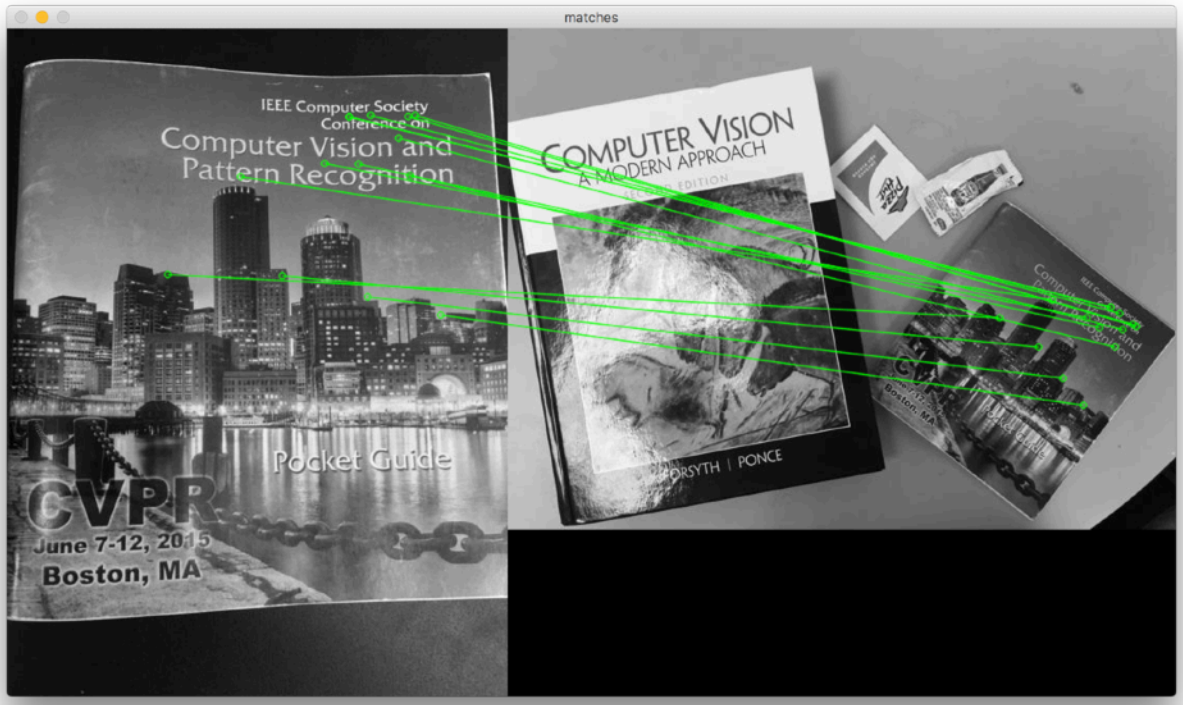
Between image_1.jpg and image_4.jpg (56 matches consistent with the computed homography)



Between image_1.jpg and image_5.jpg (620 matches consistent with the computed homography)

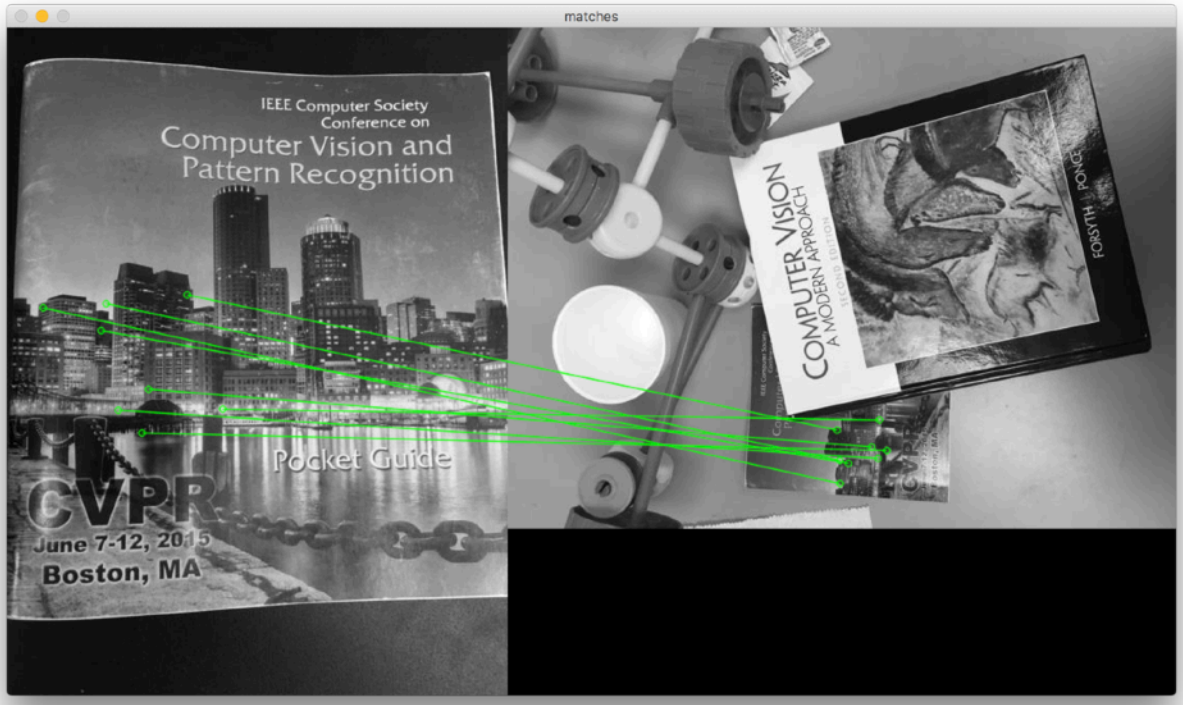


Between image_2.jpg and image_3.jpg (80 matches consistent with the computed homography)



Between image_2.jpg and image_4.jpg (262 matches consistent with the computed homography)

Between image_2.jpg and image_5.jpg (36 matches consistent with the computed homography)



4. Computed homography matrix

Between image_1.jpg and image_3.jpg

```
[[  5.36789257e-01   7.08664404e-02   2.28072812e+01]
 [ -1.03199490e-01   5.28575308e-01   9.99723986e+01]
 [ -5.95973127e-05  -6.30097760e-05   1.00000000e+00]]
```

Between image_1.jpg and image_4.jpg

```
[[ -2.58228022e-01   5.39890542e-01   2.93190356e+02]
 [ -6.09359176e-01  -2.98425766e-01   4.76411238e+02]
 [  1.06720424e-05  -1.78286153e-04   1.00000000e+00]]
```

Between image_1.jpg and image_5.jpg

```
[[ -1.17724220e-01   4.04033463e-01   2.74531039e+02]
 [ -4.77147800e-01  -1.34495724e-01   3.45464041e+02]
 [ -5.39378414e-05  -1.66643250e-04   1.00000000e+00]]
```

Between image_2.jpg and image_3.jpg

```
[[  2.75785965e-01  -1.74876075e-01   4.82950420e+02]
 [  2.27550591e-01   4.27936360e-01   1.48105614e+02]
 [ -1.54613517e-04   2.27954873e-04   1.00000000e+00]]
```

Between image_2.jpg and image_4.jpg

```
[[  1.65837768e-01   6.24277208e-01   2.64310723e+02]
 [ -5.64498307e-01   2.38729130e-01   2.24570657e+02]
 [ -1.49941463e-04   9.90998307e-05   1.00000000e+00]]
```

Between image_2.jpg and image_5.jpg

```
[[ -4.40999154e-02   4.36200114e-01   2.15458649e+02]
 [ -4.53430183e-01   9.56728969e-02   4.44800608e+02]
 [ -1.83405200e-04   1.80332984e-04   1.00000000e+00]]
```

Analysis of Test Results

How well does the method work?

It works “pretty well”. The results seem to show good matches between our object image and images of multiple objects. It is surprising that the objects were well detected even with obstacles.

Does it work equally well on the different examples?

Both yes and no. The number of matches vary widely based on the position and the color intensity distribution of the objects. Objects with barriers to viewing has a much smaller number of keypoints and matches. The second textbook (“Computer Vision and Pattern Recognition” - image_2.jpg) also generally has smaller number of matches due to its smaller font size of the book title and large mix of dark gray colors.

However, the perspective transformations applied through the homography matrices derived through cv2.findHomography() show very good matches with the objects placed in different images. Therefore, I would conclude that our method works well, although not equally throughout the examples.

Why might the performance be better in one case than the other?

There may be different ways of defining performance for this exercise. Let’s first take a look at the performance in terms of matching keypoints. As stated in the previous paragraphs, obstacles to objects disable keypoints matching in the hindered areas. This is intuitive. Another reason is due to the object image’s structure. If it is an image with many notable keypoints, it would find good matches better than other low-contrast images.

In terms of runtime performance, too many keypoints will slow down the program. Most SIFT-like applications will focus on the few best matches for real-time performance anyways, making this issue less relevant.