16-820 Advanced Computer Vision Assignment 1

Li-Wei Yang
liweiy@andrew.cmu.edu
Section A
September 19, 2023

Q1.1

Let x_π be written in homogeneous coordinates. According to the definition of the camera projection matrix, $P_1 x_\pi = x_1$, $P_2 x_\pi = x_2$, where P_1 and P_2 are $3 * 4$ matrices. Because point x_π is on a plane, we define a global frame on the plane, so the z coordinate of $x_\pi = 0$. By doing so, the third column of P_1 and P_2 could be reduced, thus making P_1 and P_2 both $3 * 3$ matrices. We could assume P_1 and P_2 are both invertible as long as the plane x_π is on is not intersecting with the camera center. So the equations become $x_\pi = P_1^{-1} x_1$, $x_\pi = P_2^{-1} x_2$, from the equations we derive $P_1^{-1} x_1 = P_2^{-1} x_2$ and thus there exist $H = P_1 P_2^{-1}$.

Q1.2

---

1. h has 9 parameters, but h is in the homogeneous system so in total has 8 degrees of freedom.
2. To solve h, we need to satisfy all its degrees of freedom. Each point pair could generate two equations, so we need 4 point pairs in total to solve h.
3. For a single point, we could write down $\mathbf{x}_1^i = \mathbf{H}\mathbf{x}_2^i$ as follows:

   $$\begin{bmatrix} x_1^i \\ y_1^i \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{x}_1^i \\ \tilde{y}_1^i \\ \tilde{z}_1^i \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_2^i \\ y_2^i \\ 1 \end{bmatrix}$$

   From the definition of homogeneous coordinates,

   $x_1^i = \tilde{x}_1^i / \tilde{z}_1^i = \frac{h_{11}x_2^i + h_{12}y_2^i + h_{13}}{h_{31}x_2^i + h_{32}y_2^i + h_{33}}$, if we rearrange the terms:
   $x_1^i \left(h_{31}x_2^i + h_{32}y_2^i + h_{33}\right) = h_{11}x_2^i + h_{12}y_2^i + h_{13}$.

   Same for the y coordinates: $y_1^i = \tilde{y}_1^i / \tilde{z}_1^i = \frac{h_{21}x_2^i + h_{22}y_2^i + h_{23}}{h_{31}x_2^i + h_{32}y_2^i + h_{33}}$,
   $y_1^i \left(h_{31}x_2^i + h_{32}y_2^i + h_{33}\right) = h_{21}x_2^i + h_{22}y_2^i + h_{23}$.

   We can write down the equation in matrix form: $\mathbf{A}_i\mathbf{h} = \mathbf{0}$:

$$\begin{bmatrix} x_2^i & y_2^i & 1 & 0 & 0 & 0 & -x_1^i x_2^i & -x_1^i y_2^i & -x_1^i \\ 0 & 0 & 0 & x_2^i & y_2^i & 1 & -y_1^i x_2^i & -y_1^i y_2^i & -y_1^i \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Thus, we can define $\mathbf{A}_i = \begin{bmatrix} x_2^i & y_2^i & 1 & 0 & 0 & 0 & -x_1^i x_2^i & -x_1^i y_2^i & -x_1^i \\ 0 & 0 & 0 & x_2^i & y_2^i & 1 & -y_1^i x_2^i & -y_1^i y_2^i & -y_1^i \end{bmatrix}$.

4. The trivial solution of h would be $\mathbf{0}$ because zero vector exists in the null space of all matrices. Say Matrix $\mathbf{A}$ be of size 8*9, matrix $\mathbf{A}$ could be full rank if we pick unique points. If $\mathbf{A}$ is not full rank, there would be singular values = 0.

Q1.4

---

1. $\mathbf{x}_1 = \mathbf{K}_1 \begin{bmatrix} \mathbf{I} & 0 \end{bmatrix} \mathbf{X}$,we assume $\mathbf{K}_1 = \begin{bmatrix} s_x & s_\theta & o_x \\ 0 & s_y & o_y \\ 0 & 0 & 1 \end{bmatrix}$, $\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix}$, $\mathbf{K}_1$ has eigenvalues s_x, s_y and 1, it's full rank so invertible, same as $\mathbf{K}_2$. The equations thus become $\mathbf{K}_1^{-1}\mathbf{x}_1 = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, $\mathbf{K}_2^{-1}\mathbf{x}_2 = \mathbf{R} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, $\mathbf{R}$ is homogeneous transform thus invertible, so there exist homography $\mathbf{H} = \mathbf{K}_1\mathbf{R}\mathbf{K}_2^{-1}$ that satisfies $\mathbf{x}_1 = \mathbf{H}\mathbf{x}_2$.

2. From last question, $\mathbf{H} = \mathbf{K}_1\mathbf{R}\mathbf{K}_2^{-1}$. In this question $\mathbf{K}_1 = \mathbf{K}_2$ so $\mathbf{H} = \mathbf{K}\mathbf{R}\mathbf{K}^{-1}$. Then $\mathbf{H}^2 = \mathbf{K}\mathbf{R}\mathbf{K}^{-1}\mathbf{K}\mathbf{R}\mathbf{K}^{-1} = \mathbf{K}\mathbf{R}^2\mathbf{K}^{-1}$. From the definition of rotation matrix we can know $\mathbf{R}^2$ is rotation corresponding to 2θ, so $\mathbf{H}^2$ is the homography of rotating 2θ.

3. Because in arbitrary scenes, we cannot define two points on same plane, so the Z coordinates cannot be reduce, thus planar homography cannot work.

4. Define a line in 3D: $\mathbf{X} = \mathbf{X}_1 + \lambda(\mathbf{X}_2 - \mathbf{X}_1)$, where $\mathbf{X}_1$ and $\mathbf{X}_2$ are two different points lying on the line. Select a different point $\mathbf{X}_3$ on the line, then $\mathbf{X}_3 = \mathbf{X}_1 + \lambda(\mathbf{X}_2 - \mathbf{X}_1)$. Project the line in 2D: $\mathbf{P}\mathbf{X}_3 = \mathbf{P}\mathbf{X}_1 + \lambda(\mathbf{P}\mathbf{X}_2 - \mathbf{P}\mathbf{X}_1) \implies \mathbf{x}_3 = \mathbf{x}_1 + \lambda(\mathbf{x}_2 - \mathbf{x}_1)$. From the equation, we can see that the three points are still on the same line with λ as scalar and $(\mathbf{x}_2 - \mathbf{x}_1)$ as direction. Thus the projection preserves line.

Q2.1

---

1. Harris corner detector needs to calculate the gradient of the image, which involves convolution, whereas the FAST detector only compares the intensity of surrounding pixels with the center pixel. Overall, the FAST detector has better computational performance compared to the Harris corner detector. In the figure, we can see that FAST has a larger pixel rate (158 MPix/s for FAST n = 12) than Harris (8.05 MPix/s).

Detector	Training set		Test set	
	Pixel rate (MPix/s)	%	MPix/s	%
FAST $n = 9$	188	4.90	179	5.15
FAST $n = 12$	158	5.88	154	5.98
Original FAST ($n = 12$)	79.0	11.7	82.2	11.2
FAST-ER	75.4	12.2	67.5	13.7
SUSAN	12.3	74.7	13.6	67.9
Harris	8.05	115	7.90	117
Shi-Tomasi	6.50	142	6.50	142
DoG	4.72	195	5.10	179

Source: https://arxiv.org/pdf/0810.2434.pdf, Table IV.

2. BRIEF descriptor describes features by comparing the pixel values in the patch, the filter bank is a set of filters that pick up a certain frequency. Filter bank could also be used as a descriptor, one example is Gabor filters, often used to capture texture (texture is a pattern with certain frequency) or edges.

3. Since BRIEF descriptor is a kind of binary feature descriptor, we could describe a feature using a binary string, i.e., "01011101". Hamming distance could easily calculate the minimum number of operations of two strings, so it can be used to match binary strings thus BRIEF descriptor. Often, the nearest neighbor uses Euclidean distance to calculate the similarity of two data points, so if we turn the binary string into a decimal number, we can calculate the Euclidean difference of two features. But in this case the higher bits in binary string would be dominant in deciding the distance between two points instead of equal contribution from each bits in Hamming distance, so Hamming distance is more suitable for our case.

4. Modified plotMatches to also plot unmatched keypoints in blue:

   displayMatch.py

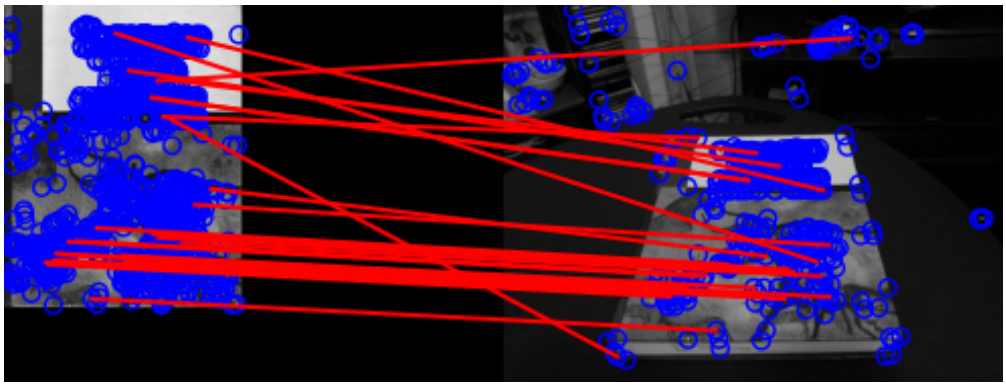
```
import numpy as np
import cv2
from matchPics import matchPics
from helper import plotMatches
from opts import get_opts
def displayMatched(opts, image1, image2):
    """
    Displays matches between two images
```

```
    Input
    -----
    opts: Command line args
    image1, image2: Source images
    """

    matches, locs1, locs2 = matchPics(image1, image2, opts)

    #display matched features
    plotMatches(image1, image2, matches, locs1, locs2)

if __name__ == "__main__":

    opts = get_opts()
    image1 = cv2.imread('../data/cv_cover.jpg')
    image2 = cv2.imread('../data/cv_desk.png')

    displayMatched(opts, image1, image2)
```

matchPics.py

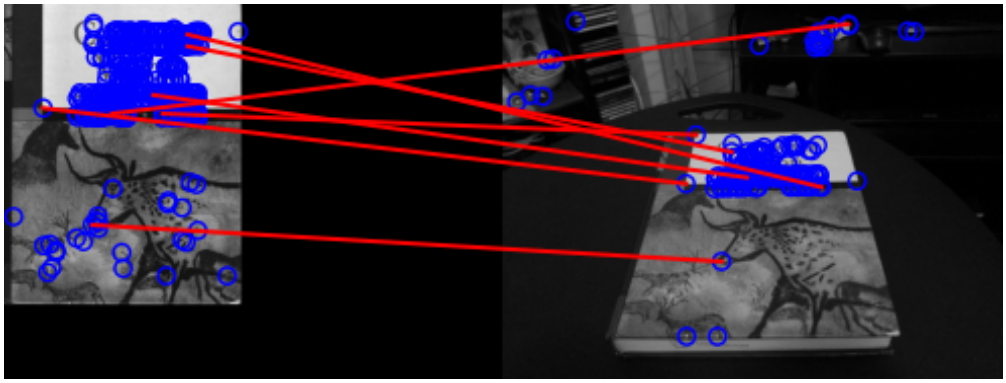
```
from skimage.color import rgb2gray
from helper import briefMatch
from helper import computeBrief
from helper import corner_detection

# Q2.1.4

def matchPics(I1, I2, opts):
        """
        Match features across images

        Input
        -----
        I1, I2: Source images
        opts: Command line args

        Returns
        -------
        matches: List of indices of matched features across I1, I2 [p x 2]
        locs1, locs2: Pixel coordinates of matches [N x 2]
        """
        ratio = opts.ratio  #'ratio for BRIEF feature descriptor'
        sigma = opts.sigma  #'threshold for corner detection using FAST feature
detector'
```

```
# TODO: Convert Images to GrayScale
I1_g = rgb2gray(I1)
I2_g = rgb2gray(I2)

# TODO: Detect Features in Both Images
locs1 = corner_detection(I1_g, sigma)
locs2 = corner_detection(I2_g, sigma)

# TODO: Obtain descriptors for the computed feature locations
desc1, desc1_locs = computeBrief(I1_g, locs1)
desc2, desc2_locs = computeBrief(I2_g, locs2)

# TODO: Match features using the descriptors
matches = briefMatch(desc1, desc2, ratio)

return matches, desc1_locs, desc2_locs
```

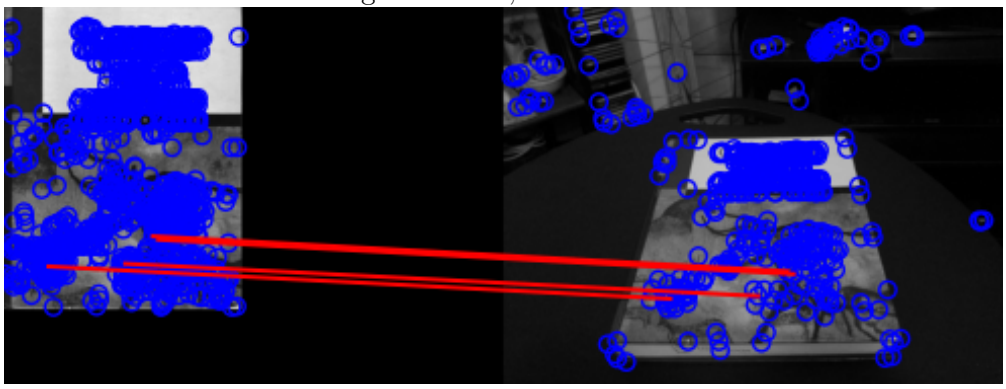


match result, sigma = 0.15, ratio = 0.7

5. σ is the FAST algorithm's threshold for detecting a patch as an interest point. With lower σ there would be more feature points. Ratio is the maximum ratio of distances between the first and second closest descriptor in the second set of descriptors. With a lower maximum ratio, we could filter out more ambiguous matches.



sigma = 0.30, ratio = 0.7

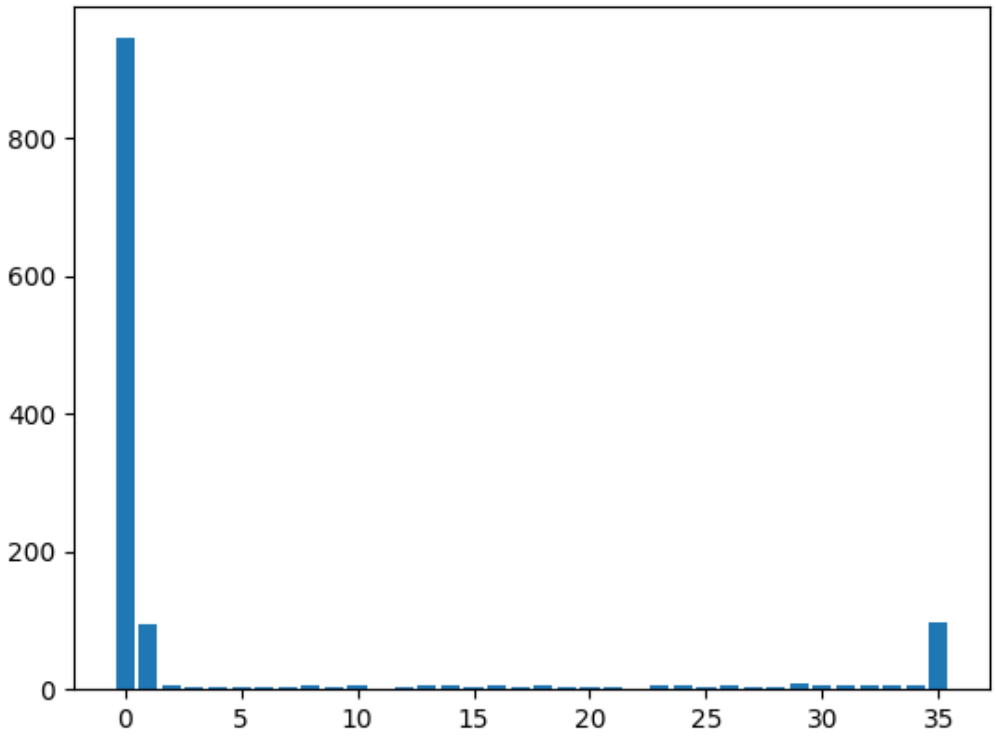


sigma = 0.15, ratio = 0.5

6. BRIEF descriptors are not invariant to rotations because they describe features by comparing the intensity of pixels in the patch. Once a large rotation occurs, the binary features no longer hold since the compared pixels are different. I did not select three very different orientations since from the bar chart we could see with large rotations the result is similar, thus I choose 0° as original, 10° as slightly rotated and 150° as large rotation examples.

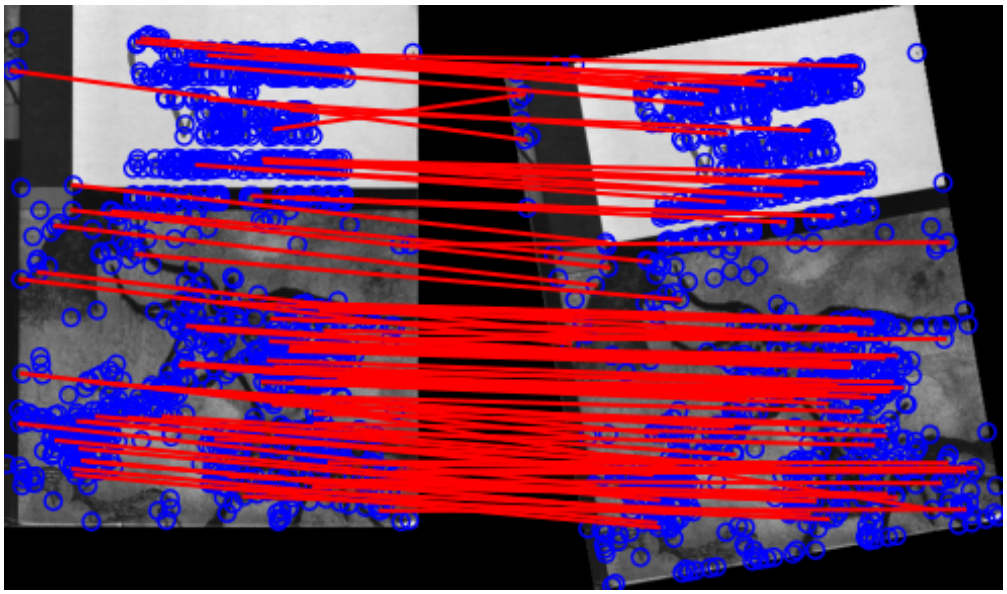
```
import numpy as np
import cv2
from matchPics import matchPics
from helper import plotMatches
from opts import get_opts
from scipy.ndimage import rotate
import matplotlib.pyplot as plt
#Q2.1.6

def rotTest(opts):

    # TODO: Read the image and convert to grayscale, if necessary
    image = cv2.imread('../data/cv_cover.jpg')
    histogram = []
    num_rot = 36
```

```
    loop = [x for x in range(num_rot)]
    for i in loop:
        print(i)

        # TODO: Rotate Image
        rotated = rotate(image, i * 10)

        # TODO: Compute features, descriptors and Match features
        matches, desc1_locs, desc2_locs = matchPics(image, rotated, opts)

        # TODO: Match features using the descriptors
        if i == 0 or i == 1 or i == 15:
            plotMatches(image, rotated, matches, desc1_locs, desc2_locs)

        # TODO: Update histogram
        histogram.append(len(matches))

    print(histogram)
    # TODO: Display histogram
    plt.bar(loop,histogram)
    plt.show()

if __name__ == "__main__":

    opts = get_opts()
    rotTest(opts)
```

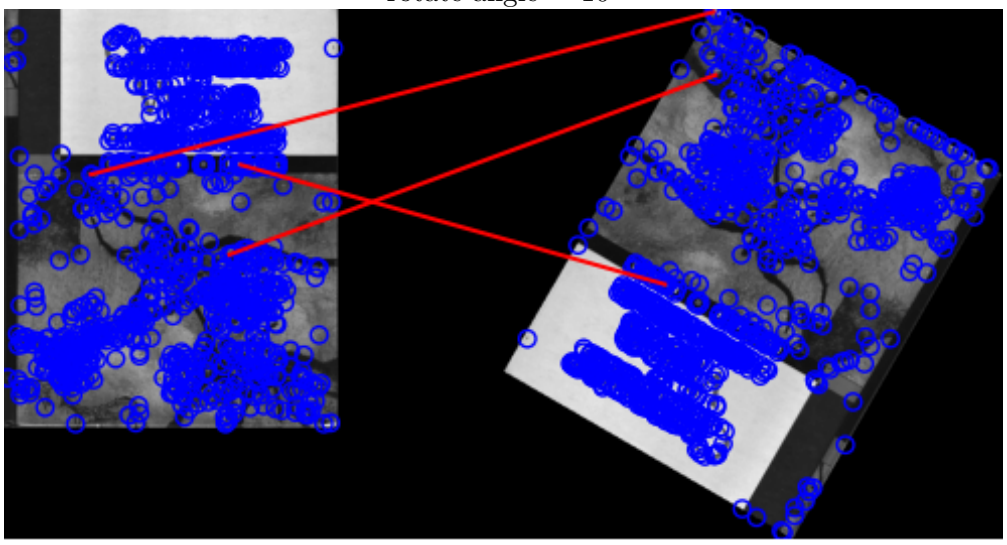


number of match pairs in different rotations

rotate angle = 0°



rotate angle = 10°



rotate angle = 150°

7. Extra

Q2.2

1.
```
def computeH(x1, x2):
    #Q2.2.1
    # TODO: Compute the homography between two sets of points
    # print(x1.shape[0])
    A_s = [0] * x1.shape[0]
```

```
    for i in range(x1.shape[0]):
        A_s[i] = np.array([[x2[i, 0], x2[i, 1], 1, 0, 0, 0,
                           -x1[i, 0]*x2[i, 0], -x1[i, 0]*x2[i, 1], -x1[i, 0]],
                          [0, 0, 0, x2[i, 0], x2[i, 1], 1,
                           -x1[i, 1]*x2[i, 0], -x1[i, 1]*x2[i, 1], -x1[i, 1]]])

    A = np.array(A_s)
    # print(A.shape)
    A = np.concatenate(A, axis=0)
    # print(A)
    # print(A.shape)

    U, S, Vh = np.linalg.svd(A, full_matrices=True)
    # print(np.shape(Vh))
    # print(Vh.T)
    # print(Vh.T)
    H = Vh[-1].reshape(3,3)
    H = H/H[2, 2]
    return H
```

2.
```
def computeH_norm(x1, x2):
    # suppose x1, x2 are of shape(4, 2)
    # Q2.2.2
    # TODO: Compute the centroid of the points
    c1 = np.sum(x1, axis=0)/x1.shape[0]
    c2 = np.sum(x2, axis=0)/x2.shape[0]

    # TODO: Shift the origin of the points to the centroid
    x1_ori = x1 - c1
    x2_ori = x2 - c2

    # TODO: Normalize the points so that the largest distance from the origin
  is equal to sqrt(2)
    x_square_sum = 0
    y_square_sum = 0
    for x in x1_ori:
        x_square_sum += x[0]**2
        y_square_sum += x[1]**2
    s_x_x1 = np.sqrt(2)/np.sqrt(x_square_sum)
    s_y_x1 = np.sqrt(2)/np.sqrt(y_square_sum)

    x_square_sum = 0
    y_square_sum = 0
    for x in x2_ori:
        x_square_sum += x[0]**2
```

```
        y_square_sum += x[1]**2
    s_x_x2 = np.sqrt(2)/np.sqrt(x_square_sum)
    s_y_x2 = np.sqrt(2)/np.sqrt(y_square_sum)

    # TODO: Similarity transform
    t1 = np.array([[s_x_x1, 0, -s_x_x1*c1[0]], [0, s_y_x1, -s_y_x1*c1[1]], [0,
0, 1]])
    t2 = np.array([[s_x_x2, 0, -s_x_x2*c2[0]], [0, s_y_x2, -s_y_x2*c2[1]], [0,
0, 1]])

    x1_ori[:, 0] = x1_ori[:, 0]*s_x_x1
    x1_ori[:, 1] = x1_ori[:, 1]*s_y_x1

    x2_ori[:, 0] = x2_ori[:, 0]*s_x_x2
    x2_ori[:, 1] = x2_ori[:, 1]*s_y_x2

    H_norm = computeH(x2_ori, x1_ori)
    # H, inliers = computeHCV2(x1_ori, x2_ori, 0)

    # TODO: Denormalization
    H2to1 = np.dot(np.linalg.inv(t2), np.dot(H_norm, t1))
    H2to1 = H2to1/H2to1[2,2]
    return H2to1
```

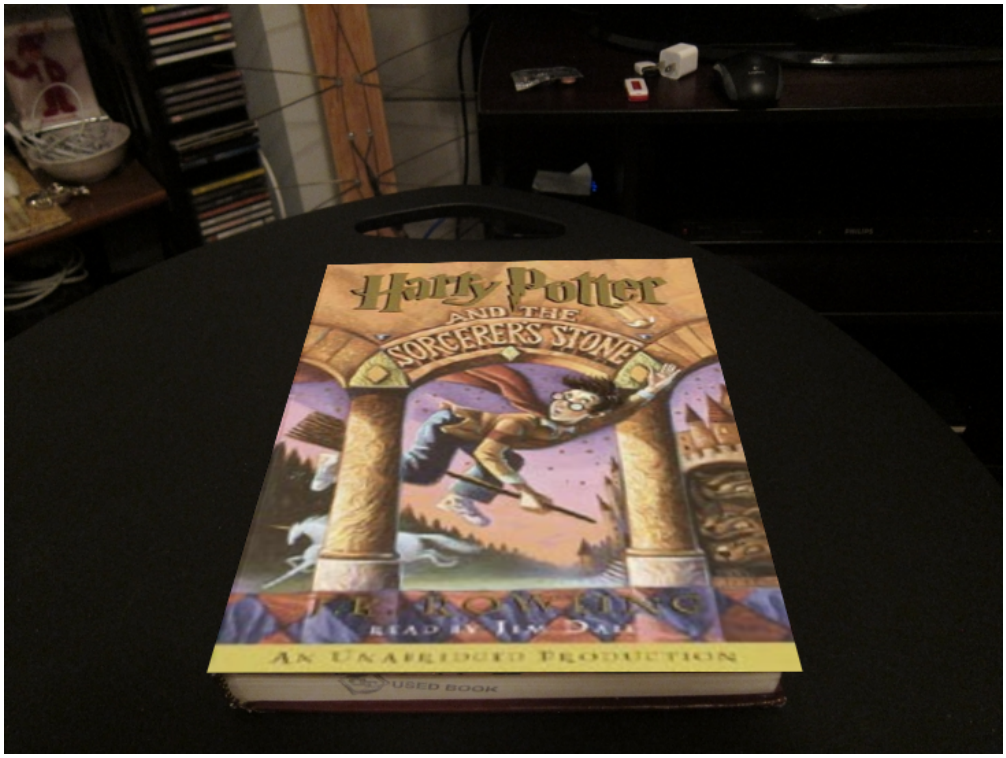
3.
```
def computeH_ransac(locs1, locs2, opts):
    # Q2.2.3
    # Compute the best fitting homography given a list of matching points

    locs1 = np.append(locs1, np.array([[1] for _ in range(locs1.shape[0])]), axis=1)
    locs2 = np.append(locs2, np.array([[1] for _ in range(locs1.shape[0])]), axis=1)

    max_iters = opts.max_iters
    # max_iters = opts.max_iters  # the number of iterations to run RANSAC for
    inlier_tol = opts.inlier_tol # the tolerance value for considering a point
to be an inlier
    # dummy values
    H_best = 0
    inliers_best = 0
    inliers_best_cnt = 0
    pick = np.floor(locs1.shape[0] * np.random.rand(max_iters,4))
    same_cnt = 0
    for i in range(max_iters):
        o = int(pick[i, 0])
        p = int(pick[i, 1])
        q = int(pick[i, 2])
```

```
    r = int(pick[i, 3])
    if o == p or o == q or o == r or p == q or p == r or q == r:
        same_cnt += 1
        continue

    l1 = np.array([locs1[o,:2], locs1[p,:2],  locs1[q,:2], locs1[r,:2]])
    l2 = np.array([locs2[o,:2], locs2[p,:2],  locs2[q,:2], locs2[r,:2]])
    # print('=====')
    H = computeH_norm(l1, l2)
    # print(H)
    # HCV, _ = computeHCV2(l1, l2, cv2.RANSAC)
    # print(HCV)
    # print('=====')

    # pdb.set_trace()

    inliers = [0] * locs1.shape[0]
    for j in range(locs1.shape[0]):
        a_s = locs1[j].T
        b_s = locs2[j].T
        a_ss = H @ a_s
        a_ss = a_ss/a_ss[2]
        sub = b_s - a_ss

        # pdb.set_trace()

        if np.sqrt(np.linalg.norm(sub)) <= inlier_tol:
            inliers[j] = 1
    if sum(inliers) > inliers_best_cnt:
        H_best = H
        inliers_best_cnt = sum(inliers)
        inliers_best = inliers
# print('H_best')
# print(H_best)
# print(inliers_best_cnt)
# print(inliers_best)
# print("same: ",same_cnt)

good1 = []
good2 = []
for idx, positive in enumerate(inliers_best):
    if positive == 1:
        good1.append(locs1[idx,:2])
        good2.append(locs2[idx,:2])
good1 = np.array(good1)
```

```
    good2 = np.array(good2)

    H_best = computeH_norm(good1, good2)
    return H_best, inliers_best
```

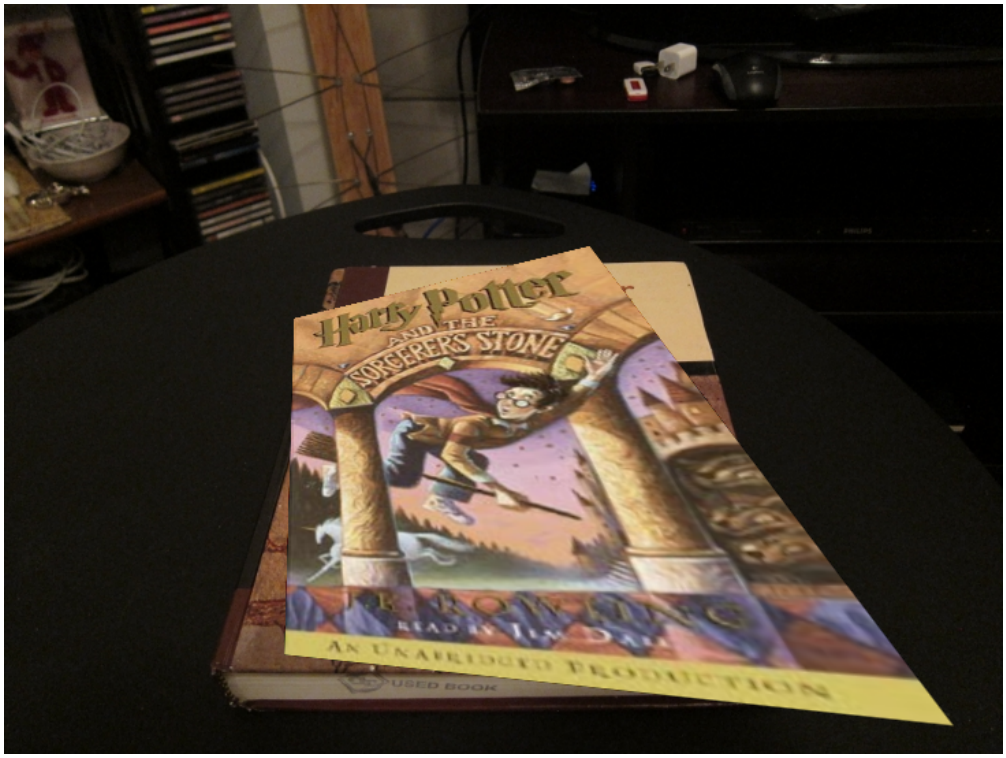
4.
```
import numpy as np
import cv2
import skimage.io
import skimage.color
from opts import get_opts
from matchPics import matchPics
from planarH import computeH_ransac, computeHCV2, compositeH

# Import necessary functions

# Q2.2.4

def warpImage():

    # step 1
    opts = get_opts()
    image1 = cv2.imread('../data/cv_cover.jpg')
    image2 = cv2.imread('../data/cv_desk.png')
    image3 = cv2.imread('../data/hp_cover.jpg')
    # fix not filling up problem
    image3 = cv2.resize(image3, (image1.shape[1], image1.shape[0]))

    # step 2
    matches, desc1_locs, desc2_locs = matchPics(image1, image2, opts)
    m1 = []
    m2 = []
    for match in matches:
        m1.append(desc1_locs[match[0]])
        m2.append(desc2_locs[match[1]])
    m1 = np.array(m1)
    m2 = np.array(m2)
    # swap x, y
    m1[:, [1, 0]] = m1[:, [0, 1]]
    m2[:, [1, 0]] = m2[:, [0, 1]]

    bestH2to1, inliers = computeH_ransac(m1, m2, opts)
    # H, inliers = computeHCV2(m1, m2, cv2.RANSAC)
    print(bestH2to1)
    # print(H)
    composite_img = compositeH(bestH2to1, image3, image2)
```

```
    return composite_img

if __name__ == "__main__":
    composite_img = warpImage()
    cv2.imshow('composite_img',composite_img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()


def compositeH(H2to1, template, img):

    # Create a composite image after warping the template image on top
    # of the image using the homography

    # Note that the homography we compute is from the image to the template;
    # x_template = H2to1*x_photo
    # For warping the template to the image, we need to invert it.

    # template: porter
    # img: desk

    # TODO: Create mask of same size as template
    mask = np.full((template.shape[0], template.shape[1]), 255, dtype=np.uint8)
    mask = cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR)

    # TODO: Warp mask by appropriate homography
    out1 = cv2.warpPerspective(mask, H2to1,(img.shape[1], img.shape[0]),flags=cv2.INTER_LI

    # TODO: Warp template by appropriate homography
    out2 = cv2.warpPerspective(template, H2to1,(img.shape[1], img.shape[0]),flags=cv2.INTE

    # TODO: Use mask to combine the warped template and the image
    img_sub = cv2.subtract(img, out1)
    img_add = cv2.add(img_sub, out2)
    return img_add
```

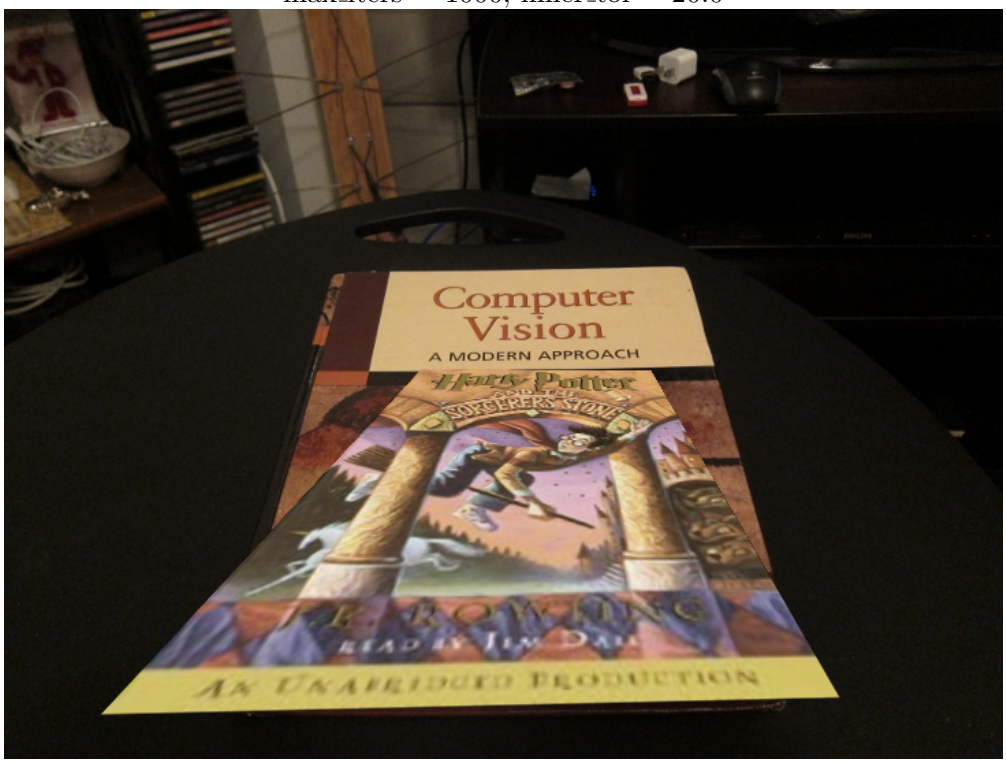


result

5. max_iters is the iteration for RANSAC, if the number of iterations is too small, the RANSAC may not sample the optimal solution. inlier_tol is the difference value for a data point to be considered inlier. If the value is too large, there would be multiple homography with high scores so RANSAC cannot distinguish the optimal one from the others.



max_iters = 1000, inlier_tol = 20.0



max_iters = 5, inlier_tol = 2.0

Q3

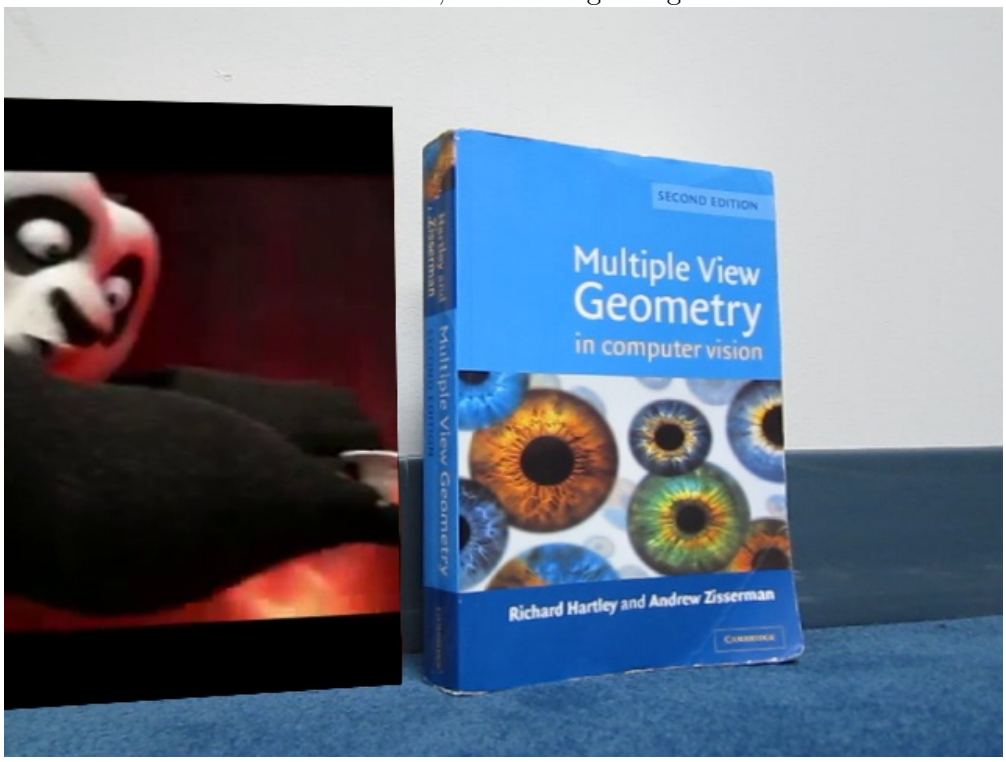
1.
```
import numpy as np
import cv2
import pdb

# Import necessary functions
from helper import loadVid
from opts import get_opts
from matchPics import matchPics
from planarH import computeHCV2, compositeH, computeH_ransac
from displayMatch import displayMatched

#Write script for Q3.1
if __name__ == "__main__":
    opts = get_opts()
    # TODO: test with 0.05
    opts.sigma = 0.15
    opts.ratio = 0.7
    opts.max_iters = 500
    opts.inlier_tol = 2.0

    src = loadVid('../data/ar_source.mov')
    dist = loadVid('../data/book.mov')
    cv_cover = cv2.imread('../data/cv_cover.jpg')
    # height, width
    src_shape = src[0].shape
    crop_shape = (src_shape[0], int(cv_cover.shape[1]*src_shape[0]/cv_cover.shape[0]))
    v_shape = dist[0].shape
    # print(src.shape): (511, 360, 640, 3)
    # print(dist.shape): (641, 480, 640, 3)
    v_len = min(src.shape[0], dist.shape[0])
    v_len_ratio = 1
    fps = 30
    output = []
    import datetime
    current_dateTime = datetime.datetime.now()
    import os
    param = current_dateTime.strftime('%S%M%H%m%d%Y') + 'p' + str(int(opts.sigma*1000))
+ 's' + \
        str(int(opts.ratio*100)) + 'r' + str(opts.max_iters) + \
    'max' + str(opts.inlier_tol) + 'tol'
    dirname = "../data/" + param + "/"
    os.mkdir(dirname)
```

```
# 435,437 prone to get problems, specialize parameters for them
for i in range(0, v_len//v_len_ratio):
    if i%30 == 0:
        print(i)
    elif i == 395:
        opts.sigma = 0.106
        opts.ratio = 0.8
        opts.max_iters = 1000
        opts.inlier_tol = 2.0
    elif i == 401:
        opts.sigma = 0.15
        opts.ratio = 0.7
        opts.max_iters = 500
        opts.inlier_tol = 2.0
    elif i == 435:
        opts.sigma = 0.106
        opts.ratio = 0.8
        opts.max_iters = 2000
        opts.inlier_tol = 2.0
    elif i == 437:
        opts.sigma = 0.105
    elif i == 439:
        opts.sigma = 0.15
        opts.ratio = 0.7
        opts.max_iters = 500
        opts.inlier_tol = 2.0


    # compute H
    dst_frame = dist[i]
    src_frame = src[i]
    matches, desc1_locs, desc2_locs = matchPics(cv_cover, dst_frame, opts)
    m1 = []
    m2 = []
    for match in matches:
        m1.append(desc1_locs[match[0]])
        m2.append(desc2_locs[match[1]])
    m1 = np.array(m1)
    m2 = np.array(m2)
    # swap x, y
    m1[:, [1, 0]] = m1[:, [0, 1]]
    m2[:, [1, 0]] = m2[:, [0, 1]]
    # bestH2to1, inliers = computeHCV2(m1, m2, cv2.RANSAC)
    bestH2to1, inliers = computeH_ransac(m1, m2, opts)
```

```
        # crop source video, h,w = 360 * 286
        cropped = src_frame[:,int(src_shape[1]/2 - crop_shape[1]/2):int(src_shape[1]/2
+ crop_shape[1]/2)]
        # fix not filling up problem
        cropped = cv2.resize(cropped, (cv_cover.shape[1], cv_cover.shape[0]))
        # cv2.imshow('out', cropped)
        # cv2.imshow('dst', dst_frame)
        # cv2.waitKey(0)
        composite_img = compositeH(bestH2to1, cropped, dst_frame)
        output.append(composite_img)
        cv2.imwrite(os.path.join(dirname, '{}.jpg'.format(i)), composite_img)

    # reference: https://www.geeksforgeeks.org/saving-a-video-using-opencv/
    # width, height
    result = cv2.VideoWriter('../result/ar.avi',
                             cv2.VideoWriter_fourcc(*'MJPG'),
                             fps, (v_shape[1], v_shape[0]))
    for frame in output:
        result.write(frame)
    result.release()
```

frame 390, book at center

frame 323, book on right edge



frame 475, book on left edge

2. Extra

Q4

1.
```
import numpy as np
import cv2
from helper import computeBrief
from helper import briefMatch
from opts import get_opts
from displayMatch import plotMatches
from matchPics import matchPics
from planarH import computeHCV2, compositeH, computeH_ransac
from skimage.feature import match_descriptors, match_descriptors, SIFT
from skimage.transform import warp
import pdb

opts = get_opts()
ratio = opts.ratio
left = cv2.imread('../data/pano_left.jpg')
right = cv2.imread('../data/pano_right.jpg')
# left = cv2.imread('../data/pano_left_scaled.jpg')
# right = cv2.imread('../data/pano_right_scaled.jpg')

# reference:
# https://scikit-image.org/docs/stable/auto_examples/features_detection/plot_sift.html
# https://scikit-image.org/docs/stable/auto_examples/registration/plot_stitching.html

lg = cv2.cvtColor(left,cv2.COLOR_BGR2GRAY)
rg = cv2.cvtColor(right,cv2.COLOR_BGR2GRAY)

descriptor_extractor = SIFT()

descriptor_extractor.detect_and_extract(lg)
locsl = descriptor_extractor.keypoints
descl = descriptor_extractor.descriptors

descriptor_extractor.detect_and_extract(rg)
locsr = descriptor_extractor.keypoints
descr = descriptor_extractor.descriptors

# TODO: Match features using the descriptors
matches = match_descriptors(descl,descr,cross_check=True,max_ratio=0.5)
plotMatches(left, right, matches, locsl, locsr)

m1 = []
```

```
m2 = []
for match in matches:
    m1.append(locsl[match[0]])
    m2.append(locsr[match[1]])
m1 = np.array(m1)
m2 = np.array(m2)
# swap x, y
m1[:, [1, 0]] = m1[:, [0, 1]]
m2[:, [1, 0]] = m2[:, [0, 1]]
# we want stitch r to l
# H, _ = computeHCV2(m2, m1, cv2.RANSAC)
H, inliers = computeH_ransac(m2, m1, opts)
# create padding
width, height, _ = left.shape
marginh = height
marginw = width
out_shape = height +  marginh, width + marginw
glob_trfm = np.eye(3)
glob_trfm[:2, 2] = marginw/6, marginh/6

left_warp = cv2.warpPerspective(left, glob_trfm, out_shape, flags=cv2.INTER_LINEAR)
# cv2.imshow('left_warp', left_warp)
right_warp = cv2.warpPerspective(right, H, out_shape, flags=cv2.INTER_LINEAR)
# cv2.imshow('right', right_warp)
# cv2.waitKey(0)
right_H = cv2.warpPerspective(right_warp, glob_trfm, out_shape, flags=cv2.INTER_LINEAR)
# cv2.imshow('right', right_H)
# cv2.waitKey(0)
# self-process image addition to solve intensity difference
out = np.zeros(shape=(out_shape[1], out_shape[0], 3), dtype=np.uint8)
for r in range(left_warp.shape[0]):
    for c in range(left_warp.shape[1]):
        if right_H[r, c].any() != 0:
            out[r, c] = right_H[r, c]
        else:
            out[r, c] = left_warp[r, c]
cv2.imshow('panorama', out)
cv2.waitKey(0)
cv2.imwrite('../data/panorama.jpg', out)
cv2.destroyAllWindows()
```

left original image

right original image

panorama result