CSC420 A3

Yu-Chieh Wu

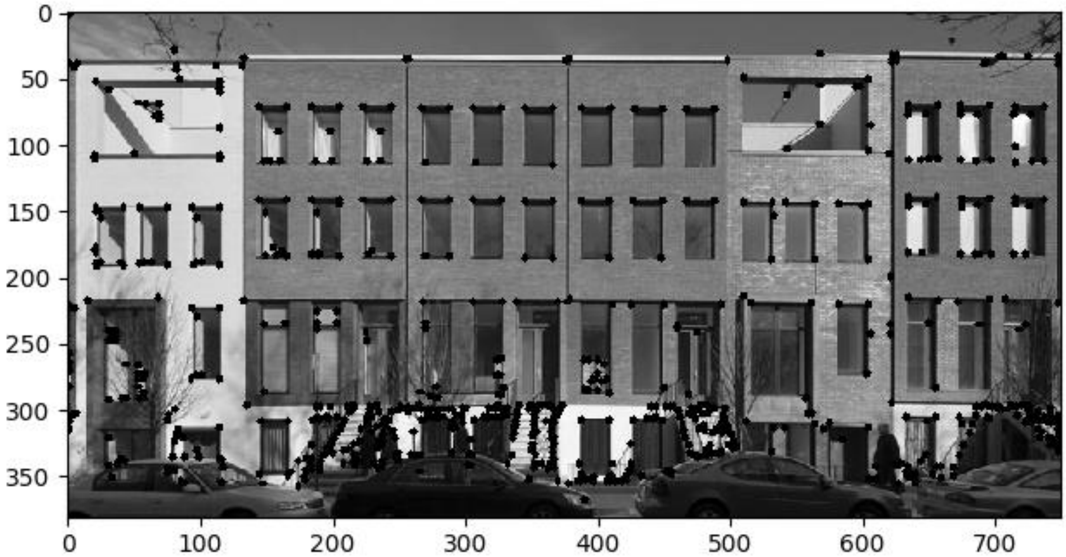
Q1 a)

```
def q1_a_harris_corner_detection(image):
    # Calculate Ix, Iy
    Mx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    My = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])
    Ix = signal.convolve2d(image, Mx, mode='same')
    Iy = signal.convolve2d(image, My, mode='same')

    #calculate Ixx, Iyy, Ixy and perform gaussian with window size 3 to get M
    Ixx, Iyy, Ixy = np.square(Ix), np.square(Iy), Ix*Iy
    gaussian_filter = get_gaussian_filter(3, 2)
    Ixx = signal.convolve2d(Ixx, gaussian_filter, mode = 'same') # M[0][0]
    Iyy = signal.convolve2d(Iyy, gaussian_filter, mode = 'same') # M[1][1]
    Ixy = signal.convolve2d(Ixy, gaussian_filter, mode = 'same') # M[0][1] and M[1][0]

    # calculate R and find points with R>threshhold
    R = (Ixx*Iyy - Ixy*Ixy) - 0.04*np.square(Ixx+Iyy)
    R = np.where(R>0.25, R, 0)

    # perform non max suppress
    num_row, num_col = R.shape
    non_max_suppress = np.zeros((num_row,num_col))
    for i in range(num_row):
        for j in range(num_col):
            # find the 3x3 submatrix to see if curr location is a local max
            right = i-1 if i>0 else 0
            left = i+2 if i<num_row-1 else i+1
            upper = j-1 if j>0 else 0
            bottom = j+2 if j<num_col-1 else j+1
            curr_max = np.max(R[right:left, upper:bottom])
            non_max_suppress[i,j] = 1 if R[i,j] == curr_max and R[i,j]>0.25 else 0
    return non_max_suppress
```

```
# reference: helper function for my CSC420 a1
def get_gaussian_filter(kernel_size,sigma):
    x_values = np.linspace(-1* (kernel_size//2), kernel_size//2, kernel_size)
    gaussian = np.zeros(x_values.shape[0])
    for i in range(x_values.shape[0]):
        gaussian[i] = 1 / (sigma * math.sqrt(2*math.pi)) * np.exp(-1*pow(x_values[i],2)/(2*pow(sigma,2)))
    #2d gaussian is the outer product of the 1D gaussian
    kernel = np.outer(gaussian.T, gaussian)
    # normalize
    kernel = kernel / np.sum(kernel)
    return kernel
```

Q1 b)

```
def q1_b_drawing(image):
    non_max_suppress = q1_a_harris_corner_detection(image)
    for i in range(non_max_suppress.shape[0]):
        for j in range(non_max_suppress.shape[1]):
            if non_max_suppress[i,j]:
                image = cv2.circle(image, (j ,i), radius=3, color=(0, 0, 255), thickness=-1)
    io.imshow(image)
    io.show()
```

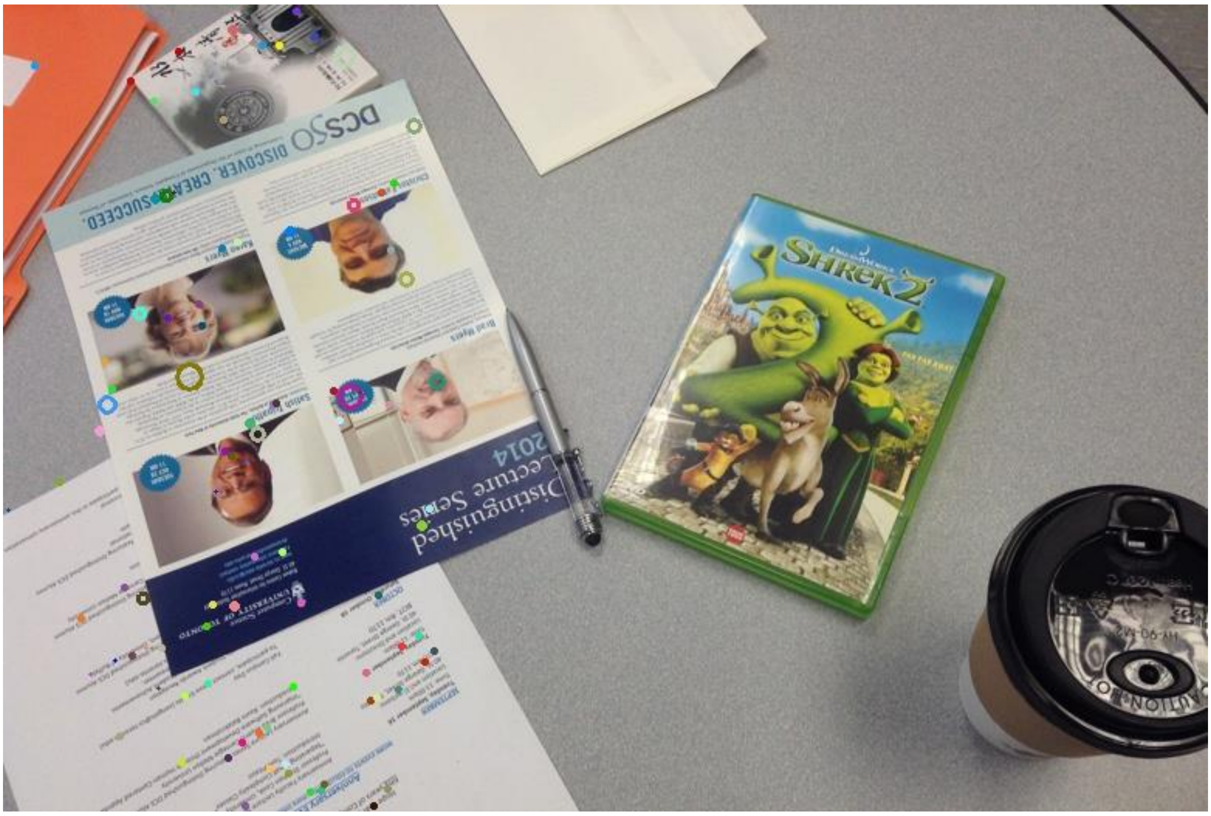


Q2 a)

```
# Reference: OpenCV SIFT documenation, link: https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html
def q2_a_feature_extraction(image, image_name):
    sift = cv2.SIFT_create()
    keypoints, descriptors = sift.detectAndCompute(image,None)
    for i in range(0, 1000, 10):
        x, y = int(keypoints[i].pt[0]), int(keypoints[i].pt[1])
        r = int(keypoints[i].size/2)
        image = cv2.circle(image, (x,y), r, (random.randint(0, 255), random.randint(0, 255), random.randint(0, 255)), 2)
    cv2.imwrite('100_keypoints_{}.jpg'.format(image_name),image)
```

100 keypoints plotted for **reference.png**:

100 keypoints plotted for **test.png**:



100 keypoints plotted for **test2.png**:

Q2 b)

Matching: The simple algorithm I'm using to find the top3 matches is the one described in the lecture. I compared the ratio = ||fi – f'i_1|| / ||fi – f'i_1||, where

fi = the descriptor of keypoint i in reference.png

f'i_1 = the closest match to fi among the descriptors of the test image

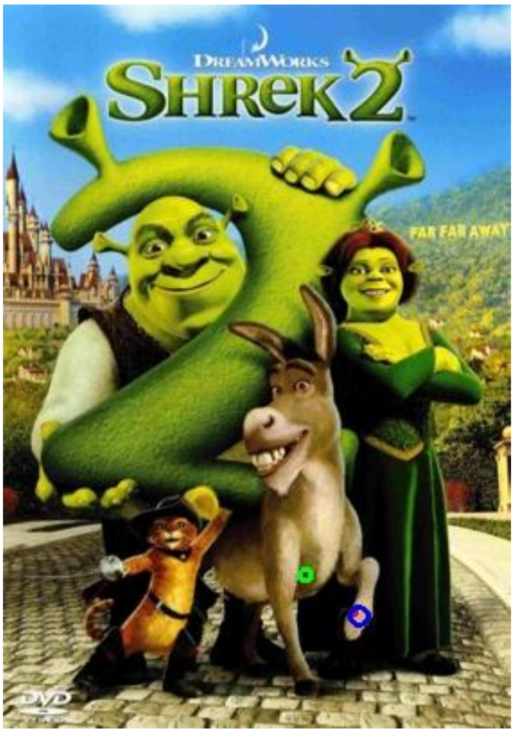
f'i_2 = the second closest match to fi among the descriptors of the test image

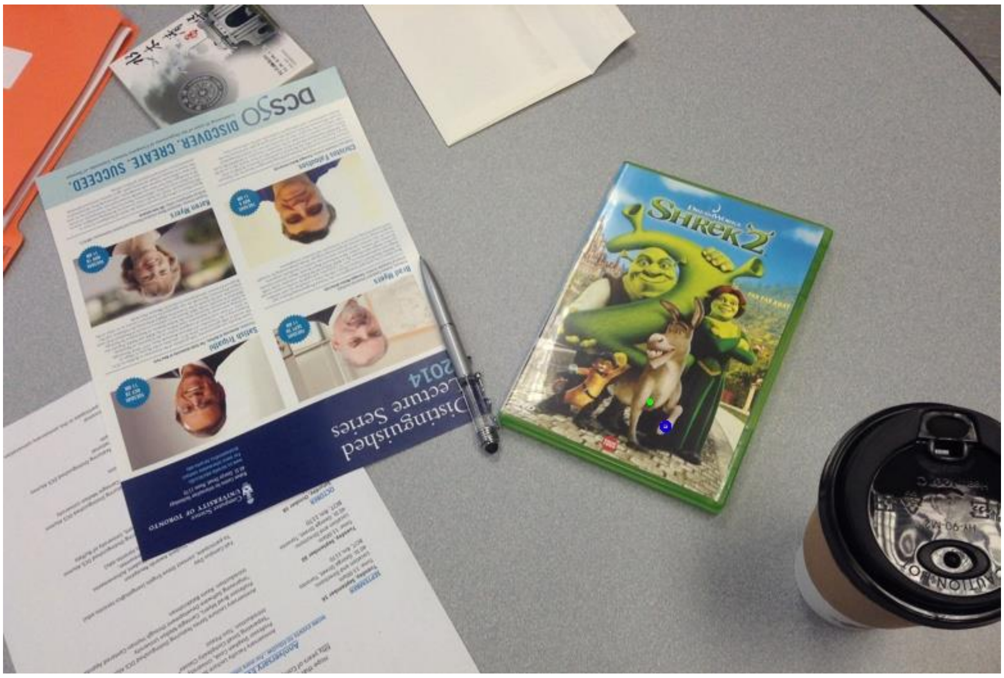
I pick the top3 minimum ratio to be the top 3 matches and make sure they are under the threshold, which I set to 0.8(as suggested in lecture slide).

```python
def q2_b_matching(ref_img, test_img):
    sift = cv2.SIFT_create()
    ref_keypoints, ref_descriptors = sift.detectAndCompute(ref_img,None)
    test_keypoints, test_descriptors = sift.detectAndCompute(test_img,None)
    top3_ratio = []
    top3_keypoints = []
    for i in range(ref_descriptors.shape[0]):
        # calculate Euclidean distance and fnd the ratio between closest and sec closest
        distance = np.linalg.norm(ref_descriptors[i] - test_descriptors, axis=1)
        closest_idx = np.argmin(distance)
        closest = distance[closest_idx]
        distance[closest_idx] = float('inf')
        sec_close_idx = np.argmin(distance)
        sec_close = distance[sec_close_idx]
        ratio = closest/sec_close
        if ratio > 0.8:
            continue
        if len(top3_ratio) < 3:
            top3_ratio.append(ratio)
            top3_keypoints.append([ref_keypoints[i], test_keypoints[closest_idx]])
        else:
            max_ratio = max(top3_ratio)
            if ratio <= max_ratio:
                index = top3_ratio.index(max_ratio)
                top3_ratio.pop(index)
                top3_keypoints.pop(index)
                top3_ratio.append(ratio)
                top3_keypoints.append([ref_keypoints[i], test_keypoints[closest_idx]])
    color = [(0, 255, 0),(0, 0, 255), (255, 0, 0)]
    for i in range(len(top3_keypoints)):
        r_keypoint = top3_keypoints[i][0]
        t_keypoint = top3_keypoints[i][1]
        r_x, r_y = int(r_keypoint.pt[0]), int(r_keypoint.pt[1])
        r_radius = int(r_keypoint.size/2)
        t_x, t_y = int(t_keypoint.pt[0]), int(t_keypoint.pt[1])
        t_radius = int(t_keypoint.size/2)
        ref_img = cv2.circle(ref_img, (r_x,r_y), r_radius, color[i], 2)
        test_img = cv2.circle(test_img, (t_x,t_y), t_radius, color[i], 2)
    cv2.imwrite('ref_top3_keypoints.jpg',ref_img)
    cv2.imwrite('test_top3_keypoints.jpg',test_img)
    return top3_keypoints
```

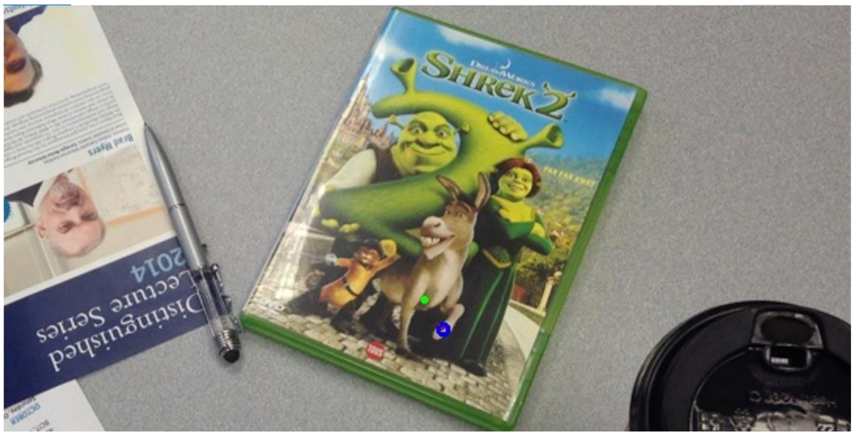
Top3 matches between **reference.png** and **test.png**:

 # Note: the red point is a bit blocked by the blue one, but there are 3 points in total

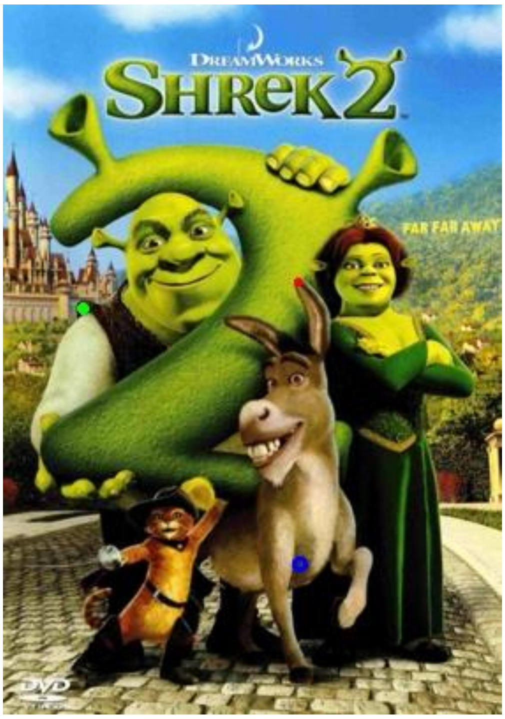




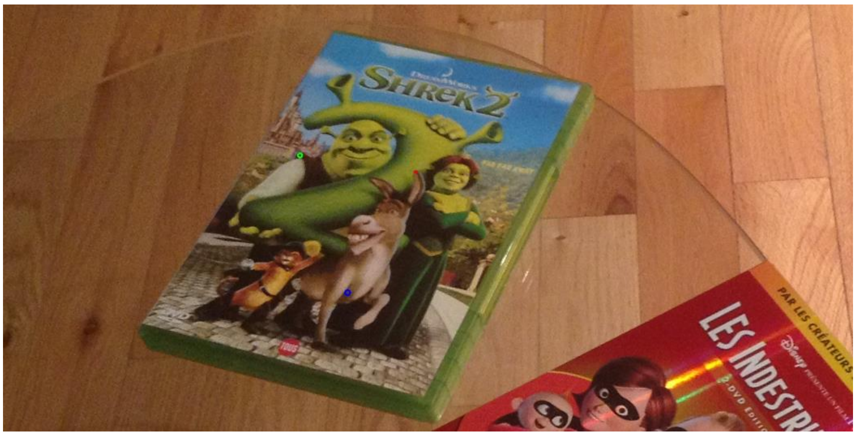
Here is a zoom in version of test.png:



Top3 matches between **reference.png** and **test2.png**:



Here is a zoom in version of test2.png:



Q2 c)

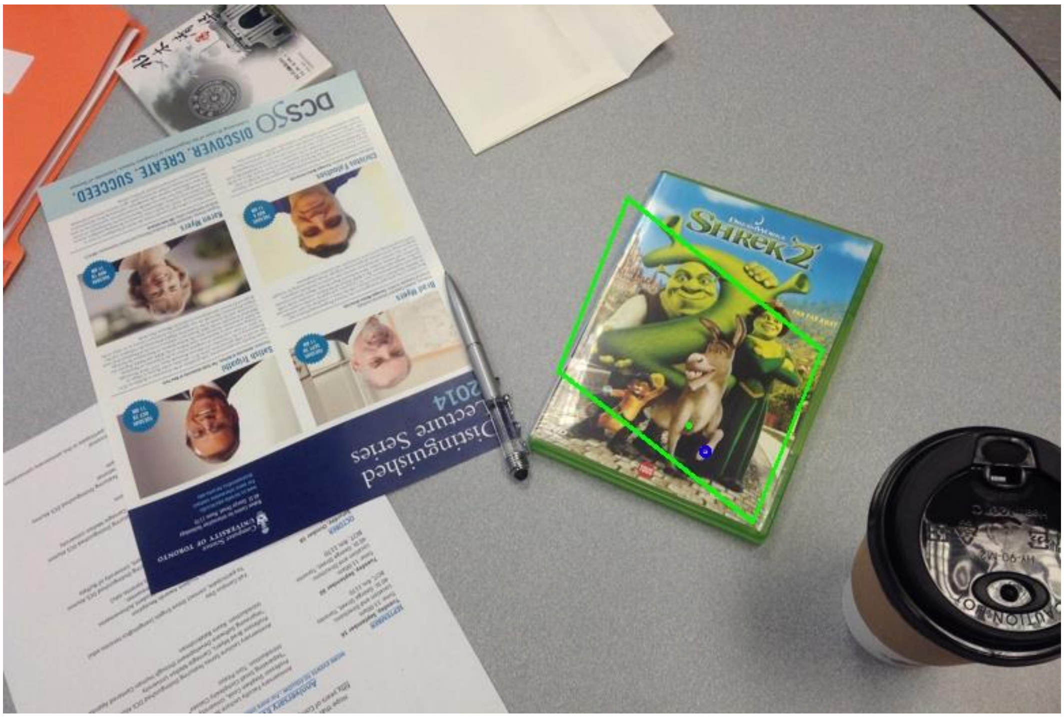
```python
def q2_c_affine_transformation(ref_img, test_img):
    top3_keypoints = q2_b_matching(ref_img, test_img)
    P, prime = [], []
    for kp in top3_keypoints:
        x_i, y_i = kp[0].pt[0], kp[0].pt[1]
        x_i_prime, y_i_prime = kp[1].pt[0], kp[1].pt[1]
        P += [[x_i, y_i, 0, 0, 1, 0],[0, 0, x_i, y_i, 0, 1]]
        prime += [x_i_prime, y_i_prime]
    P, prime = np.array(P), np.array(prime)
    AT_matrix_1D = np.dot(np.dot(np.linalg.inv(np.dot(P.T, P)), P.T), prime)
    AT_matrix_2by3 = AT_matrix_1D[:4].reshape((2,2)).tolist()
    AT_matrix_2by3[0].append(AT_matrix_1D[4])
    AT_matrix_2by3[1].append(AT_matrix_1D[5])
    AT_matrix_2by3 = np.array(AT_matrix_2by3)
    return AT_matrix_2by3
```

Q2 d)

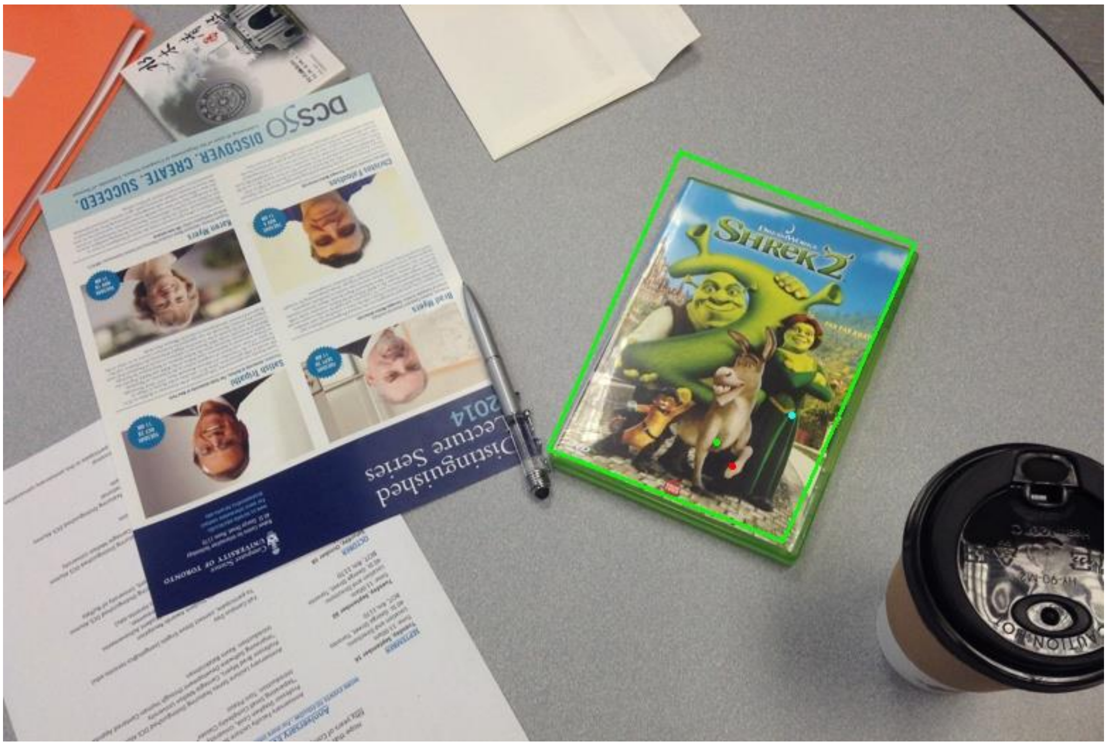
```python
def q2_d_visualize_q2c(ref_img, test_img):
    AT_matrix = q2_c_affine_transformation(ref_img, test_img)
    # get the 4 corners of ref image
    ul = np.dot(AT_matrix, np.array([0, 0, 1]))
    ur = np.dot(AT_matrix, np.array([ref_img.shape[1]-1, 0, 1]))
    ll = np.dot(AT_matrix, np.array([0, ref_img.shape[0]-1, 1]))
    lr = np.dot(AT_matrix, np.array([ref_img.shape[1]-1, ref_img.shape[0]-1, 1]))
    # plot the affine transformed corners on test img
    test_img = cv2.line(test_img, (int(ur[0]), int(ur[1])), (int(ul[0]), int(ul[1])), (0, 255, 0), thickness=2)
    test_img = cv2.line(test_img, (int(ur[0]), int(ur[1])), (int(lr[0]), int(lr[1])), (0, 255, 0), thickness=2)
    test_img = cv2.line(test_img, (int(ll[0]), int(ll[1])), (int(lr[0]), int(lr[1])), (0, 255, 0), thickness=2)
    test_img = cv2.line(test_img, (int(ll[0]), int(ll[1])), (int(ul[0]), int(ul[1])), (0, 255, 0), thickness=2)
    cv2.imwrite('test_AT.jpg',test_img)
```

Visualize affine transform with reference.png and test.png:

Note: By using the top3 keypoints I previously obtained in q2b does not give an accurate transformation (picture below). I believe this is because there are two keypoints that are very close to each other (red and blue are really close), which does not give me an accurate transformation matrix (They are too close and basically contribute the same during calculation).



Therefore, I discarded the blue keypoint and picked the keypoint that has the 4th smallest ratio meaning the top 4th match (colored in light blue). I performed affine transformation again and got a more accurate transformation shown in the picture below:



Visualize affine transform with reference.png and test2.png: