CSCI 574: COMPUTER VISION

Homework 3

Submitted by: Govind Raghu
Email: graghu@usc.edu
USC ID: 1911265604

1:

Solution Description:

The code listed below solves the task of finding and matching SIFT features between two images. First we find all SIFT feature descriptors from the two images. Then we try to get closest match for each point by using Euclidean distance. Matches are ignored if the second best matching point score is close to the first best matching point score. Once we get a set of good matches we try to solve for Homography matrix by considering 4 random matches. We test whether this Homography matrix is correct by verifying the fraction of inliers. If it falls below a threshold the process is repeated (RANSAC). Once we find the correct Homography matrix we use this matrix to transform the SIFT points in image 1 to image 2. Finally we overlay the SIFT points from together to verify the results.

Code:

```
import cv2
import numpy as np
import random

# Load the two images to match with
img1 = cv2.imread('image_2.jpg')
img2 = cv2.imread('image_5.jpg')
print 'Loaded images..'

# Store the dimensions of images
height1, width1, depth1 = img1.shape
height2, width2, depth2 = img2.shape

# Stack two images together for final display
resimg2  = np.zeros((height1, width2, depth2),dtype='uint8')
```

```
resimg2[:height2,:width2, :] = img2
both = np.hstack((img1, resimg2))

# Detect SIFT features of the two images
sift_detector  = cv2.xfeatures2d.SIFT_create()
kp1, features1 = sift_detector.detectAndCompute(img1, None)
kp2, features2 = sift_detector.detectAndCompute(img2, None)
print 'SIFT features detected..'

numkp1 = len(kp1)
matches = []

# Find similar features b/w images and filter the matches using a threshold value
# Eucledian distance used for calculating similarity of SIFT features.

for i in range(numkp1):
        diffs = np.subtract(features2, features1[i])
        norms = np.linalg.norm(diffs, axis=1)
        sortednormidx = np.argsort(norms)
        mintwodiff = norms[sortednormidx[1]] - norms[sortednormidx[0]]

  if(mintwodiff >= 150):
        matches.append((kp1[i], kp2[sortednormidx[0]]))

print 'Found SIFT matches b/w two images..'

# RANSAC for finding Homography
N=1000
inliertreshold = 0.9

print 'Finding Homography..'

for i in range(N):

  print 'RANSAC iteration ' + str(i)

  # Solve Homography Matrix using 4 points
   rand3 = random.sample(matches, 4)

  a = np.array([ [match[0].pt[0] , match[0].pt[1] , 1 ] for match in rand3])
  b = np.array([ [match[1].pt[0] , match[1].pt[1] , 1 ] for match in rand3])

  H = cv2.solve(a, b, flags=cv2.DECOMP_SVD)
  H = H[1].T

  # Check if fraction of inliers is higher than treshold.
  fit = []
  fittreshold = 4
```

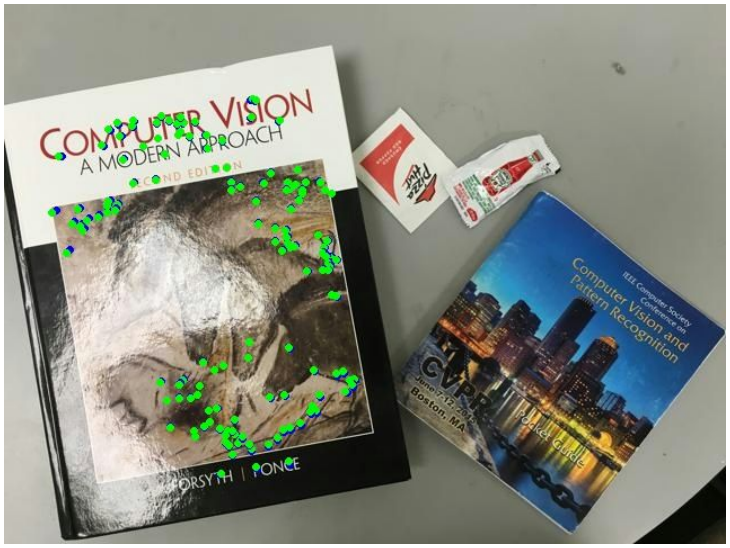
```
    for match in matches:
            x = [match[0].pt[0], match[0].pt[1], 1]
            y = [match[1].pt[0], match[1].pt[1], 1]
            result = np.dot(H, x)
            fit.append(np.linalg.norm(result-y))

    inliers = [f for f in fit if f < fittreshold]
    fraction = float(len(inliers))/len(fit)
    if(fraction>inliertreshold):
            break

print 'Homography H: '
print H

# Draw lines joining matching points
for match in matches:
    img1_kp = match[0]
    img2_kp = match[1]

    x1 = int(img1_kp.pt[0])
    y1 = int(img1_kp.pt[1])

    x2 = int(img2_kp.pt[0] + width1)
    y2 = int(img2_kp.pt[1])

    cv2.circle(both, (x1, y1), radius=1, color=(0, 0, 255), thickness=1)
    cv2.circle(both, (x2, y2), radius=1, color=(0, 0, 255), thickness=1)
    cv2.line(both, (x1, y1), (x2, y2), color=(0, 0, 255))

# Draw circles on transformed SIFT feature points
for match in matches:
    X = np.array([ match[0].pt[0] , match[0].pt[1] , 1 ])
    X2 = np.array([ match[1].pt[0] , match[1].pt[1] , 1 ])
    Y = np.dot(H, X)
    cv2.circle(img2, (int(X2[0]), int(X2[1])), radius=2, color=(255, 0, 0), thickness=2);
    cv2.circle(img2, (int(Y[0]), int(Y[1])), radius=2, color=(0, 255, 0), thickness=2);

# Display images and write to file
cv2.namedWindow('b.Features Overlay',cv2.WINDOW_NORMAL)
cv2.imshow('b.Features Overlay', img2)
cv2.imwrite('img2_5_partb.jpg',img2.astype('uint8'))
cv2.namedWindow('a. SIFT matches',cv2.WINDOW_NORMAL)
cv2.imshow('a. SIFT matches', both)
cv2.imwrite('img2_5_parta.jpg', both.astype('uint8'))
cv2.waitKey(0)
cv2.destroyAllWindows()
```

2.

Results:

Image 1 & 3 [Part (a)]

Image 1 & 3 [Part (b)]



Homography H:

[[5.68564953e-01 8.52355154e-02 1.73153561e+01]
[-9.82829426e-02 5.63176060e-01 9.81286025e+01]
[8.67361738e-19 5.42101086e-19 1.00000000e+00]]

Image 2 & 3 [Part (a)]

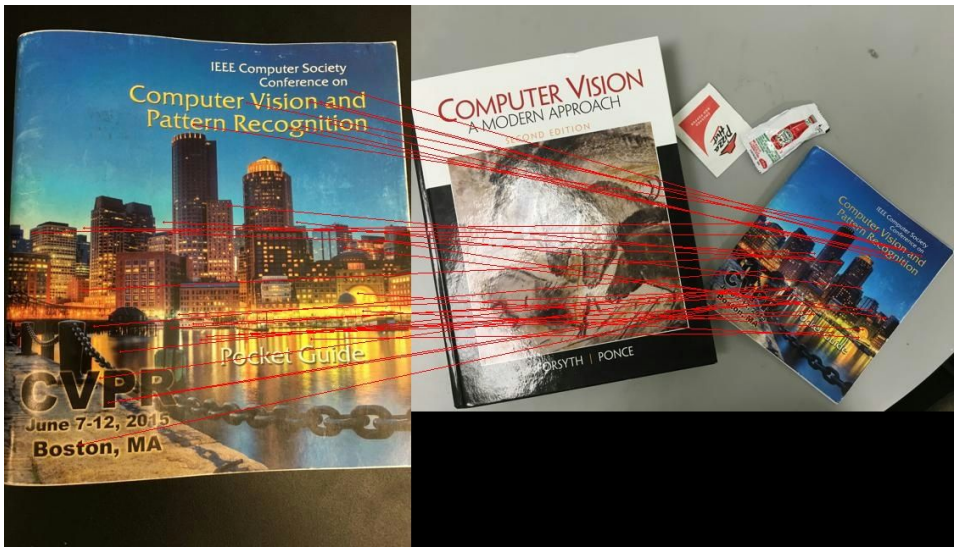
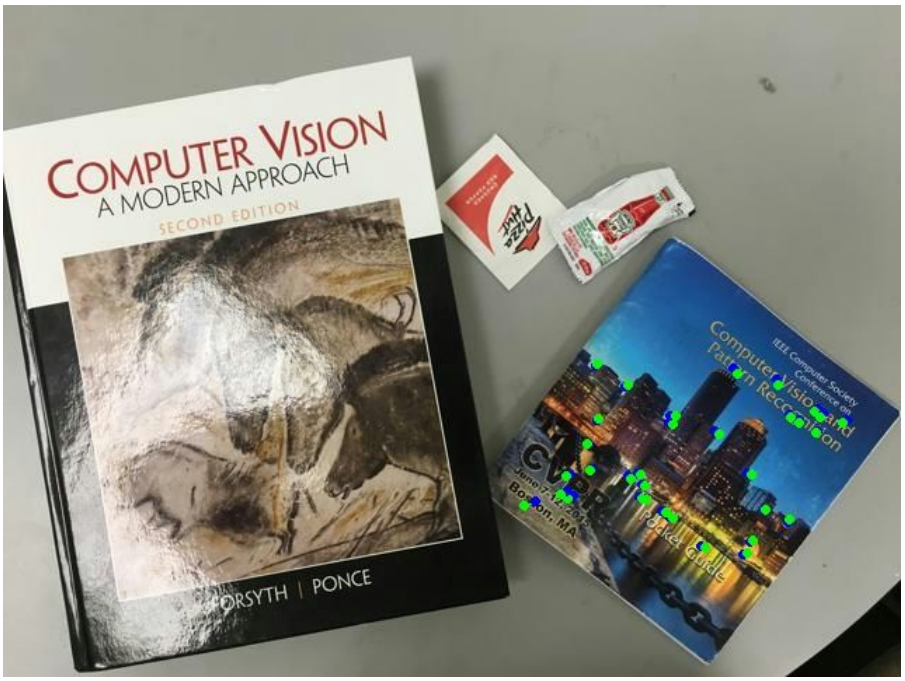


Image 2 & 3 [Part (b)]



Homography H:

[[3.42643476e-01 -2.82199729e-01 4.87193516e+02]
[2.65427506e-01 3.42551378e-01 1.55522307e+02]
[-4.33680869e-19 -4.33680869e-19 1.00000000e+00]]

Image 1 & 4 [Part (a)]

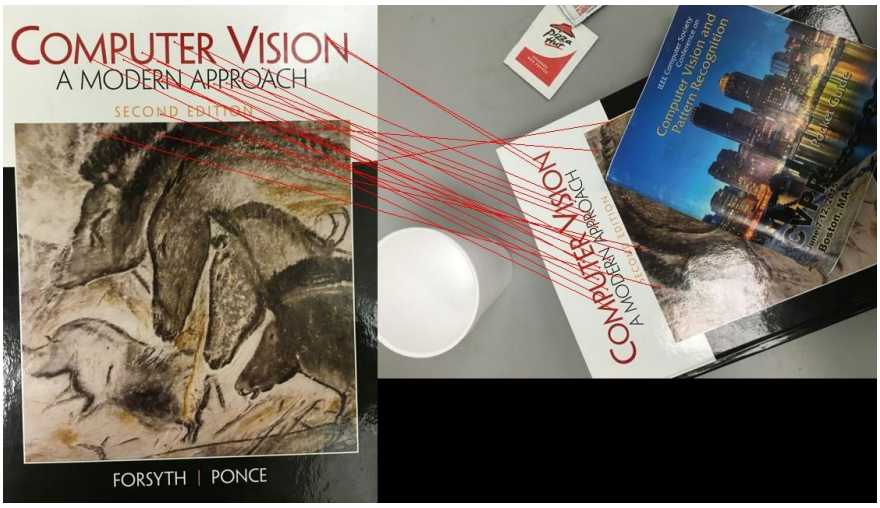
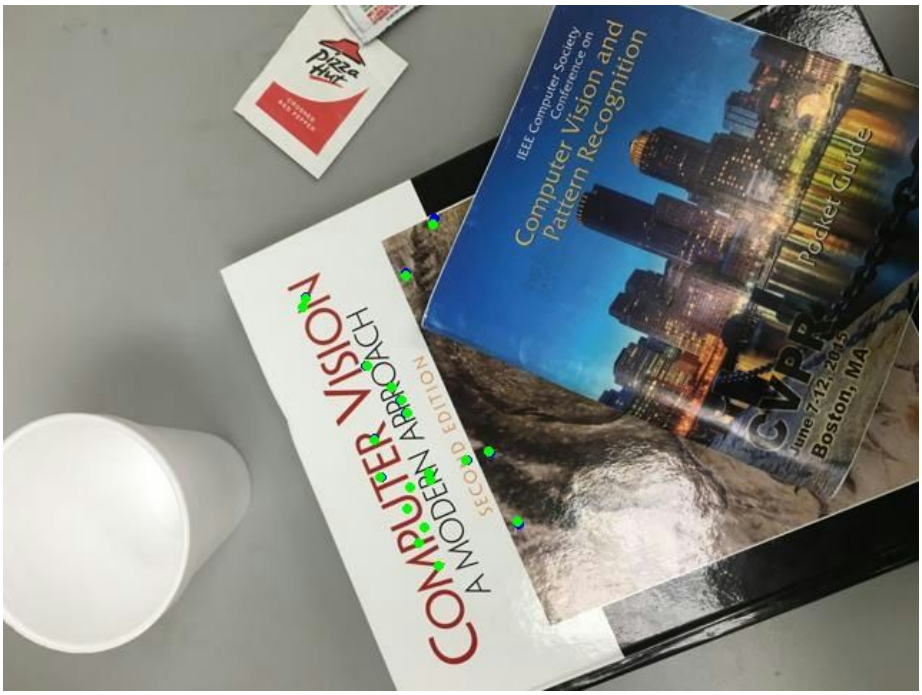


Image 1 & 4 [Part (b)]



Homography H:

[[-2.61847883e-01 5.93462748e-01 2.93201388e+02]
[-6.17254672e-01 -2.50238994e-01 4.78137979e+02]
[8.67361738e-19 -1.73472348e-18 1.00000000e+00]]

Image 2 & 4 [Part (a)]

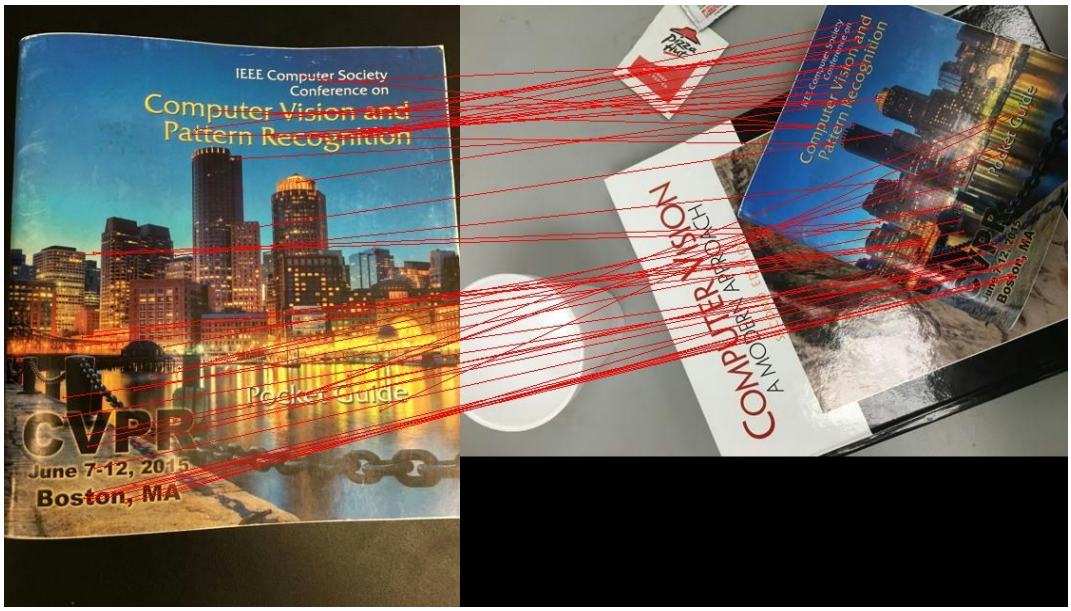
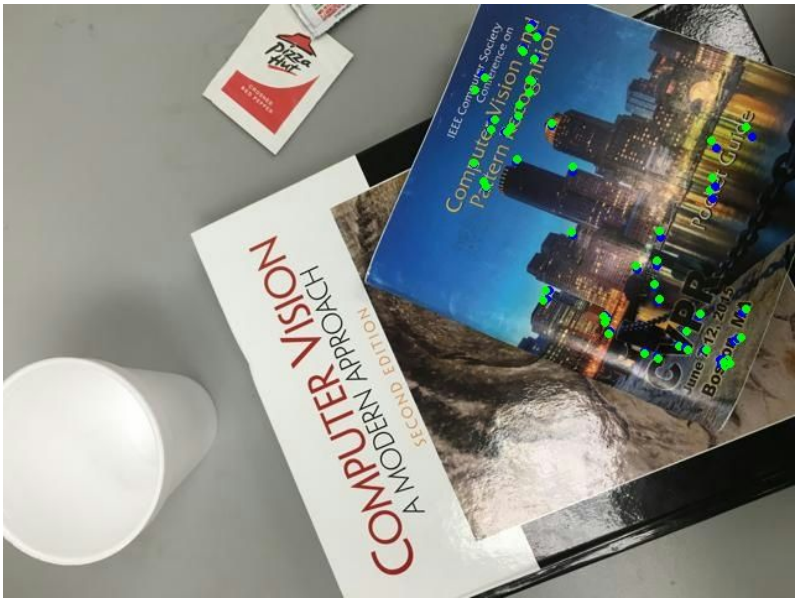


Image 2 & 4 [Part (b)]



Homography H:

[[2.21642741e-01 5.65167176e-01 2.68481899e+02]
[-5.56005969e-01 2.13691411e-01 2.27857615e+02]
[8.67361738e-19 8.67361738e-19 1.00000000e+00]]

Image 1 & 5 [Part (a)]

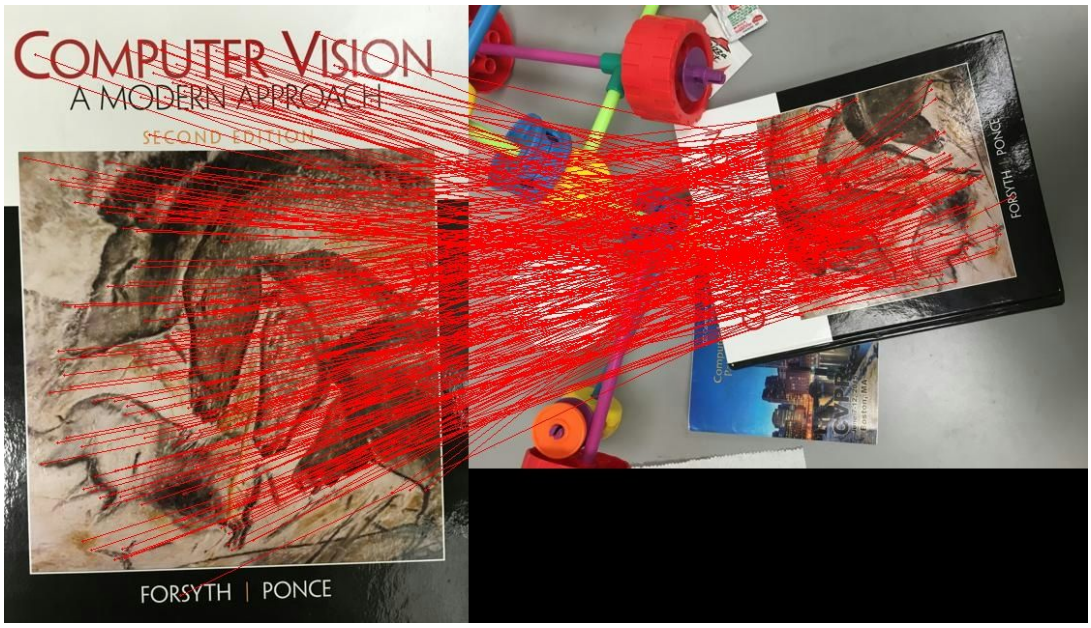


Image 1 & 5 [Part (b)]

Homography H:

[[-1.07877445e-01 5.06659367e-01 2.69011503e+02]
 [-5.00792057e-01 -1.05988928e-01 3.54666743e+02]
 [-4.33680869e-19 -8.67361738e-19 1.00000000e+00]]

Image 2 & 5 [Part (a)]

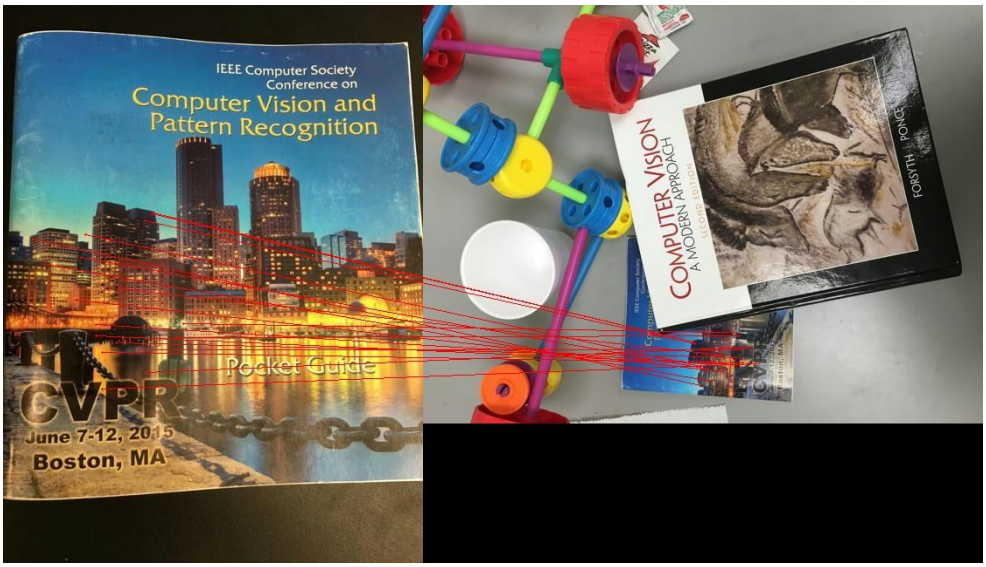
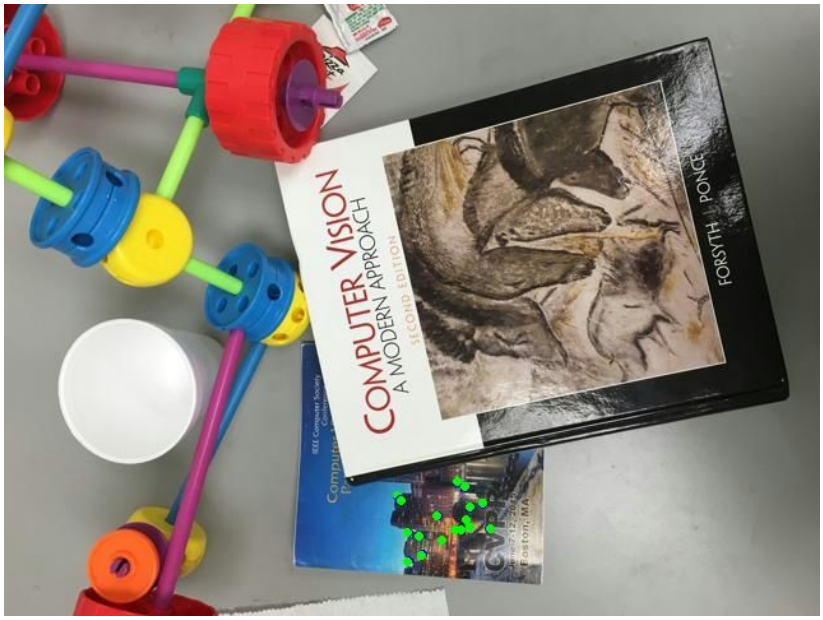


Image 2 & 5 [Part (b)]



Homography H:

[[1.85103317e-02 3.67416250e-01 2.17339365e+02]
[-3.75145406e-01 2.00376051e-02 4.44997055e+02]
[-3.90312782e-18 -2.60208521e-18 1.00000000e+00]]

3.

Analysis:

Considering SIFT matches b/w images (1 ,3) and (1 ,5). The number of SIFT features that show good match is very high. This is due to the presence of sharper features, more corner features from the art form, larger capitalized fonts. On the other hand Image 2 does not have very sharp features. Also the font size is small and curvy. Thus there are very less sharp or corner features to extract. So the number of feature matches when comparing images (2, 3) and (2, 4) is much lesser.

Specular highlights from illumination in image 3 is causing lesser features to get detected and matched.

Occlusion caused in images 5 and 4 is causing lesser features to get detected.

Presence of other objects in scene also causes wrong matches to get detected. For example while comparing images 1 and 4 I had to increase the threshold as some of the features were getting matched to the PizzaHut logo.