Computer Vision Homework 2

Sagar Sachdev
Sagarsac

Q1.2.1
8 degrees of freedom.

Q1.2.2
4 point pairs.

Q1.2.3

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}, x_1 = \begin{bmatrix} a \\ b \\ 1 \end{bmatrix}, x_2 = \begin{bmatrix} c \\ d \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} a \\ b \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} * \begin{bmatrix} c \\ d \\ 1 \end{bmatrix}$$

$$-h_{11}c - h_{12}d - h_{13} + (h_{31}ca + h_{32}da + h_{33}a) = 0$$

$$-h_{21}c - h_{22}d - h_{23} + (h_{31}ca + h_{32}da + h_{33}a) = 0$$

Using Linear Algebra, this can be written as:

$$A_i h = 0$$

$$=> A_i = \begin{bmatrix} -c & -d & -1 & 0 & 0 & 0 & ac & ad & a \\ 0 & 0 & 0 & -c & -d & -1 & bc & bd & b \end{bmatrix}$$

$$h = \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix}$$

Q2.1.1

FAST and the Harris corner detectors are both used in corner detection. The FAST detector works by selecting a pixel and comparing the surrounding pixels and compares the pixel intensity to gauge whether something is a corner while the Harris corner detection works using a sliding window. As a result, FAST does not need to compute the lambda for an entire window and is, as a result much faster.

Q2.1.2

Brief descriptors create a patch around a selected point. BRIEF descriptors are very fast and allow for faster matching by comparing intensities of location pairs that are sampled along with smoother patches. Filter banks on the other hand require less memory but are computationally expensive. Histogram of Gradients can be used as a descriptor.

Q2.1.3

Nearest neighbors entails creating two sets. For the first set, a set of points are extracted from the image of interest. Then from a training dataset, a set of points are extracted. The descriptors are then computed and compared using the Nearest Neighbor approach.

Hamming distance is faster since XOR computations are quite efficient on CPUs

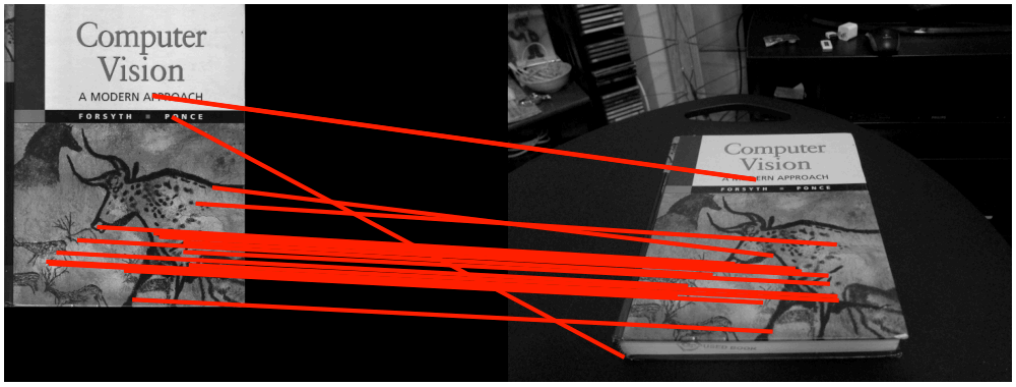
Q2.1.4

```python
import numpy as np
import cv2
import skimage.color
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 = cv2.cvtColor(I1, cv2.COLOR_BGR2GRAY)
        I2 = cv2.cvtColor(I2, cv2.COLOR_BGR2GRAY)

        # TODO: Detect Features in Both Images
        locs1 = corner_detection(I1, sigma)
        locs2 = corner_detection(I2, sigma)


        # TODO: Obtain descriptors for the computed feature locations
        desc_1, locs1 = computeBrief(I1, locs1)
        desc_2, locs2 = computeBrief(I2, locs2)

        # TODO: Match features using the descriptors
        matches = briefMatch(desc_1, desc_2, ratio)

        return matches, locs1, locs2
```

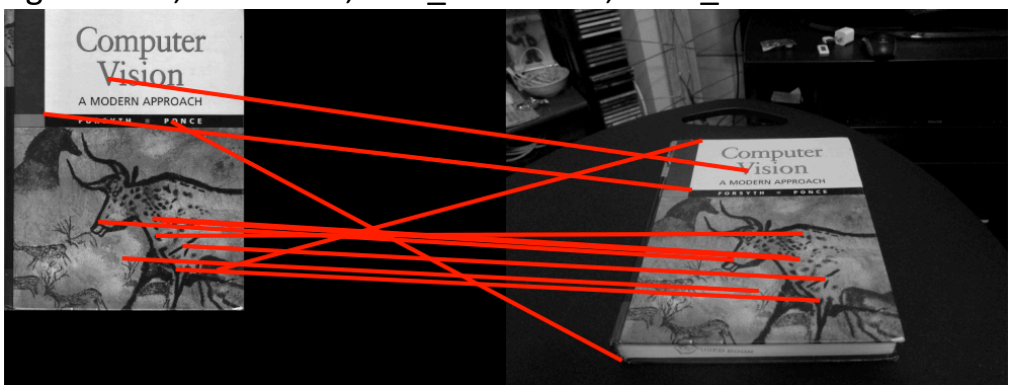
Q2.1.5

Default Params:

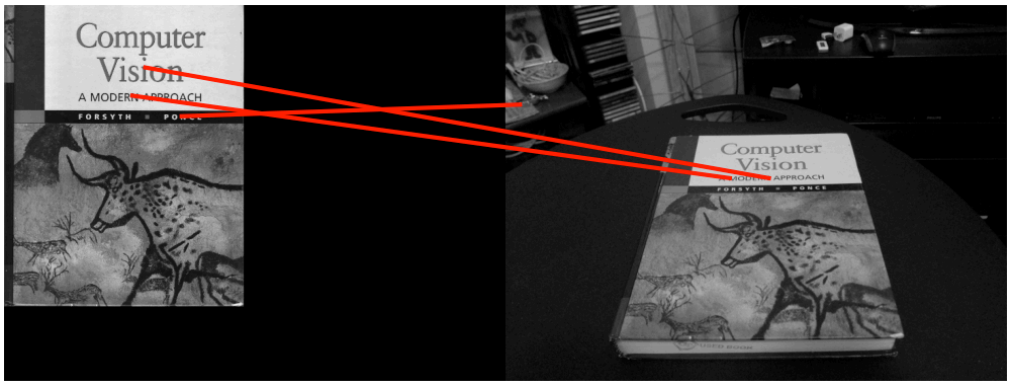
Sigma = 0.15, ratio = 0.7, max_iters = 500, inlier_tol = 2.0



Sigma = 0.2, ratio = 0.7, max_iters = 500, inlier_tol = 2.0

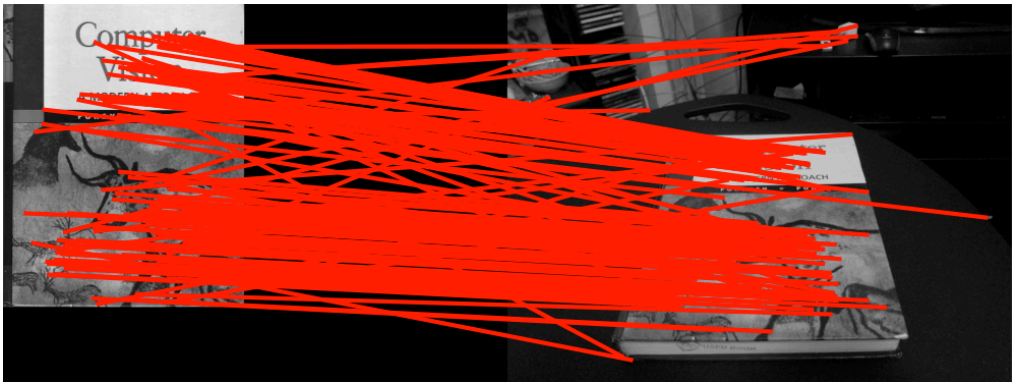


Sigma = 0.5, ratio = 0.7, max_iters = 500, inlier_tol = 2.0

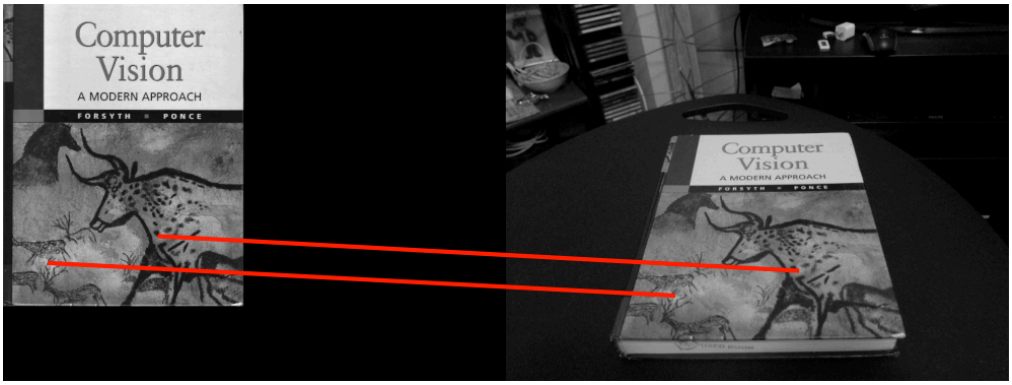


Then varying ratio:

Sigma = 0.15, ratio = 0.9, max_iters = 500, inlier_tol = 2.0



Sigma = 0.15, ratio = 0.5, max_iters = 500, inlier_tol = 2.0

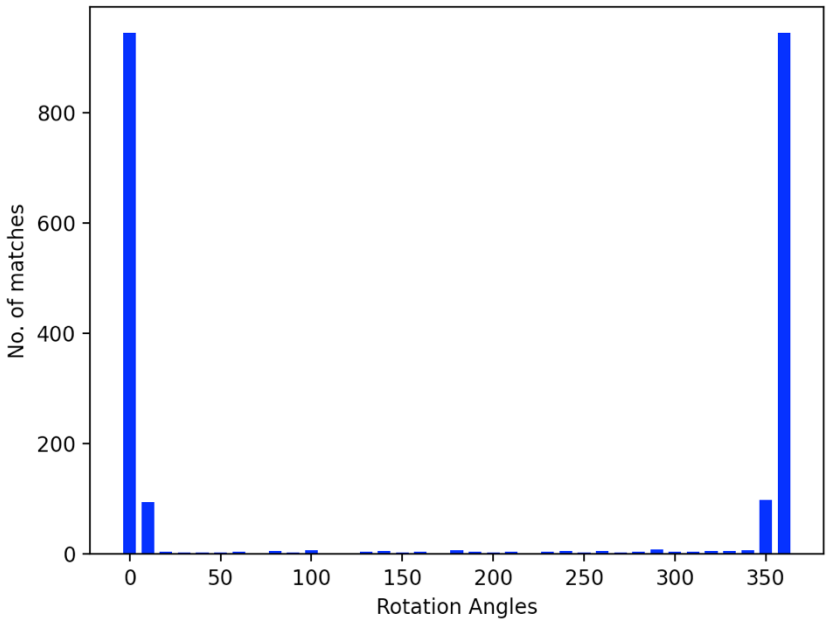


Sigma = 0.15, ratio = 0.2, max_iters = 500, inlier_tol = 2.0

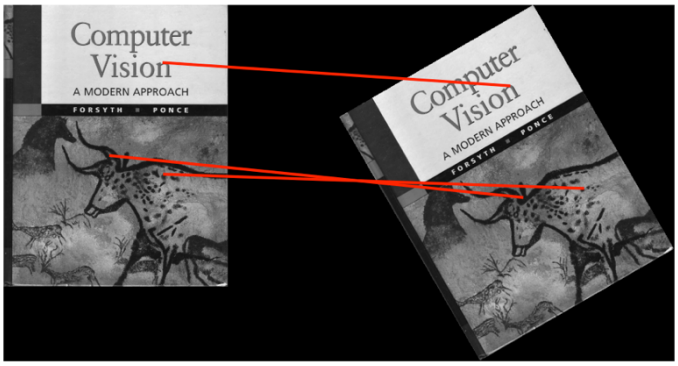
From this ablation study it can be concluded that as the sigma increases, the number of matched points decreases and vice-versa. In addition, as the ratio increases, the number of matched points increases and vice versa.

Q2.1.6

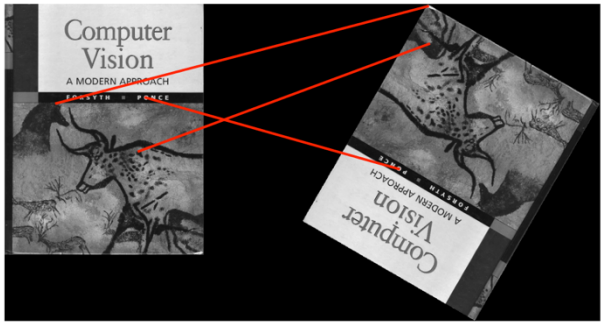
```python
import numpy as np
import cv2
import matplotlib.pyplot as plt
from matchPics import matchPics
from opts import get_opts
import scipy
from helper import plotMatches
from pylab import savefig
#Q2.1.6

img = cv2.imread('../data/cv_cover.jpg')
hist_update = []

def rotTest(img,opts):

    #Read the image and convert to grayscale, if necessary
    for i in range(37):

        #Rotate Image
        rot = scipy.ndimage.rotate(img,10*i,reshape=True)

        #Compute features, descriptors and Match features
        matches, locs1, locs2 = matchPics(img, rot, opts)

        #Update histogram
        hist_update.append(len(matches))

        if i == 0:
            n = plotMatches(img, rot, matches, locs1, locs2)
            # n.savefig('../results/img_0.jpg')
        if i == 3:
            n = plotMatches(img, rot, matches, locs1, locs2)
            # n.savefig('../results/img_3.jpg')
        if i == 15:
           n = plotMatches(img, rot, matches, locs1, locs2)
           # n.savefig('../results/img_15.jpg')
        if i == 27:
            n = plotMatches(img, rot, matches, locs1, locs2)
            # n.savefig('../results/img_27.jpg')
        if i == 36:
            n = plotMatches(img, rot, matches, locs1, locs2)
            # n.savefig('../results/img_36.jpg')
        # hist, bins = np.histogram(hist_update,37,(0,360))
        # print(rot.shape)
        # print(hist_update)
    #Display histogram
    # plt.hist(hist_update)
    return hist_update

if __name__ == "__main__":

    opts = get_opts()
    hist_update = rotTest(img,opts)
    plt.bar(np.arange(0,370,10), hist_update, color='blue', width=7)
    plt.ylabel('No. of matches')
    plt.xlabel('Rotation Angles')
    plt.show()
```



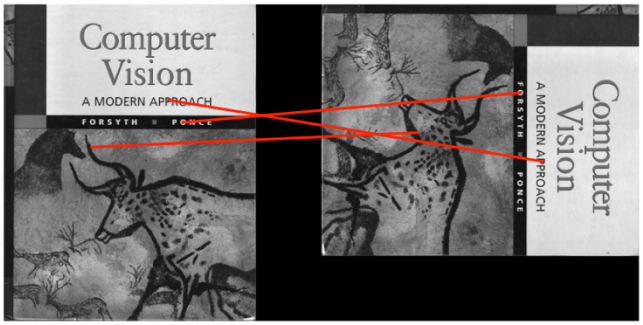
0 degree Rotation



30 degree rotation



150 degree rotation



270 degree rotation

360 degree Rotation

In the above histogram it can be seen that as the image is rotated, the number of matches decreases until it is completely rotated back to its original position (360 degrees). This is because BRIEF creates a patch around a pixel and randomly compares its distance to some randomly selected descriptors. Rotation causes different descriptors, and the result is a mismatch.

Q2.2.1

```
def computeH(x1, x2):
    #Q2.2.1
    #Compute the homography between two sets of points

    A = []
    for i in range(len(x1)):
        A.append([x1[i,0], x1[i,1],1, 0, 0, 0, -x2[i,0]*x1[i,0], x2[i,0]*x1[i,1], -x2[i,0]])
        A.append([0, 0, 0, x1[i,0], x1[i,1], 1, -x2[i,1]*x1[i,0], x2[i,1]*x1[i,1],-x2[i,1]])

    np.array(A)

    U,Sigma,V_transpose = np.linalg.svd(A)

    H2to1 = V_transpose[-1,:]
    H2to1 = np.reshape((H2to1),(3,3))

    return H2to1
```

Q2.2.2

```python
def computeH_norm(x1, x2):
    #Q2.2.2
    #Compute the centroid of the points
    x1_x = np.mean(x1[:,0])
    x1_y = np.mean(x1[:,1])
    x2_x = np.mean(x2[:,0])
    x2_y = np.mean(x2[:,1])

    x1_centroid = [x1_x,x1_y]
    x2_centroid = [x2_x,x2_y]

    #Shift the origin of the points to the centroid
    x1_shifted = x1-x1_centroid
    x2_shifted = x2-x2_centroid

    #Normalize the points so that the largest distance from the origin is equal to sqrt(2)

    # x1_norm = np.linalg.norm(x1_centroid)
    # x2_norm = np.linalg.norm(x2_centroid)


    # # x1_scaled = x1_norm*sqrt(2)/
    # if x1_norm > x2_norm:
    #     x1_scaled = np.testing.assert_equal(x1_norm,np.sqrt(2))
    #     x2_scaled = (x2_norm * np.sqrt(2))/x1_norm
    # else:
    #     x2_scaled = np.testing.assert_equal(x2_norm,np.sqrt(2))
    #     x1_scaled = (x1_norm*np.sqrt(2))/x2_norm

    # print(x1_scaled)
    # print(x2_scaled)
    # print(x1_norm)
    #Similarity transform 1

    x1_norm = np.sqrt((x1_shifted[:,0]**2)+(x1_shifted[:,1]**2))
    x2_norm = np.sqrt((x2_shifted[:,0]**2)+(x2_shifted[:,1]**2))

    x1_scaled = np.sqrt(2)/np.max(x1_norm)
    x2_scaled = np.sqrt(2)/np.max(x2_norm)


    A1 = np.eye(3)*x1_scaled
    A2 = np.array([[1,0,-x1_x],[0,1,-x1_y],[0,0,1]])

    T1 = np.matmul(A1,A2)

    #Similarity transform 2
    A3 = np.eye(3)*x2_scaled
    A4 = np.array([[1,0,-x2_x],[0,1,-x2_y],[0,0,1]])

    T2 = np.matmul(A3,A4)
    #Compute homography
    H = computeH(x1_shifted,x2_shifted)

    #Denormalization

    H2to1 = np.matmul(np.linalg.inv(T1),H,T2)

    return H2to1
```

Q2.2.3

```python
def computeH_ransac(locs1, locs2, opts):
    #Q2.2.3
    #Compute the best fitting homography given a list of matching points
    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
    #converting locs from (y,x) to (x,y)
    locs1 = locs1[:,[1,0]]
    locs2 = locs2[:,[1,0]]

    l1 = np.hstack((locs1,np.ones((locs1.shape[0], 1))))
    l2 = np.hstack((locs2,np.ones((locs2.shape[0], 1))))

    inlier_no = 0

    for i in range(max_iters):
        np.random.seed(seed=None)
        point_pairs = np.random.choice(locs1.shape[0],4,False)
        locs1_pred = locs1[point_pairs,:]
        locs2_pred = locs2[point_pairs,:]

        H = computeH(locs1_pred,locs2_pred)
        l_1 = (np.matmul(H,l2.T))
        last_col = l_1[2,:].T
        l_1 = l_1.T/last_col[:,None]

        # l_1 = np.transpose(l_1.T/(l_1[2,:])[:,None])


        inliers = np.zeros(locs1.shape[0])
        # c_inlier = np.zeros(len(inliers))

        err_l1 = np.linalg.norm(l1 - l_1,axis=1)
        inliers = np.sum(err_l1<inlier_tol)


        # for j in range(len(inliers)):
        # #     err_l1 = np.linalg.norm(l1 - l_1)
        #     if err_l1 < inlier_tol:
        #         c_inlier[i] = 1
        #         i+=1

        if inliers>=inlier_no:
            bestH2to1 = H
            # inliers = c_inlier
            inlier_no = inliers


        # # err_l2 = np.linalg.norm(l_2 - l2)

    return bestH2to1, inliers
```

Q2.2.4

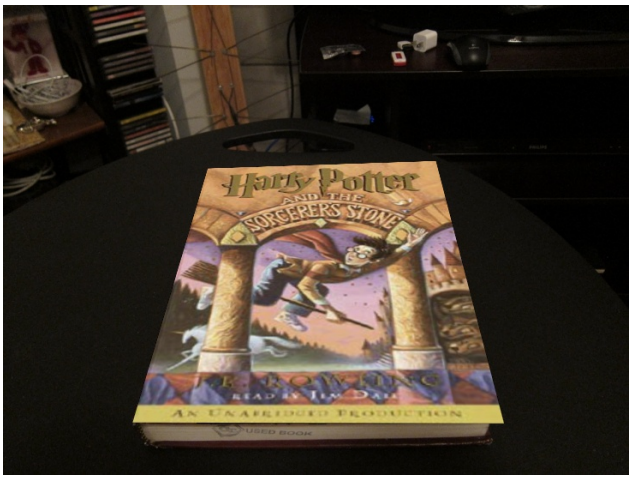
```python
import numpy as np
import cv2
import skimage.io
import skimage.color
from opts import get_opts
from planarH import computeH_ransac
from planarH import compositeH
from matchPics import matchPics
import matplotlib.pyplot as plt

# Import necessary functions

# Q2.2.4
cv_cover = cv2.imread('../data/cv_cover.jpg')
cv_desk = cv2.imread('../data/cv_desk.png')
hp_cover = cv2.imread('../data/hp_cover.jpg')

hp_resize = cv2.resize(hp_cover,(cv_cover.shape[1],cv_cover.shape[0]))

def warpImage(opts):

    pass

if __name__ == "__main__":
    opts = get_opts()
    matches, locs1, locs2 = matchPics(cv_desk,cv_cover,opts)
    best_H2to1, inliers = computeH_ransac(locs2[matches[:,1]],locs1[matches[:,0]],opts)

    composite_img = compositeH(best_H2to1, hp_resize, cv_desk)
    cv2.imwrite('../results/hp_new.jpg',composite_img)
```

Script for HarryPotterize.py

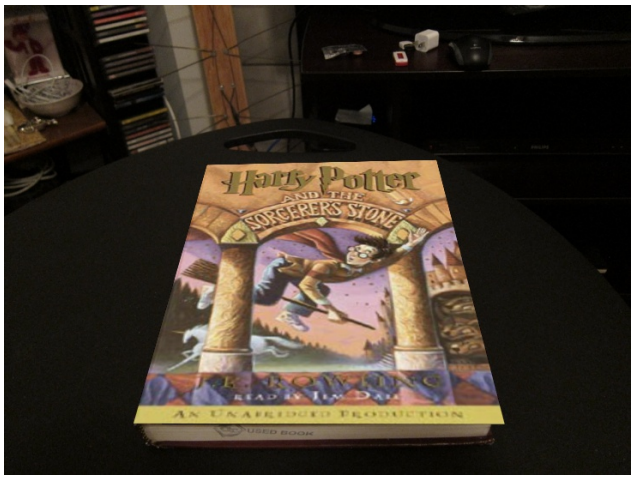
```python
def compositeH(H2to1, template, img):

    #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.
    #Create mask of same size as template

    h = img.shape[0]
    w = img.shape[1]

    mask = np.ones((template.shape))
    #Warp mask by appropriate homography
    mask = cv2.warpPerspective(mask, np.linalg.inv(H2to1), (w,h))



    # idx = np.nonzero(mask.T)
    #Warp template by appropriate homography
    warp_homography = cv2.warpPerspective(template, np.linalg.inv(H2to1), (w,h))
    idx = mask == 0
    idx = idx.astype(int)
    #Use mask to combine the warped template and the image

    composite_img = idx*img+warp_homography

    # warp_homography = warp_homography.T


    # img[idx[0],0] = warp_homography[idx[0],0]
    # img[idx[1],1] = warp_homography[idx[1],1]
    # img[idx[2],2] = warp_homography[idx[2],2]

    # composite_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    return composite_img
```

Script for composite.py

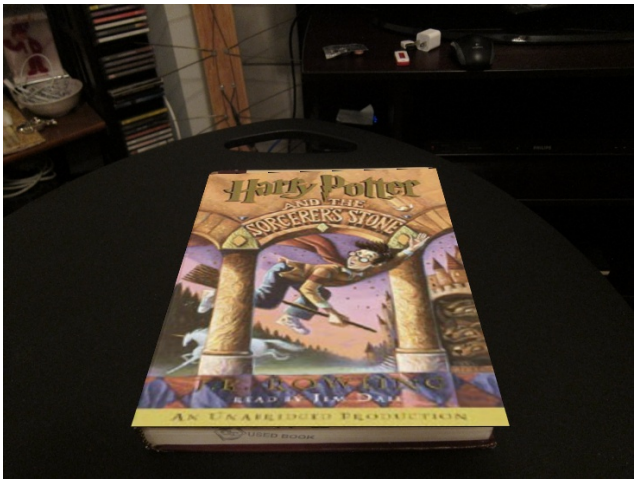
Q2.2.5:



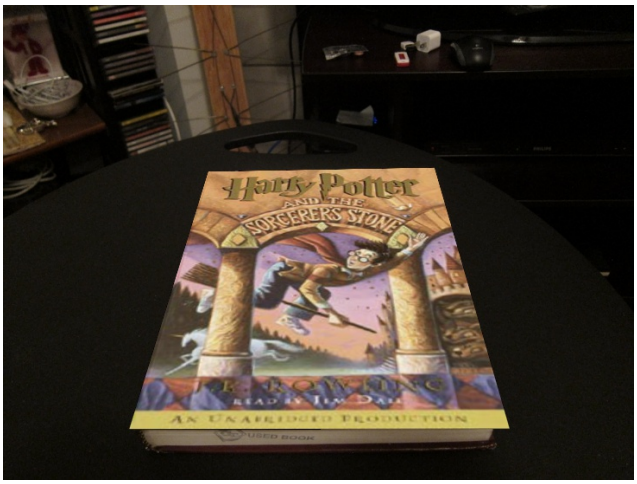
Max_iterations = 500, inlier_tol = 2.0



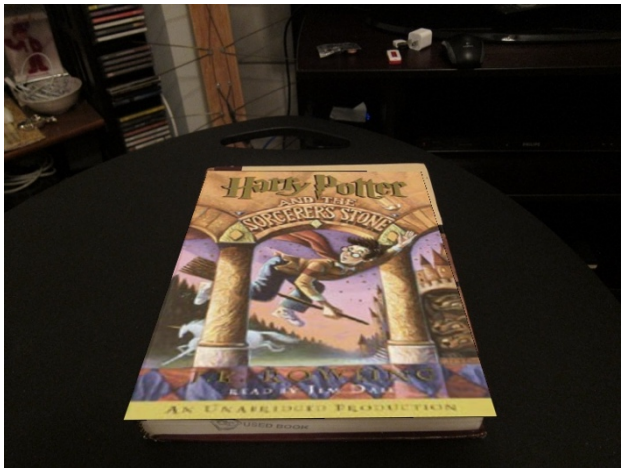
Max_iterations = 1000, inlier_tol = 2.0



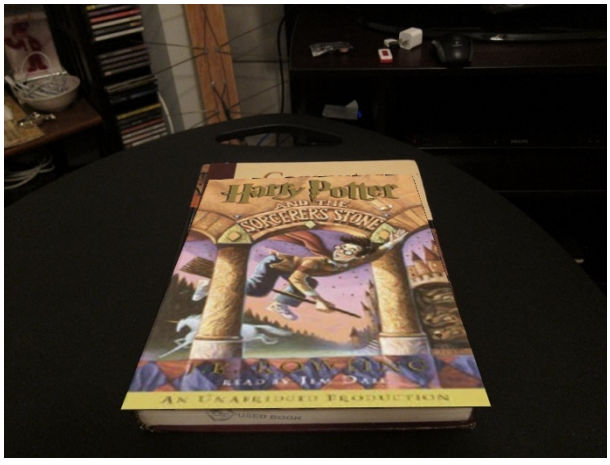
Max_iterations = 7000, inlier_tol = 2.0



Max_iterations = 50, inlier_tol = 2.0



Max_iterations = 500, inlier_tol = 5.0



Max_iterations = 500, inlier_tol = 20

As the number of iterations increases, the number of inlier points that can be matched also increase. As a result, it can be seen that the product of higher iterations leads to a more aptly matched image. In addition, as the inlier_tolerance increases, the inliers that can be matched decreases and this leads to a worse fit image as can be evidenced by the image with max_iterations = 500 and inlier_tol = 20

Q3.1

```python
import time
import numpy as np
import cv2
import multiprocessing as mp
from itertools import repeat
import multiprocessing as mp
from itertools import repeat
from opts import get_opts
import skimage.io
import matplotlib.pyplot as plt

#Import necessary functions
from helper import loadVid
from helper import plotMatches
from planarH import computeH_ransac
from planarH import compositeH
from matchPics import matchPics
import imageio


start = time.time()
def frame_gen(opts,book,ar_vid,cv_cover):
    # print('frame time')
    # print(book.shape)
    matches, locs1, locs2 = matchPics(np.array(book),np.array(cv_cover),opts)
    best_H2to1, inliers = computeH_ransac(locs2[matches[:,1]],locs1[matches[:,0]],opts)
    composite_img = compositeH(best_H2to1, ar_vid, book)
    # print('aaaa')
    return composite_img


opts = get_opts()

#Write script for Q3.1
#Load all required images and videos
book = loadVid('../data/book.mov')
ar_vid = loadVid('../data/ar_source.mov')
cv_cover = cv2.imread('../data/cv_cover.jpg')


#compute the aspect ratio:
aspect_ratio = cv_cover.shape[1]/cv_cover.shape[0]

#Removing black space from the ar_vid
pixel_reject = np.where(np.sum(np.sum(ar_vid[1,:,:,:] == 0, 1),1) > 700)
video_frames = np.delete(ar_vid,pixel_reject,axis=1)

#computing dimensions of the book
book_height = book.shape[1]
book_width = book.shape[0]

#Since we need the video to be in the center of the book\
video_center = int(book_width/2)

#we need to now crop the video such that it is in the center

video_frames = video_frames[:,:,130:510,:]

#Multiprocessing script \

# output = cv2.VideoWriter('../results/ar.avi',fourcc = cv2.VideoWriter_fourcc('X','V','I','D'),fps=30,)
if __name__== '__main__':
    n_worker = mp.cpu_count()
    with mp.Pool(processes=n_worker) as pool:
        frames = pool.starmap(frame_gen, zip(repeat(opts),repeat(book), repeat(video_frames), repeat(cv_cover)))
        imageio.mimwrite('../result/ar.avi', frames, fps=30)



    #Write out the video and store them

end = time.time()
print('Elapsed Time: {}'.format(end - start))
```

Acknowledgements:

1) My friend Filip Nowicki helped me with the AR portion of the assignment
2) My friend from my lab, Bassam Bikdash helped me write the multiprocessing portion of the assignment since my implementation did not work for Assignment 1.
3) https://stackoverflow.com/questions/69655705/runtimeerror-python-multiprocessing-error