16-720 Computer Vision: Homework 3 (Fall 2022)

Augmented Reality with Planar Homographies

Haejoon Lee

Q1.1 (5 points): Prove that there exists a homography $\mathbf{H}$ that satisfies equation 1 given two 3×4 camera projection matrices $\mathbf{P}_1$ and $\mathbf{P}_2$ corresponding to the two cameras and a plane Π. You do not need to produce an actual algebraic expression for $\mathbf{H}$. All we are asking for is a proof of the existence of $\mathbf{H}$.

$$\mathbf{x}_1 \equiv \mathbf{H}\mathbf{x}_2 \tag{1}$$

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & s_\theta & o_x \\ 0 & s_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad \text{Plug in Z=0}$$

↳ camera metrics

$$= \begin{bmatrix} m_{11} & m_{12} & m_{14} \\ m_{21} & m_{22} & m_{24} \\ m_{31} & m_{32} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

$$\lambda_1 \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \lambda_1 \vec{x_1} = P_1 \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = P_1 X \quad , \quad \lambda_2 \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \lambda_2 \vec{x_2} = P_2 \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = P_2 X$$

$$x_1 \equiv P_1 P_2^{-1} X_2$$

Q1.2 (15 points): Let $\mathbf{x_1}$ be a set of points in an image and $\mathbf{x_2}$ be the set of corresponding points in an image taken by another camera. Suppose there exists a homography $\mathbf{H}$ such that:

$$\mathbf{x_1^i} \equiv \mathbf{H}\mathbf{x_2^i} \quad (i \in \{1 \dots N\})$$

where $\mathbf{x_1^i} = [x_1^i \;\; y_1^i \;\; 1]$ are in homogenous coordinates, $\mathbf{x_1^i} \in \mathbf{x_1}$ and $\mathbf{H}$ is a 3×3 matrix. For each point pair, this relation can be rewritten as

$$\mathbf{A_i h} = 0$$

where $\mathbf{h}$ is a column vector reshaped from $\mathbf{H}$, and $\mathbf{A_i}$ is a matrix with elements derived from the points $\mathbf{x_1^i}$ and $\mathbf{x_2^i}$. This can help calculate $\mathbf{H}$ from the given point correspondences.

1. How many degrees of freedom does $\mathbf{h}$ have? (3 points)

Ans: 8

2. How many point pairs are required to solve $\mathbf{h}$? (2 points)

Ans: 4

3. Derive $\mathbf{A_i}$. (5 points)

$$\lambda \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$$

$$x_1 = \frac{ax_2 + by_2 + c}{gx_2 + hy_2 + i}, \quad ax_2 + by_2 + c - gx_1x_2 - hx_1y_2 - ix_1 = 0$$

$$y_1 = \frac{dx_2 + ey_2 + f}{gx_2 + hy_2 + i}, \quad dx_2 + ey_2 + f - gy_1x_2 - hy_1y_2 - iy_1 = 0$$

->

$$\begin{bmatrix} x_2^i & y_2^i & 1 & 0 & 0 & 0 & -x_1^i x_2^i & -x_1^i y_2^i & -x_1^i \\ 0 & 0 & 0 & x_2^i & y_2^i & 1 & -y_1^i x_2^i & -y_1^i y_2^i & -y_1^i \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \end{bmatrix} = A_i h$$

4. When solving $\mathbf{Ah} = 0$, in essence you're trying to find the $\mathbf{h}$ that exists in the null space of $\mathbf{A}$. What that means is that there would be some non-trivial solution for $\mathbf{h}$ such that that product $\mathbf{Ah}$ turns out to be 0. What will be a trivial solution for $\mathbf{h}$? Is the matrix $\mathbf{A}$ full rank? Why/Why not? What impact will it have on the singular values (i.e. eigenvalues of $\mathbf{A}^\top\mathbf{A}$)? (5 points)

The trivial solution for h will be h = 0

The matrix A will not be full rank if there are some non-trivial solutions thus Nullity(A) $\neq$ 0

If the smallest singular values is 0, then the corresponding eigen vector is the solution for Ah = 0.

There is no transition:

Q1.4.1 (5 points): Homography under rotation Prove that there exists a homography $\mathbf{H}$ that satisfies $\mathbf{x_1} \equiv \mathbf{Hx_2}$, given two cameras separated by a pure rotation. That is, for camera 1, $\mathbf{x_1} = \mathbf{K_1}[\mathbf{I}\ \mathbf{0}]\mathbf{X}$ and for camera 2, $\mathbf{x_2} = \mathbf{K_2}[\mathbf{R}\ \mathbf{0}]\mathbf{X}$. Note that $\mathbf{K_1}$ and $\mathbf{K_2}$ are the 3×3 intrinsic matrices of the two cameras and are different. $\mathbf{I}$ is 3×3 identity matrix, $\mathbf{0}$ is a 3×1 zero vector and $\mathbf{X}$ is a point in 3D space. $\mathbf{R}$ is the 3×3 rotation matrix of the camera.

$$X = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \rightarrow X' = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

$$\lambda_1 x_1 = K_1 \begin{bmatrix} I & 0 \end{bmatrix} X = K_1 X', \quad \lambda_2 x_2 = K_2 \begin{bmatrix} I & 0 \end{bmatrix} X = K_2 X'$$

->

$$\lambda_1 x_1 = \lambda_1 K_1 (K_2 R)^{-1} X_2 \Longrightarrow x_1 \equiv H x_2$$

Q1.4.2 (5 points): Understanding homographies under rotation Suppose that a camera is rotating about its center **C**, keeping the intrinsic parameters **K** constant. Let **H** be the homography that maps the view from one camera orientation to the view at a second orientation. Let θ be the angle of rotation between the two. Show that $\mathbf{H}^2$ is the homography corresponding to a rotation of 2θ. Please limit your answer within a couple of lines. A lengthy proof indicates that you're doing something too complicated (or wrong).

$$H = KR^{-1}K^{-1} \rightarrow H^2 = KR^{-1}K^{-1}KR^{-1}K^{-1} = K(R^2)^{-1}K^{-1}$$

$$R^2 \rightarrow rotationof2\theta$$

Q1.4.3 (5 points): Limitations of the planar homography Why is the planar homography not completely sufficient to map any arbitrary scene image to another viewpoint? State your answer concisely in

Ans: If the camera is heading exactly toward an edge of a plane, then it will appear as a line in a image and non-invertible.

Q1.4.4 (5 points): Behavior of lines under perspective projections We stated in class that perspective projection preserves lines (a line in 3D is projected to a line in 2D). Verify algebraically that this is the case, i.e., verify that the projection $\mathbf{P}$ in $\mathbf{x} = \mathbf{PX}$ preserves lines.

Let's express a line in 3D as below:

$$X = k + ta, Y = l + tb, Z = m + tc$$

t is a variable and k, a, l, b, m, c are constants

$$x = \frac{fX}{Z} = \frac{k + ta}{m + tc} \rightarrow t = \frac{mx - fk}{fa - cx}$$

$$y = \frac{fY}{Z} = \frac{l + tb}{m + tc} = \frac{bm - cl}{f(am - ck)}x + \frac{al - bk}{am - ck}$$

Thus, it's line in the image.

Q2.1.1 (5 points): FAST Detector How is the FAST detector different from the Harris corner detector that you've seen in the lectures? Can you comment on its computational performance compared to the Harris corner detector? Reference links: Original Paper, OpenCV Tutorial

Ans:

FAST detector detects key points by comparing pixel intensities. Therefore, it doesn't need to compute gradient image, det(), or trace() like Harris corner detector so could be faster.

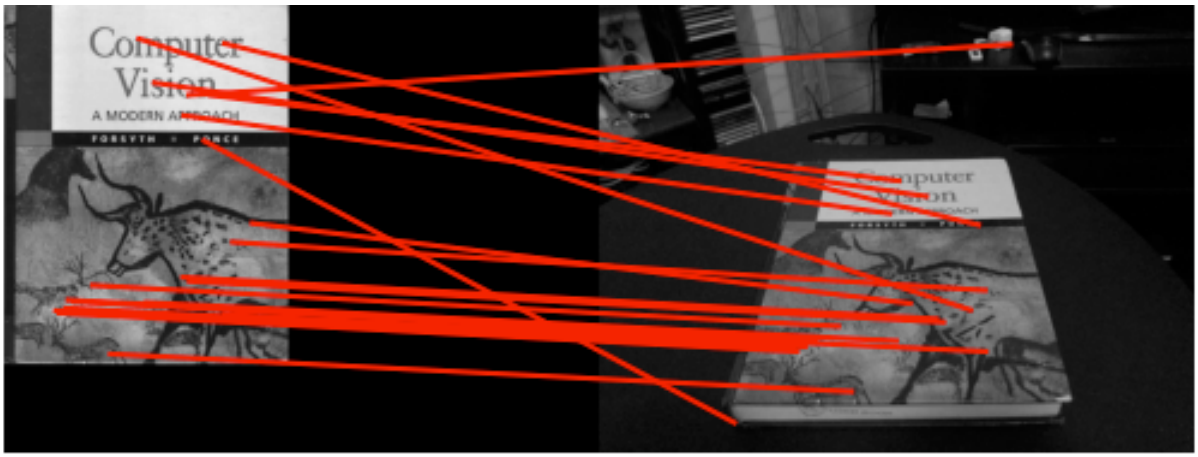
Q2.1.2 (5 points): BRIEF Descriptor How is the BRIEF descriptor different from the filterbanks you've seen in the lectures? Could you use any one of those filter banks as a descriptor?

BRIEF only provide binary vector as output by comparing pixel values, the other filter banks generate the result with constant values from convolution. I think edge detection filters could be used as a descriptor.

Q2.1.3 (5 points): Matching Methods The BRIEF descriptor belongs to a category called binary descriptors. In such descriptors the image region corresponding to the detected feature point is represented as a binary string of 1s and 0s. A commonly used metric used for such descriptors is called the Hamming distance. Please search online to learn about Hamming distance and Nearest Neighbor, and describe how they can be used to match interest points with BRIEF descriptors. What benefits does the Hamming distance distance have over a more conventional Euclidean distance measure in our setting?

By calculating Hamming distances between binary vectors of feature points in BRIEF descriptor, the nearest point is matched to one. Since the sequence of binary values in a vector is encoding which comparison result of two pixel points the value is indicating, Hamming distance was more appropriate than Euclidean.

Q2.1.4 (10 points): Feature Matching

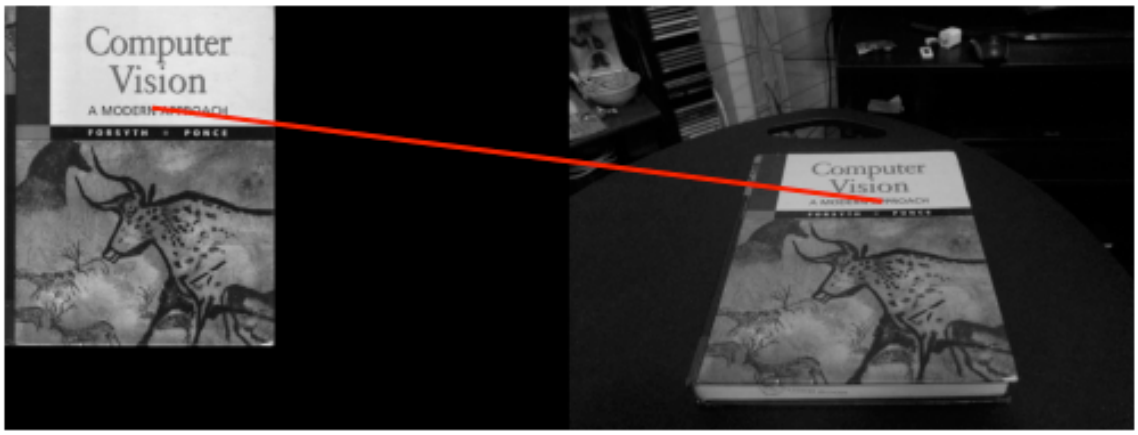



```python
def matchPics(I1, I2, opts):
    """
    Match features across images

    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'

    # sigma=float(0.15)
    # ratio=float(0.7)
    # max_iters=int(500)
    # inlier_tol=float(2.0)

    # TODO: Convert Images to GrayScale
    I1_gray = cv2.cvtColor(I1, cv2.COLOR_BGR2GRAY)
    I2_gray = cv2.cvtColor(I2, cv2.COLOR_BGR2GRAY)

    # TODO: Detect Features in Both Images
    locs1 = corner_detection(I1_gray, sigma)
    locs2 = corner_detection(I2_gray, sigma)

    # TODO: Obtain descriptors for the computed feature locations
    desc1, locs1 = computeBrief(I1_gray, locs1)
```

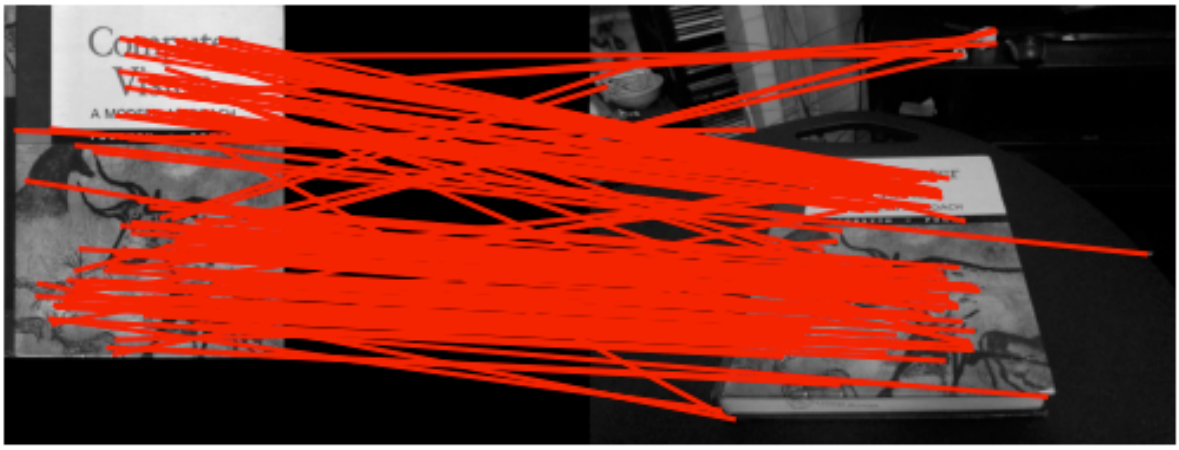
```
    desc2, locs2 = computeBrief(I2_gray, locs2)

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

    return matches, locs1, locs2
```

Q2.1.5 (10 points): Feature Matching and Parameter Tuning

Run the provided starter code `displayMatch.py` to display matched features. There are two tunable parameters, both stored in the `opts` variable, and are loaded from `opts.py`. You can change the values by changing their default fields or by command-line arguments. For example, `python displayMatch.py --sigma 0.15 --ratio 0.7`.



When I increased sigma value to 0.5, I got only one match. Because the sigma value is the threshold value for Fast detector, it will result in fewer detected points and match as it increased.



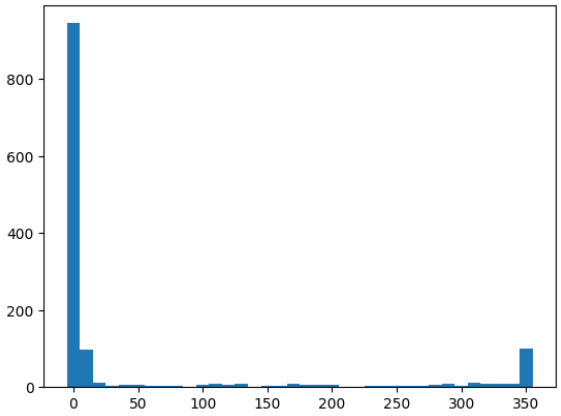
When I increased ratio to 0.9, I got much more matches. The value represent the maximum ratio of distances between first and second closest descriptor in the second set of descriptors. As increasing this threshold, more ambiguous matches between the two descriptor sets appeared.

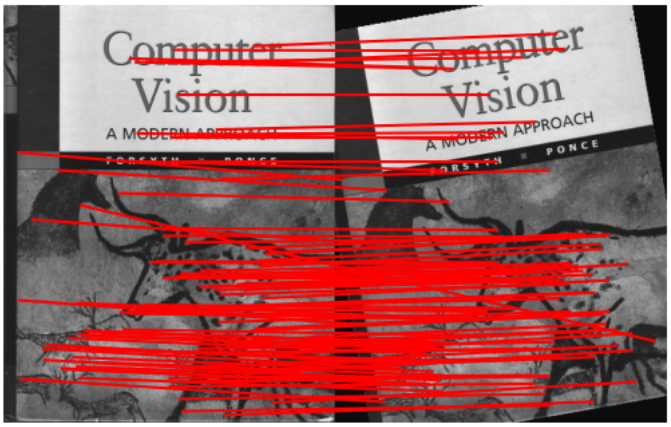
Q2.1.6 (10 points): BRIEF and Rotations

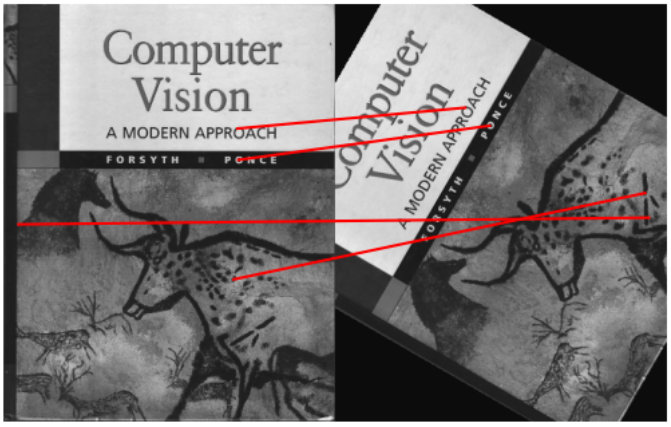
Let's investigate how BRIEF works with rotations. Write a script `briefRotTest.py` that:

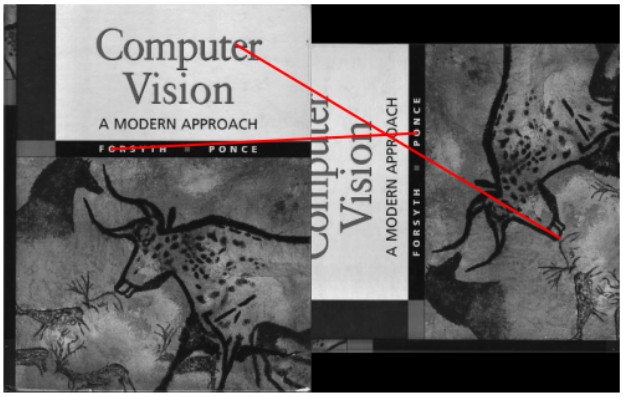
- Takes the `cv_cover.jpg` and matches it to itself rotated from 0 to 360 degrees in increments of 10 degrees. [*Hint*: use `scipy.ndimage.rotate`]
- Stores a histogram of the count of matches for each orientation.
- Plots the histogram using `matplotlib.pyplot.hist` (x axis: rotation, y axis: number of matches).

Visualize the histogram and the feature matching result at three different orientations and include them in your write-up. Explain why you think the BRIEF descriptor behaves this way. **Please include the code snippet in your writeup.**









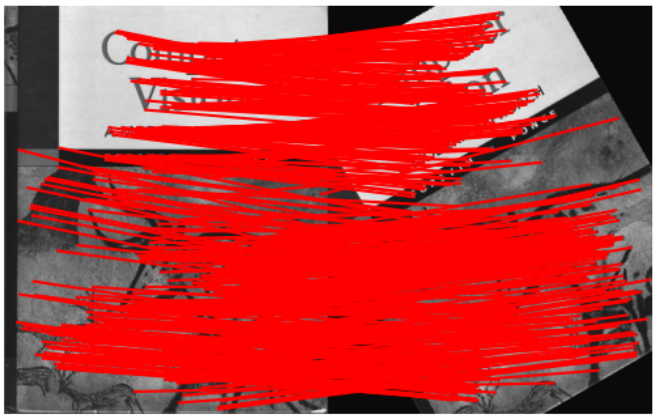

```
img_path = '../data/cv_cover.jpg'

img = Image.open(img_path).convert("RGB")
img = np.array(img).astype(np.float32) / 255 #normalize

# img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

counts_matches = []

for i in tqdm(range(36)):
    # Rotate Image
    #Compute features, descriptors and Match features
    rot_ang = i * 10
    img_rot = ndimage.rotate(img, rot_ang, reshape=False)
    matches, locs1, locs2 = matchPics(img, img_rot) #Should put opt!
    counts_matches.append(matches.shape[0])

plt.bar(range(0, 360, 10), counts_matches, width=10)
plt.show()
```

Q2.1.7 (Extra Credit - 10 points): Improving Performance The extra credit opportunities described below are optional and provide an avenue to explore computer vision and improve the performance of the techniques developed above.

1. **(5 pts)** As we have seen, BRIEF is not rotation invariant. Design a simple fix to solve this problem using the tools you have developed so far (think back to edge detection and/or Harris corner's covariance matrix). You are not allowed to use any additional OpenCV or Scikit-Image functions. Include the code in your PDF, and explain your design decisions and how you selected any parameters that you use. Demonstrate the effectiveness of your algorithm on image pairs related by large rotation.

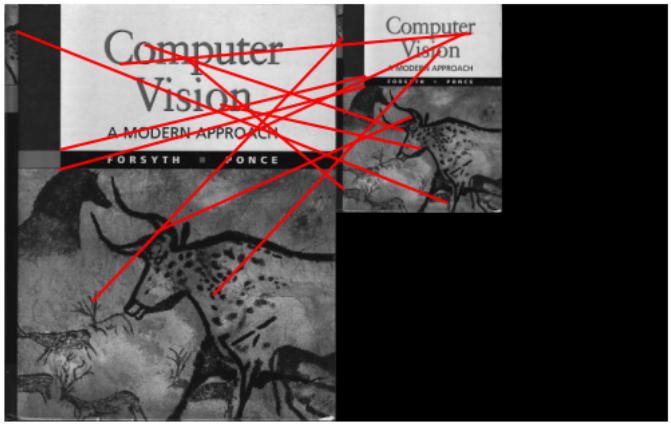
```
 # Q2.1.4
def matchPics_modi(I1, I2, angle_rot, opts):
      """
      Match features across images

      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'
      # ratio = 0.7  #'ratio for BRIEF feature descriptor'
      sigma = opts.sigma  #'threshold for corner detection using FAST feature
detector'

      # TODO: Convert Images to GrayScale
      I1_gray = cv2.cvtColor(I1, cv2.COLOR_BGR2GRAY)
      I2_gray = cv2.cvtColor(I2, cv2.COLOR_BGR2GRAY)
      I2_gray = ndimage.rotate(I2_gray, -angle_rot, reshape = False) # rotate I2
to be aligned with I1.

      # TODO: Detect Features in Both Images
      locs1 = corner_detection(I1_gray, sigma)
      locs2 = corner_detection(I2_gray, sigma)

      # TODO: Obtain descriptors for the computed feature locations
      desc1, locs1 = computeBrief(I1_gray, locs1)
      desc2, locs2 = computeBrief(I2_gray, locs2)

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

      R = np.array([[np.cos(angle_rot*(np.pi/180)), -
np.sin(angle_rot*(np.pi/180))],
                [np.sin(angle_rot*(np.pi/180)), np.cos(angle_rot*(np.pi/180))]])
```

```
        #move the center coordinate of locs2 to (0, 0)
        locs2[:,0] = locs2[:,0] - I2_gray.shape[0]/2
        locs2[:,1] = locs2[:,1] - I2_gray.shape[1]/2

        #rotate locs according to the original angle
        locs2 = (R @ locs2.T).T

        #relocate the center coordinate of locs2
        locs2[:,0] = locs2[:,0] + I2_gray.shape[0]/2
        locs2[:,1] = locs2[:,1] + I2_gray.shape[1]/2
#         locs2 = R @ locs2.T

        return matches, locs1, locs2
```

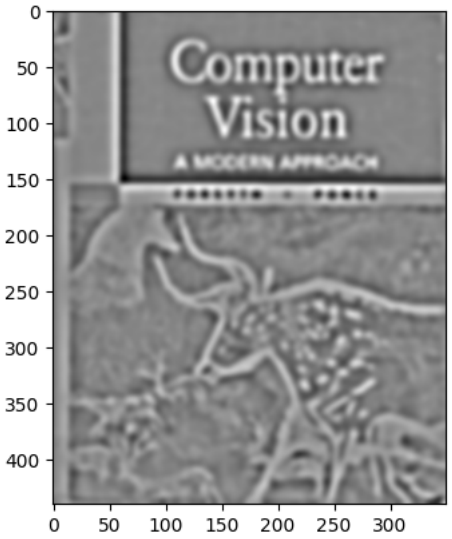


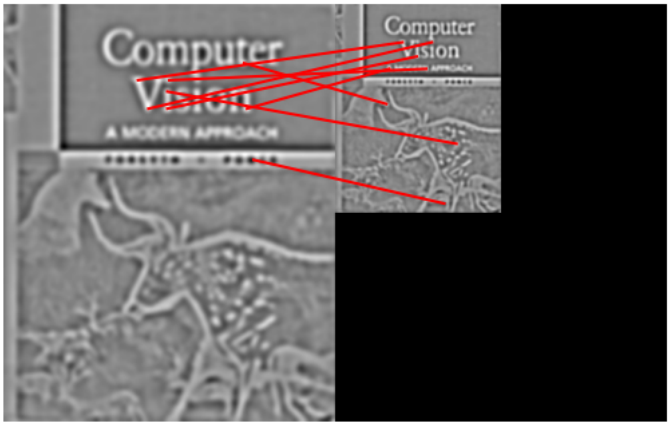
If we know the rotated angle, we can rotate the input image with the same angle by opposite direction and find matches. And then, rotate the coordinates of matches again. With this method, I could get rotate-invariante results as above.

2. **(5 pts)** This implementation of BRIEF has some scale invariance, but there are limits. What happens when you match a picture to the same picture at half the size? Look to section 3 of [Lowe2004], for a technique that will make your detector more robust to changes in scale. Implement it and demonstrate it in action with several test images. Include your code and the test images in your PDF. You are not allowed to call any additional OpenCV or Scikit-Image functions. You may simply rescale some of the test images we have given you.



As shown above, the BRIEF descriptor showed poor result when I halved the width and height of the image. Following the section 3 of Lowe2004, I computed DOG and perform detection.






```
img_dog = gaussian_filter(img, sigma=5, truncate=tr) - gaussian_filter(img, sigma=3, truncate=tr)

img_dog_min = img_dog.min(axis=(0, 1), keepdims=True)
img_dog_max = img_dog.max(axis=(0, 1), keepdims=True)
img_dog = (img_dog - img_dog_min) / (img_dog_max - img_dog_min)

# plt.imshow(img_dog)
img_dog_resize = cv2.resize(img_dog, (int(img_dog.shape[1]/2), int(img_dog.shape[0]/2)))

matches, locs1, locs2 = matchPics(img_dog, img_dog_resize, opts) #Should put opt!
plotMatches(img_dog, img_dog_resize, matches, locs1, locs2)
```

Q2.2.1 (15 points): Computing the Homography

Write a function `computeH` in `planarH.py` that estimates the planar homography from a set of matched point pairs.

```
H2to1 = computeH(x1, x2)
```

`x1` and `x2` are $N \times 2$ matrices containing the coordinates (x, y) of point pairs between the two images. `H2to1` should be a 3×3 matrix for the best homography from image 2 to image 1 in the least-squares sense. The `numpy.linalg` function `eig()` or `svd()` will be useful to get the eigenvectors (see Section 1 of this handout for details). **Please include the code snippet in your writeup.**

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

    indx_rand = np.random.randint(len(x1), size=4)
    A = []
    for i in indx_rand:
        xcor2, ycor2 = x2[i]
        xcor1, ycor1 = x1[i]

        Ai = np.array([[xcor2, ycor2, 1, 0, 0, 0, -xcor1 * xcor2, -xcor1 * ycor2, -
xcor1],
                       [0, 0, 0, xcor2, ycor2, 1, -ycor1 * xcor2, -ycor1 * ycor2, -
ycor1]])
        A.append(Ai)

    A = np.vstack(A)

    U, S, V = np.linalg.svd(A)
    H2to1 = V.T[:, -1].reshape([3, 3])

    return H2to1
```

Q2.2.2 (10 points): Homography Normalization

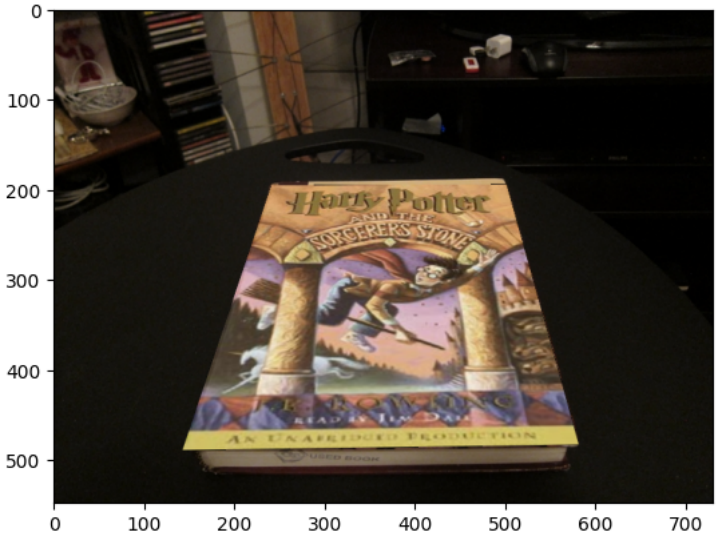
```python
def computeH_norm(x1, x2):
    #Q2.2.2
    #Compute the centroid of the points
    mean1 = np.mean(x1, axis=0)
    mean2 = np.mean(x2, axis=0)
    #Shift the origin of the points to the centroid
    x1 = x1 - mean1
    x2 = x2 - mean2

    #Normalize the points so that the largest distance from the origin is equal to
sqrt(2)
    max1 = np.max(np.linalg.norm(x1, axis=1))
    max2 = np.max(np.linalg.norm(x2, axis=1))

    if max1 == 0:   #if max = 0 -> # of point in x is single and its value is 0
        c1 = 1
    else:
        c1 = 1 / (max1 / 2 ** 0.5)
    if max2 == 0:
        c2 = 1
    else:
        c2 = 1 / (max2 / 2 ** 0.5)

    x1 *= c1
    x2 *= c2

    #Similarity transform 1
    T1 = np.array([[c1, 0, -c1 * mean1[0]], [0, c1, -c1 * mean1[1]], [0, 0, 1]])

    #Similarity transform 2
    T2 = np.array([[c2, 0, -c2 * mean2[0]], [0, c2, -c2 * mean2[1]], [0, 0, 1]])

    #Compute homography
    h = computeH(x1, x2)

    #Denormalization
    H2to1 = np.linalg.inv(T1).dot(h).dot(T2)

    return H2to1
```

Q2.2.3 (25 points): Implement RANSAC

```
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

   locs1_hc = np.hstack((locs1, np.ones((len(locs1), 1))))
   locs2_hc = np.hstack((locs2, np.ones((len(locs2), 1))))

   for i in range(0, max_iters):
      H = computeH_norm(locs1, locs2)

      locs2_warped = H @ locs2_hc.T
      locs2_warped /= locs2_warped[2, :]
      locs2_warped[2, :] = np.ones(locs2_warped.shape[1])

      error_sub = locs1_hc.T - locs2_warped
      error_squar = np.sum(error_sub ** 2, axis=0)

      inlier_idx = (error_squar <= inlier_tol**2).astype(int)

      if i == 0:
        bestH2to1 = H
        inliers_best = inlier_idx

      if np.sum(inlier_idx) > np.sum(inliers_best):
         bestH2to1 = H
         inliers_best = inlier_idx
         # print(inlier_idx)

   return bestH2to1, inliers_best
```

Q2.2.4 (10 points): Automated Homography Estimation and Warping

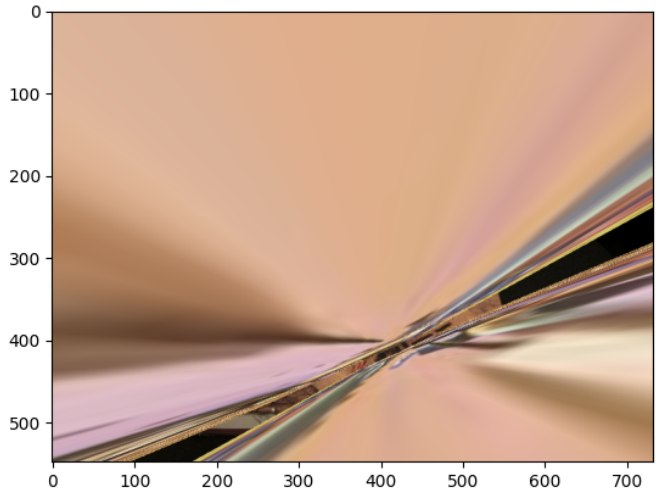
Write a function `warpImage()` in `HarryPotterize.py` that



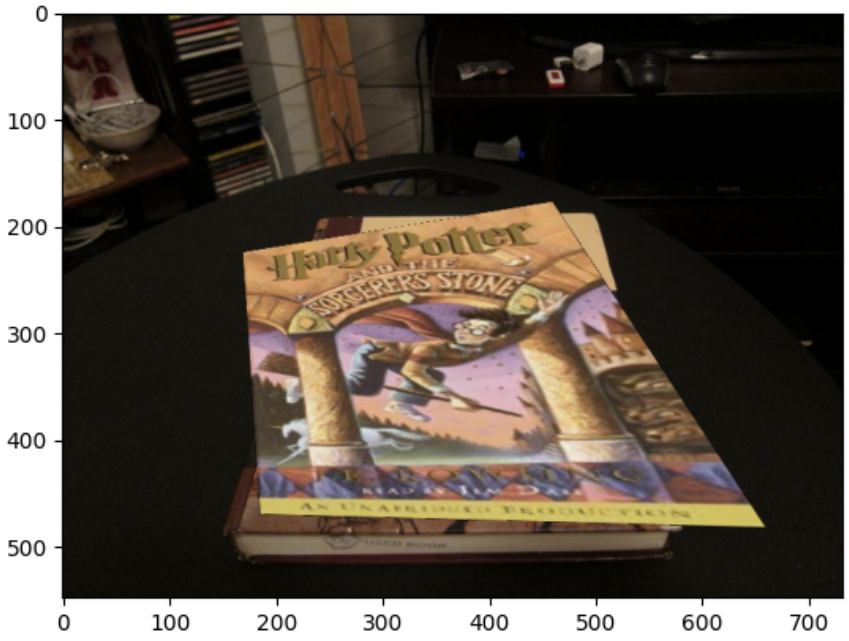

```
def warpImage(opts):
    img_cv_covoer_path = '../data/cv_cover.jpg'
    img_cv_cover = Image.open(img_cv_covoer_path).convert("RGB")
    img_cv_cover = np.array(img_cv_cover).astype(np.float32) / 255 #normalize

    img_cv_desk_path = '../data/cv_desk.png'
    img_cv_desk = Image.open(img_cv_desk_path).convert("RGB")
    img_cv_desk = np.array(img_cv_desk).astype(np.float32)/ 255 #normalize

    img_hp_cover_path = '../data/hp_cover.jpg'
    img_hp_cover = Image.open(img_hp_cover_path).convert("RGB")
    img_hp_cover = np.array(img_hp_cover).astype(np.float32)/ 255 #normalize

    # cv_cover: 440x350x3, hp_cover: 295x200x3
    # resize hp_cover to be matched with cv_cover
    h_cv_cover, w_cv_cover = img_cv_cover.shape[0], img_cv_cover.shape[1]
    img_hp_cover_resized = cv2.resize(img_hp_cover, (w_cv_cover, h_cv_cover))
    # rotated_image = ndimage.rotate(cv_desk, 40, reshape=False)

    matches, locs1, locs2 = matchPics(img_cv_desk, img_cv_cover, opts)

    locs1_matched = locs1[matches[:, 0]]
    locs2_matched = locs2[matches[:, 1]]

    bestH2to1, inliers = computeH_ransac(locs1_matched, locs2_matched, opts)

    img_hp_warped = cv2.warpPerspective(img_hp_cover_resized.swapaxes(0, 1),
bestH2to1, (img_cv_desk.shape[0], img_cv_desk.shape[1])).swapaxes(0, 1)
    plt.imshow(img_hp_warped)
    plt.show()
```

```
img_composited = compositeH(bestH2to1, img_cv_desk, img_hp_cover_resized)
plt.imshow(img_composited)
plt.show()
```

Q2.2.5 (10 points): RANSAC Parameter Tuning



Above result: Max iteration = 10. Due to the small number of iterations, the ransac algorithm couldn't find the accurate best2to1 matrix.



Above result: inlier tolerance = 10. By increasing inlier_tor, the ransac algorithm couldn't find the accurate homography matrix since broad range of points were considered as inlier, leading to hinder the optimization.

3.1 Incorporating video (20 points)

d

```python
import numpy as np
import cv2
import sys
from PIL import Image
from opts import get_opts
from tqdm import tqdm
import time

#Import necessary functions
from matchPics import matchPics
from planarH import computeH_ransac, computeH, computeH_norm, compositeH # Import
necessary functions
from helper import loadVid

start = time.time()
#Write script for Q3.1
opts = get_opts()

img_cv_covoer_path = '../data/cv_cover.jpg'
img_cv_cover = Image.open(img_cv_covoer_path).convert("RGB")
img_cv_cover = np.array(img_cv_cover).astype(np.float32) / 255 #normalize

vd_book_path = "../data/book.mov"
vd_source_path = "../data/ar_source.mov"

vd_book = loadVid(vd_book_path)
vd_source = loadVid(vd_source_path)

ar_vd_list = []

# for i in tqdm(range(110, 130)):
for i in tqdm(range(len(vd_source))):

    frame_book = np.array(vd_book[i]).astype(np.float32) / 255
    frame_source = np.array(vd_source[i]).astype(np.float32) / 255 #normalize

    # cv_cover: 440x350x3, frame_source: 360x640x3 -> crop frame to get rid of
black background and have the same ratio as cv_cover
    # then resize the frame
    frame_source_crop = frame_source[44:314, 212:427, :]
    h_cv_cover, w_cv_cover = img_cv_cover.shape[0], img_cv_cover.shape[1]
    frame_source_resized = cv2.resize(frame_source_crop, (w_cv_cover, h_cv_cover))
    # rotated_image = ndimage.rotate(cv_desk, 40, reshape=False)

    matches, locs1, locs2 = matchPics(frame_book, img_cv_cover, opts)

    locs1_matched = locs1[matches[:, 0]]
    locs2_matched = locs2[matches[:, 1]]

    bestH2to1, inliers = computeH_ransac(locs1_matched, locs2_matched, opts)
```

```
    img_composited = compositeH(bestH2to1, frame_book, frame_source_resized)

    ar_vd_list.append(img_composited)

print("time :", time.time() - start, "s")

#Saving video
ar_vd = np.stack(ar_vd_list, axis = 0)

width = ar_vd.shape[2]
hieght = ar_vd.shape[1]
channel = ar_vd.shape[3]

fps = 25.5 #similar fps to source video

fourcc = cv2.VideoWriter_fourcc(*'MJPG') # FourCC is a 4-byte code used to specify
the video codec.

video = cv2.VideoWriter('ar_result_modi.avi', fourcc, float(fps), (width, hieght))

for frame_count in range(len(ar_vd)):
    img = (ar_vd[frame_count]*255).astype(np.uint8)
    video.write(img)

video.release()
```

3.2 Make Your AR Real Time (Extra Credit - 15 points)

Before modification, full video frames composition:

```
time : 3848.9757509231567 s
```

After modification:

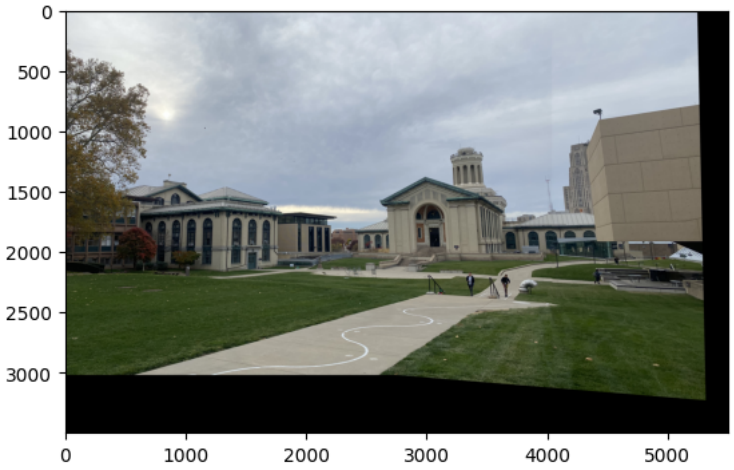
```
time : 2010.716341972351 s
```

I assumed the video frame changed slowly. Then, I computed the homography matrixes only for even order of the video first and performed compositing, then averaged each set of two matrixes to get the homography matrixes for odd order of the video frames.

```
opts = get_opts()

start = time.time()
# for i in tqdm(range(110, 130)):
img_cv_covoer_path = '../data/cv_cover.jpg'
img_cv_cover = Image.open(img_cv_covoer_path).convert("RGB")
img_cv_cover = np.array(img_cv_cover).astype(np.float32) / 255 #normalize

vd_book_path = "../data/book.mov"
vd_source_path = "../data/ar_source.mov"

vd_book = loadVid(vd_book_path)
vd_source = loadVid(vd_source_path)

H_list = []
ar_vd_even_list = []
length_vid = len(vd_source) - len(vd_source)%2 #Simply cut the last frame for even
number of frames

# for i in tqdm(range(len(vd_source))):
for i in tqdm(range(int(length_vid/2))):
    frame_book = np.array(vd_book[2*i]).astype(np.float32) / 255
    frame_source = np.array(vd_source[2*i]).astype(np.float32) / 255 #normalize

    # cv_cover: 440x350x3, frame_source: 360x640x3 -> crop frame to get rid of
black background and have the same ratio as cv_cover
    # then resize the frame
    frame_source_crop = frame_source[44:314, 212:427, :]
    h_cv_cover, w_cv_cover = img_cv_cover.shape[0], img_cv_cover.shape[1]
    frame_source_resized = cv2.resize(frame_source_crop, (w_cv_cover, h_cv_cover))
    # rotated_image = ndimage.rotate(cv_desk, 40, reshape=False)

    matches, locs1, locs2 = matchPics(frame_book, img_cv_cover, opts)

    locs1_matched = locs1[matches[:, 0]]
```

```
    locs2_matched = locs2[matches[:, 1]]

    bestH2to1, inliers = computeH_ransac(locs1_matched, locs2_matched, opts)
    H_list.append(bestH2to1)

    img_composited = compositeH(bestH2to1, frame_book, frame_source_resized)

    ar_vd_even_list.append(img_composited)

    ar_vd_odd_list = []

for i in range(len(H_list)-1):
    frame_book = np.array(vd_book[2*i+1]).astype(np.float32) / 255
    frame_source = np.array(vd_source[2*i+1]).astype(np.float32) / 255 #normalize

    # cv_cover: 440x350x3, frame_source: 360x640x3 -> crop frame to get rid of
black background and have the same ratio as cv_cover
    # then resize the frame
    frame_source_crop = frame_source[44:314, 212:427, :]
    h_cv_cover, w_cv_cover = img_cv_cover.shape[0], img_cv_cover.shape[1]
    frame_source_resized = cv2.resize(frame_source_crop, (w_cv_cover, h_cv_cover))

    bestH2to1 = (H_list[i] + H_list[i+1])/2

    img_composited = compositeH(bestH2to1, frame_book, frame_source_resized)

    ar_vd_odd_list.append(img_composited)

ar_vd_list = []
for i in range(length_vid-1):
    if i%2 == 0:
        ar_vd_list.append(ar_vd_even_list[int(i/2)])
    else:
        ar_vd_list.append(ar_vd_odd_list[int((i-1)/2)])
```

4 Create a Simple Panorama (10 points)




```
# Q4
```

```
opts = get_opts()

img_pano_left = Image.open('../data/my_pano_left.jpeg').convert("RGB")
img_pano_left = np.array(img_pano_left).astype(np.float32) / 255 #normalize

img_pano_right = Image.open('../data/my_pano_right.jpeg').convert("RGB")
img_pano_right = np.array(img_pano_right).astype(np.float32) / 255 #normalize

matches, locs1, locs2 = matchPics(img_pano_left, img_pano_right, opts)

locs1_matched = locs1[matches[:, 0]]
locs2_matched = locs2[matches[:, 1]]

bestH2to1, inliers = computeH_ransac(locs1_matched, locs2_matched, opts)

width = img_pano_left.shape[1] + img_pano_right.shape[1]
height = img_pano_left.shape[0] + img_pano_right.shape[0]

img_warped = cv2.warpPerspective(img_pano_right.swapaxes(0, 1), bestH2to1, (height,
width)).swapaxes(0, 1)
img_warped[0:img_pano_left.shape[0], 0:img_pano_left.shape[1]] = img_pano_left

plt.imshow(img_warped[:1250,:2000])
plt.show()
```