Homework 3: Augmented Reality with Planar Homographies

For each question please refer to the handout for more details.

Programming questions begin at **Q2**. **Remember to run all cells** and save the notebook to your local machine as a pdf for gradescope submission.

Collaborators

List your collaborators for all questions here:

Q1 Preliminaries

Q1.1 The Direct Linear Transform

Q1.1.1 (3 points)

How many degrees of freedom does **h** have?

h has 8 degrees of freedom. Although h is a 3x3 matrix, h is defined up to a scale factor. This makes it possible for the 9th value to have a value of 1, giving exactly 8 degrees of freedom.

Q1.1.2 (2 points)

How many point pairs are required to solve **h**?

To solve h, we would require 4 point pairs. Each point pair would give us 2 equations. With 4 points, we would have 8 equations for our 8 unknowns in A, making it sufficient to solve h.

Q1.1.3 (5 points)

Derive $\mathbf{A}_i$

Q1.1.4 (5 points)

What will be the trivial solution for **h**? Is the matrix **A** full rank? Why/Why not? What impact will it have on the singular values (i.e. eigenvalues of $\mathbf{A}^T\mathbf{A}$)?

The trivial solution for Ah=0 would be a matrix h of all zeros. This would essentially represent no transformation. It isn't full rank because it represents the correspondence between points in two different planes, leading to linearly dependent rows. Some singular values will be zero, indicating solutions with minimal effect.

Q1.2 Homography Theory Questions

Q1.2.1 (5 points)

Prove that there exists a homography **H** that satisfies $\mathbf{x}_1 \equiv \mathbf{H}\mathbf{x}_2$, given two cameras separated by a pure rotation.

YOUR ANSWER HERE...

Q1.2.2 (5 points):

Show that $\mathbf{H}^2$ is the homography corresponding to a rotation of 2θ.

This is equivalent to H.H

H^2 = (K R(theta) K^{-1})^2

```
= K R(theta) K^{-1} K R(theta) K^{-1}
```

```
= K R(theta) R(theta) K^{-1}
```

```
= K R(2theta) K^{-1}
```

This is because upon multiplication, the terms correspond to sin(2*theta) and cos(*2theta)

[[cos^2(theta) - sin^2(theta), -2sin(theta)cos(theta)], [2sin(theta)cos(theta), -sin^2(theta)+cos^2(theta)]]

which is equivalent to:

[[cos(2theta), sin(2theta)], [sin(2theta), cos(2theta)]]

= R (2theta)

Initialization

Run the following code to import the modules you'll need.

```
import os
import numpy as np
import cv2
import skimage.color
import pickle
from matplotlib import pyplot as plt
import scipy
from skimage.util import montage
import time
from skimage.io import imread

PATCHWIDTH = 9

def read_pickle(path):
    with open(path, "rb") as f:
        return pickle.load(f)

def write_pickle(path, data):
    with open(path, "wb") as f:
        pickle.dump(data, f)

def briefMatch(desc1,desc2,ratio):

    matches = skimage.feature.match_descriptors(desc1,desc2,
                                                'hamming',
                                                cross_check=True,
                                                max_ratio=ratio)
    return matches

def plotMatches(img1,img2,matches,locs1,locs2):

    fig, ax = plt.subplots(nrows=1, ncols=1)
    """
    print("cvtColor")
    print(img1.shape)
    print(img2.shape)

    if len(img1.shape) > 2:
        I1_gray = skimage.color.rgb2gray(img1)
    else:
        I1_gray = img1

    if len(img2.shape) > 2:
        I2_gray = skimage.color.rgb2gray(img2)
    else:
        I2_gray = img2
    """

    img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
    img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
    plt.axis('off')
    skimage.feature.plot_matches(ax,img1,img2,locs1,locs2,
                                 matches,matches_color='r',only_matches=True)
    plt.show()
    return

def makeTestPattern(patchWidth, nbits):

    np.random.seed(0)
    compareX = patchWidth*patchWidth * np.random.random((nbits,1))
    compareX = np.floor(compareX).astype(int)
    np.random.seed(1)
    compareY = patchWidth*patchWidth * np.random.random((nbits,1))
    compareY = np.floor(compareY).astype(int)

    return (compareX, compareY)

def computePixel(img, idx1, idx2, width, center):

    halfWidth = width // 2
    col1 = idx1 % width - halfWidth
    row1 = idx1 // width - halfWidth
    col2 = idx2 % width - halfWidth
    row2 = idx2 // width - halfWidth
    return 1 if img[int(center[0]+row1)][int(center[1]+col1)] < img[int(center[0]+row2)][int(center[1]+col2)] else 0

def computeBrief(img, locs):

    patchWidth = 9
    nbits = 256
    compareX, compareY = makeTestPattern(patchWidth,nbits)
    m, n = img.shape

    halfWidth = patchWidth//2

    locs = np.array(list(filter(lambda x: halfWidth <= x[0] < m-halfWidth and halfWidth <= x[1] < n-halfWidth, locs)))
    desc = np.array([list(map(lambda x: computePixel(img, x[0], x[1], patchWidth, c), zip(compareX, compareY))) for c in locs])

    return desc, locs

def corner_detection(img, sigma):

    # fast method
    result_img = skimage.feature.corner_fast(img, n=PATCHWIDTH, threshold=sigma)
    locs = skimage.feature.corner_peaks(result_img, min_distance=1)
    return locs

def loadVid(path):

    # Create a VideoCapture object and read from input file
    # If the input is the camera, pass 0 instead of the video file name

    cap = cv2.VideoCapture(path)

    # get fps, width, and height
    fps = cap.get(cv2.CAP_PROP_FPS)
    width  = cap.get(cv2.CAP_PROP_FRAME_WIDTH)
    height = cap.get(cv2.CAP_PROP_FRAME_HEIGHT)

    # Append frames to list
    frames = []

    # Check if camera opened successfully
    if cap.isOpened()== False:
        print("Error opening video stream or file")
```

```
    # Read until video is completed
    while(cap.isOpened()):

        # Capture frame-by-frame
        ret, frame = cap.read()

        if ret:
            #Store the resulting frame
            frames.append(frame)
        else:
            break

    # When everything done, release the video capture object
    cap.release()
    frames = np.stack(frames)

    return frames, fps, width, height
```

Download data

Download the required data and setup the results directory. If running on colab, DATA_PARENT_DIR must be
DATA_PARENT_DIR = '/content/'

Otherwise, use the local directory of your choosing. Data will be downloaded to DATA_PARENT_DIR/hw3_data and a subdirectory DATA_PARENT_DIR/results will be created.

```
# Only change this if you are running locally
# Default on colab: DATA_PARENT_DIR = '/content/'

# Data will be downloaded to DATA_PARENT_DIR/hw3_data/
# A subdirectory DATA_PARENT_DIR/results will be created

DATA_PARENT_DIR = '/content/'

if not os.path.exists(DATA_PARENT_DIR):
  raise RuntimeError('DATA_PARENT_DIR does not exist: ', DATA_PARENT_DIR)

RES_DIR = os.path.join(DATA_PARENT_DIR, 'results')
if not os.path.exists(RES_DIR):
  os.mkdir(RES_DIR)
  print('made directory: ', RES_DIR)


#paths different files are saved to
# OPTIONAL:
# feel free to change if funning locally
ROT_MATCHES_PATH = os.path.join(RES_DIR, 'brief_rot_test.pkl')
ROT_INV_MATCHES_PATH = os.path.join(RES_DIR, 'ec_brief_rot_inv_test.pkl')
AR_VID_FRAMES_PATH = os.path.join(RES_DIR, 'q_3_1_frames.npy')
AR_VID_FRAMES_EC_PATH = os.path.join(RES_DIR, 'q_3_2_frames.npy')

HW3_SUBDIR = 'hw3_data'
DATA_DIR = os.path.join(DATA_PARENT_DIR, HW3_SUBDIR)
ZIP_PATH = DATA_DIR + '.zip'
if not os.path.exists(DATA_DIR):
  !wget 'https://www.andrew.cmu.edu/user/hfreeman/data/16720_spring/hw3_data.zip' -O $ZIP_PATH
  !unzip -qq $ZIP_PATH -d $DATA_PARENT_DIR
```

```
made directory:  /content/results
--2024-03-02 16:30:36--  https://www.andrew.cmu.edu/user/hfreeman/data/16720_spring/hw3_data.zip
Resolving www.andrew.cmu.edu (www.andrew.cmu.edu)... 128.2.42.53
Connecting to www.andrew.cmu.edu (www.andrew.cmu.edu)|128.2.42.53|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 36434294 (35M) [application/zip]
Saving to: '/content/hw3_data.zip'

/content/hw3_data.z 100%[===================>]  34.75M  3.21MB/s    in 11s

2024-03-02 16:30:48 (3.06 MB/s) - '/content/hw3_data.zip' saved [36434294/36434294]
```

Q2 Computing Planar Homographies

Q2.1 Feature Detection and Matching

Q2.1.1 (5 points):

How is the FAST detector different from the Harris corner detector that you've seen in the lectures? Can you comment on its computation performance compared to the Harris corner detector?

The FAST detector identifies corners by checking the pixel intensities around a neighbourhood around the particular pixel. If a pixel has a significant number of pixels in its neighbourhood that are either darker or lighter than that pixel, then it is defined as a corner. To make the FAST detector even quicker, a Decision Tree is often used to partition pixels into either of three sets: Darker than Center, Lighter than Center, and Similar to the Center Pixel. The process is iteratively called until there is no more information gain.

The Harris corner detector, on the other hand, identifies corners by analyzing the changes in intensity that occur when a small window around a pixel is shifted in different directions. It computes a corner response function based on the gradients of the image and determines corners where this response is maximized. For this reason, the Harris corner detection is more robust to noise.

The computation performance of the FAST detector is significantly faster than the Harris Corner Detector. This is because the FAST detector only performs comparison of Pixel Intensities, whereas the Harris Corner Detector peforms gradient computations. Furthermore, the FAST detector involves around 9 or 12 operations per pixel, this can be further reduced based on some stopping criteria that can effectively say a pixel is not a corner. The FAST detector uses a Decision Tree model that can efficiently partition pixels into corner.

Q2.1.2 (5 points):

How is the BRIEF descriptor different from the filterbanks you've seen in the lectures? Could you use any one of the those filter banks as a descriptor?

The BRIEF descriptor is different from the filterbanks we'd seen earlier in the sense that BRIEF computes the difference in pixel intensities for a random set of pixels based on some sampling strategy. BRIEF is a binary descriptor because it only checks whether the pixel is darker or brighter than the other pixel. In filterbanks, multiple filters are applied on the same image to get different information regarding the image. Filterbanks are more often used as feature extractors rather than feature descriptors.

However, aggregating filterbanks together could result in a descriptor of the image.

Q2.1.3 (5 points):

Describe how the Hamming distance and Nearest Neighbor can be used to match interest points with BRIEF descriptors. What benefits does the Hamming distance have over a more conventional Euclidean distance measure in our setting?

The output of BRIEF is a binary string of 1s and 0s. To help match interest points with BRIEF descriptors, the Euclidean distance may not be suitable. This is because the Euclidean distance operates in a continuos space. This would mean that each BRIEF descriptor would have to be converted into such a space, and then compare the distances. The Hamming distance, however, operates in a binary discrete space, making it more suitable for comparing BRIEF decriptors between two interest points. Using Hamming Distance as the distance function, the Nearest Neighbour for an interest point can be computed based on this distance, effictively matching interest points between two images.

Q2.1.4 (10 points):

Implement the function matchPics()

```
def matchPics(I1, I2, ratio, sigma):
    """
    Match features across images

    Input
    -----
    I1, I2: Source images (RGB or Grayscale uint8)
    ratio: ratio for BRIEF feature descriptor
    sigma: threshold for corner detection using FAST feature detector

    Returns
    -------
    matches: List of indices of matched features across I1, I2 [p x 2]
    locs1, locs2: Pixel coordinates of matches [N x 2]
    """

    # ===== your code here! =====

    # TODO: Convert images to GrayScale
    # Input images can be either RGB or Grayscale uint8 (0 -> 255). Both need
    # to be supported.
    # Input images must be converted to normalized Grayscale (0.0 -> 1.0)
    # skimage.color.rgb2gray may be useful if the input is RGB.
    # TODO: Detect features in both images
    # TODO: Obtain descriptors for the computed feature locations
    # TODO: Match features using the descriptors
    # Convert images to grayscale if they are RGB
    print("Image1 Shape: "+str(I1.shape))
    print("Image2 Shape: "+str(I2.shape))

    if len(I1.shape) > 2:
        I1_gray = skimage.color.rgb2gray(I1)
    else:
        I1_gray = I1

    if len(I2.shape) > 2:
        I2_gray = skimage.color.rgb2gray(I2)
    else:
        I2_gray = I2

    print("Image1 Shape: "+str(I1_gray.shape))
    print("Image2 Shape: "+str(I2_gray.shape))

    # Detect features in both images
    locs1 = corner_detection(I1_gray, sigma)
    locs2 = corner_detection(I2_gray, sigma)

    # Compute descriptors for the detected feature locations using BRIEF
    desc1, locs1 = computeBrief(I1_gray, locs1)
    desc2, locs2 = computeBrief(I2_gray, locs2)

    # Match features using the computed descriptors
    matches = briefMatch(desc1, desc2, ratio)
    # ==== end of code ====

    return matches, locs1, locs2
```

Implement the function displayMatched

```
def displayMatched(I1, I2, ratio, sigma):
    """
    Displays matches between two images

    Input
    -----
    I1, I2: Source images
    ratio: ratio for BRIEF feature descriptor
    sigma: threshold for corner detection using FAST feature detector
    """

    print('Displaying matches for ratio: ', ratio, ' and sigma: ', sigma)

    # ===== your code here! =====
    # TODO: Use matchPics and plotMatches to visualize your results

    # Find matches between images
    matches, locs1, locs2 = matchPics(I1, I2, ratio, sigma)
    print("Matches: "+str(len(matches)))
    print("locs1: "+str(len(locs1)))
    print("locs2: "+str(len(locs2)))

    # Visualize matches
    plotMatches(I1, I2, matches, locs1, locs2)

    # ==== end of code ====

```

Visualize the matches

Use the cell below to visualize the matches. The resulting figure should look similar (but not necessarily identical) to Figure 2.

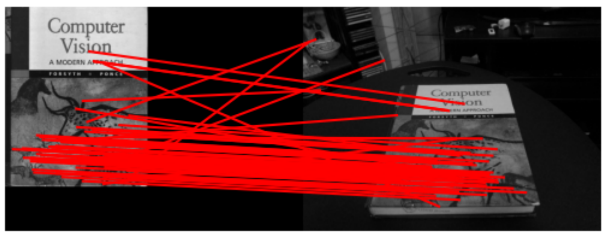
Feel free to play around with the images and parameters. Please use the original images when submitting the report.

Figure 2 parameters:

- image1_name = "cv_cover.jpg"
- image1_name = "cv_desk.png"
- ratio = 0.7
- sigma = 0.15

```
# Feel free to play around with these parameters
# BUT when submitting the report use the original images
image1_name = "cv_cover.jpg"
image2_name = "cv_desk.png"
ratio = 0.7
sigma = 0.1

image1_path = os.path.join(DATA_DIR, image1_name)
image2_path = os.path.join(DATA_DIR, image2_name)

image1 = cv2.imread(image1_path)
image2 = cv2.imread(image2_path)

#bgr to rgb
if len(image1.shape) == 3 and image1.shape[2] == 3:
  image1 = cv2.cvtColor(image1, cv2.COLOR_BGR2RGB)

if len(image2.shape) == 3 and image2.shape[2] == 3:
  image2 = cv2.cvtColor(image2, cv2.COLOR_BGR2RGB)

displayMatched(image1, image2, ratio, sigma)
```

```
Displaying matches for ratio:  0.7  and sigma:  0.1
Image1 Shape: (440, 350, 3)
Image2 Shape: (548, 731, 3)
Image1 Shape: (440, 350)
Image2 Shape: (548, 731)
<ipython-input-10-2fa2590324ba>:75: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from your array before performin
  return 1 if img[int(center[0]+row1)][int(center[1]+col1)] < img[int(center[0]+row2)][int(center[1]+col2)] else 0
Matches: 64
locs1: 1813
locs2: 862
```



Q2.1.5 (10 points):

Experiment with different sigma and ratio values. Conduct a small ablation study, and include the figures displaying the matched features with various parameters in your write-up. Explain the effect of these two paremeters respectively.

Here are the results of the ablation study.

1) Varying the Ratio value

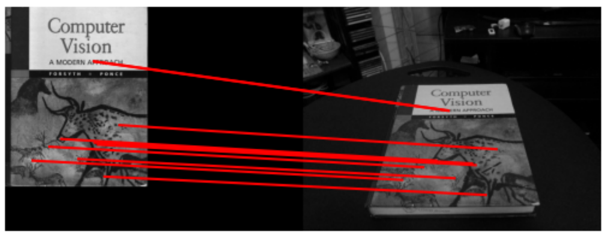
We see that by varying the Ratio value from 0.6, 0.7, 0.8, all the way to 0.9, the number of matched interest points increases. This is because the ratio parameter here specifies the maximum ratio between the first best match and second best match between matches of an interest point to two different interest points. By increasing this ratio, we are effectively allowing unstable matches.

2) Varying the Sigma value

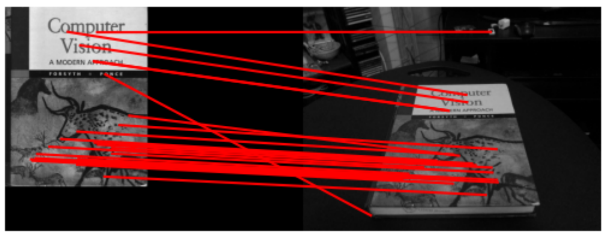
We see that by varying the Sigma value from 0.1, 0.15, 0.2, 0.3, all the way to 0.5, the threshold for pixel intensity comparison is altered. This means that as the sigma goes up, less points are marked as interest points as the neighbouring pixels would have to have a higher difference in intensities. This can be seen in the images below

```
image1_name = "cv_cover.jpg"
image2_name = "cv_desk.png"

image1_path = os.path.join(DATA_DIR, image1_name)
image2_path = os.path.join(DATA_DIR, image2_name)

image1 = cv2.imread(image1_path)
image2 = cv2.imread(image2_path)

#bgr to rgb
if len(image1.shape) == 3 and image1.shape[2] == 3:
  image1 = cv2.cvtColor(image1, cv2.COLOR_BGR2RGB)

if len(image2.shape) == 3 and image2.shape[2] == 3:
  image2 = cv2.cvtColor(image2, cv2.COLOR_BGR2RGB)

# ===== your code here! =====
ratio = [0.6, 0.7, 0.8, 0.9]
sigma = [0.1, 0.15, 0.2, 0.3, 0.5]

for r in ratio:
  displayMatched(image1, image2, r, 0.15)

for s in sigma:
  displayMatched(image1, image2, 0.7, s)

# ==== end of code ====
```

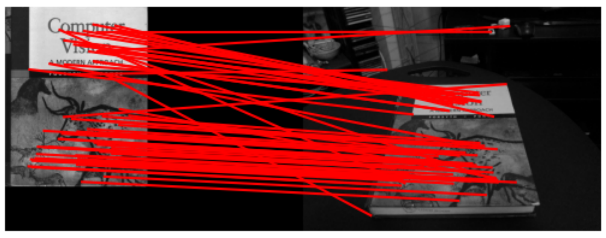
```
Displaying matches for ratio:  0.6  and sigma:  0.15
Image1 Shape: (440, 350, 3)
Image2 Shape: (548, 731, 3)
Image1 Shape: (440, 350)
Image2 Shape: (548, 731)
<ipython-input-10-2fa2590324ba>:75: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from your array before performin
  return 1 if img[int(center[0]+row1)][int(center[1]+col1)] < img[int(center[0]+row2)][int(center[1]+col2)] else 0
Matches: 10
locs1: 945
locs2: 476
```




```
Displaying matches for ratio:  0.7  and sigma:  0.15
Image1 Shape: (440, 350, 3)
Image2 Shape: (548, 731, 3)
Image1 Shape: (440, 350)
Image2 Shape: (548, 731)
Matches: 24
locs1: 945
locs2: 476
```

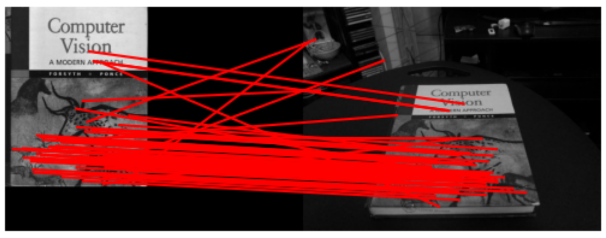


```
Displaying matches for ratio:  0.8  and sigma:  0.15
Image1 Shape: (440, 350, 3)
Image2 Shape: (548, 731, 3)
Image1 Shape: (440, 350)
Image2 Shape: (548, 731)
Matches: 61
locs1: 945
locs2: 476
```

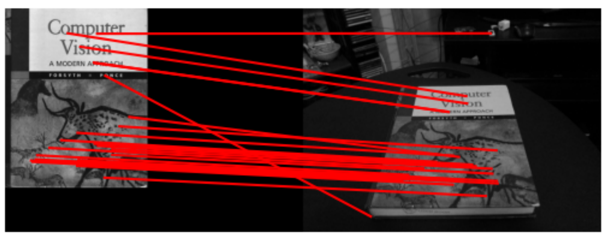


```
Displaying matches for ratio:  0.9  and sigma:  0.15
Image1 Shape: (440, 350, 3)
Image2 Shape: (548, 731, 3)
Image1 Shape: (440, 350)
Image2 Shape: (548, 731)
Matches: 125
locs1: 945
locs2: 476
```

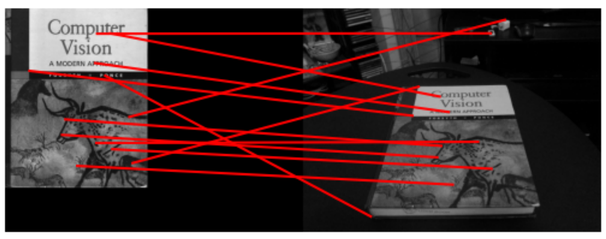
```
Displaying matches for ratio:  0.7  and sigma:  0.1
Image1 Shape: (440, 350, 3)
Image2 Shape: (548, 731, 3)
Image1 Shape: (440, 350)
Image2 Shape: (548, 731)
Matches: 64
locs1: 1813
locs2: 862
```



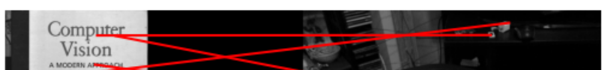
```
Displaying matches for ratio:  0.7  and sigma:  0.15
Image1 Shape: (440, 350, 3)
Image2 Shape: (548, 731, 3)
Image1 Shape: (440, 350)
Image2 Shape: (548, 731)
Matches: 24
locs1: 945
locs2: 476
```

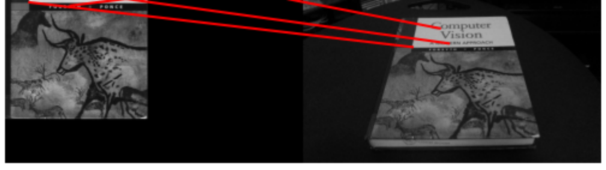


```
Displaying matches for ratio:  0.7  and sigma:  0.2
Image1 Shape: (440, 350, 3)
Image2 Shape: (548, 731, 3)
Image1 Shape: (440, 350)
Image2 Shape: (548, 731)
Matches: 12
locs1: 601
locs2: 309
```

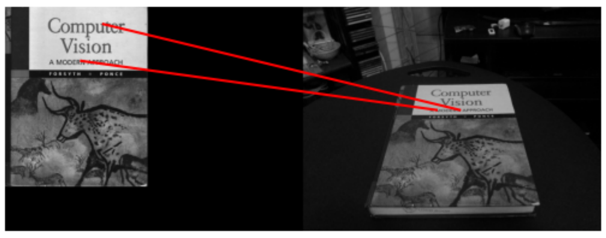



```
Displaying matches for ratio:  0.7  and sigma:  0.3
Image1 Shape: (440, 350, 3)
Image2 Shape: (548, 731, 3)
Image1 Shape: (440, 350)
Image2 Shape: (548, 731)
Matches: 5
locs1: 320
locs2: 136
```





```
Displaying matches for ratio:  0.7  and sigma:  0.5
Image1 Shape: (440, 350, 3)
Image2 Shape: (548, 731, 3)
Image1 Shape: (440, 350)
Image2 Shape: (548, 731)
Matches: 2
locs1: 154
locs2: 29
```



Q2.1.6 (10 points):

Implement the function briefRot

```python
def briefRot(min_deg, max_deg, deg_inc, ratio, sigma, filename):
    """
    Tests Brief with rotations.

    Input
    -----
    min_deg: minimum degree to rotate image
    max_deg: maximum degree to rotate image
    deg_inc: number of degrees to increment when iterating
    ratio: ratio for BRIEF feature descriptor
    sigma: threshold for corner detection using FAST feature detector
    filename: filename of image to rotate

    """

    if not os.path.exists(RES_DIR):
      raise RuntimeError('RES_DIR does not exist. did you run all cells?')

    # Read the image and convert bgr to rgb
    image_path = os.path.join(DATA_DIR, filename)
    image = cv2.imread(image_path)
    if len(image.shape) == 3 and image.shape[2] == 3:
      image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    match_degrees = [] # stores the degrees of rotation
    match_counts = [] # stores the number of matches at each degree of rotation

    for i in range(min_deg, max_deg, deg_inc):
        print(i)

        # ===== your code here! =====

        # Rotate Image
        rotated_image = scipy.ndimage.rotate(image, i, reshape=False)

        matches, locs1, locs2 = matchPics(image, rotated_image, ratio, sigma)

        match_degrees.append(i)
        match_counts.append(len(matches))
        # ==== end of code ====

    # Save to pickle file
    matches_to_save = [match_counts, match_degrees, deg_inc]
    write_pickle(ROT_MATCHES_PATH, matches_to_save)

def dispBriefRotHist(matches_path=ROT_MATCHES_PATH):
    # Check if pickle file exists
    if not os.path.exists(matches_path):
      raise RuntimeError('matches_path does not exist. did you call briefRot?')

    # Read from pickle file
    match_counts, match_degrees, deg_inc = read_pickle(matches_path)

    # Display histogram
    # Bins are centered and separated every 10 degrees
    plt.figure()
    bins = [x - deg_inc/2 for x in match_degrees]
    bins.append(bins[-1] + deg_inc)
    plt.hist(match_degrees, bins=bins, weights=match_counts, log=True)
    #plt.hist(match_degrees, bins=[10 * (x-0.5) for x in range(37)], weights=match_counts, log=True)
    plt.title("Histogram of BREIF matches")
    plt.ylabel("# of matches")
    plt.xlabel("Rotation (deg)")
    plt.tight_layout()

    output_path = os.path.join(RES_DIR, 'histogram.png')
    plt.savefig(output_path)
```

Visualize the matches under rotation

See debugging tips in handout.

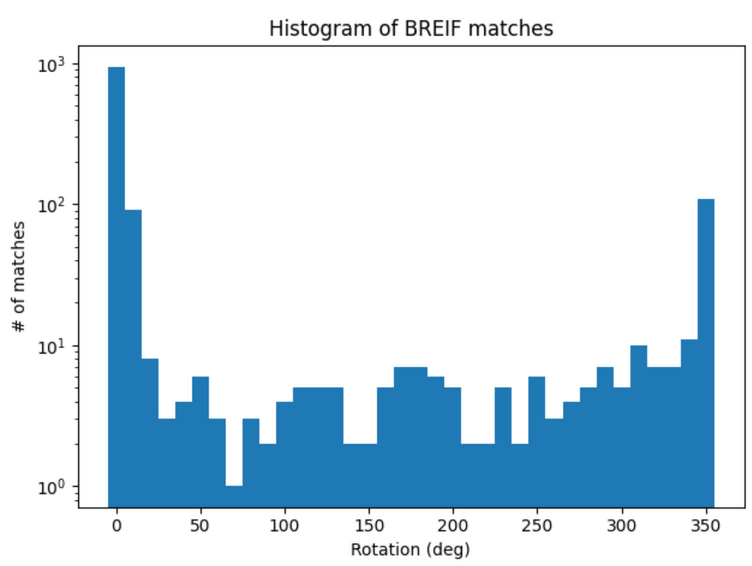
```
# defaults are:
# min_deg = 0
# max_deg = 360
# deg_inc = 10
# ratio = 0.7
# sigma = 0.15
# filename = 'cv_cover.jpg'

# Controls the rotation degrees
min_deg = 0
max_deg = 360
deg_inc = 10

# Brief feature descriptor and Fast feature detector paremeters
# (change these if you want to use different values)
ratio = 0.7
sigma = 0.15

# image to rotate and match
# (no need to change this but can if you want to experiment)
filename = 'cv_cover.jpg'

# Call briefRot
briefRot(min_deg, max_deg, deg_inc, ratio, sigma, filename)
```

```
Image1 Shape: (440, 350)
Image2 Shape: (440, 350)
20
Image1 Shape: (440, 350, 3)
Image2 Shape: (440, 350, 3)
Image1 Shape: (440, 350)
Image2 Shape: (440, 350)
30
Image1 Shape: (440, 350, 3)
Image2 Shape: (440, 350, 3)
Image1 Shape: (440, 350)
Image2 Shape: (440, 350)
40
Image1 Shape: (440, 350, 3)
Image2 Shape: (440, 350, 3)
Image1 Shape: (440, 350)
Image2 Shape: (440, 350)
50
Image1 Shape: (440, 350, 3)
Image2 Shape: (440, 350, 3)
Image1 Shape: (440, 350)
Image2 Shape: (440, 350)
60
Image1 Shape: (440, 350, 3)
Image2 Shape: (440, 350, 3)
Image1 Shape: (440, 350)
Image2 Shape: (440, 350)
70
Image1 Shape: (440, 350, 3)
Image2 Shape: (440, 350, 3)
Image1 Shape: (440, 350)
Image2 Shape: (440, 350)
80
Image1 Shape: (440, 350, 3)
Image2 Shape: (440, 350, 3)
Image1 Shape: (440, 350)
Image2 Shape: (440, 350)
90
Image1 Shape: (440, 350, 3)
Image2 Shape: (440, 350, 3)
Image1 Shape: (440, 350)
Image2 Shape: (440, 350)
100
Image1 Shape: (440, 350, 3)
Image2 Shape: (440, 350, 3)
Image1 Shape: (440, 350)
Image2 Shape: (440, 350)
110
Image1 Shape: (440, 350, 3)
Image2 Shape: (440, 350, 3)
Image1 Shape: (440, 350)
Image2 Shape: (440, 350)
120
Image1 Shape: (440, 350, 3)
Image2 Shape: (440, 350, 3)
Image1 Shape: (440, 350)
Image2 Shape: (440, 350)
130
```

Plot the histogram

See debugging tips in handout.

```
dispBriefRotHist()
```



Explain why you think the BRIEF descriptor behves this way: BRIEF is not rotation invariant. The binary descriptor for the matched points will change as the orientation of the image changes. This reduces the number of matched points. This can be seen in the histogram. For small rotations, like 0, 10, and 350, we see that there are still a large no. of matches. This is probably due to the fact that such small rotations would not change the final binary descriptor too much, which depends on a certain sampling strategy.

Q2.1.7.1 (Extra Credit - 5 points):

Design a fix to make BRIEF more rotation invariant. Feel free to make any helper functions as necessary. But you cannot use any additional OpenCV or Scikit-Image functions.

```python
# ===== your code here! =====
# TODO: Define any helper functions here
# (Feel free to put anything in its own cell)

# TODO: Feel free to modify the inputs and the function body as necessary
# This is only an outline
def briefRotInvEc(min_deg, max_deg, deg_inc, ratio, sigma, filename):
    """
    Rotation invariant Brief.

    Input
    -----
    min_deg: minimum degree to rotate image
    max_deg: maximum degree to rotate image
    deg_inc: number of degrees to increment when iterating
    ratio: ratio for BRIEF feature descriptor
    sigma: threshold for corner detection using FAST feature detector
    filename: filename of image to rotate

    """

    if not os.path.exists(RES_DIR):
      raise RuntimeError('RES_DIR does not exist. did you run all cells?')

    #Read the image and convert bgr to rgb
    image_path = os.path.join(DATA_DIR, filename)
    image = cv2.imread(image_path)
    if len(image.shape) == 3 and image.shape[2] == 3:
      image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    match_degrees = [] # stores the degrees of rotation
    match_counts = [] # stores the number of matches at each degree of rotation

    for i in range(min_deg, max_deg, deg_inc):
        print(i)

        # TODO: Rotate Image (Hint: use scipy.ndimage.rotate)

        # TODO: Brief matcher that is rotation invariant
        # Feel free to define additional helper functions as necessary

        # TODO: visualizes matches at at least 3 different orientations
        # to include in your report
        # (Hint: use plotMatches)

        # TODO: Update match_degrees and match_counts (see descriptions above)

    # Save to pickle file
    matches_to_save = [match_counts, match_degrees, deg_inc]
    write_pickle(ROT_INV_MATCHES_PATH, matches_to_save)

# ==== end of code ====
```

Visualize your implemented function

```python
min_deg = 0
max_deg = 360
deg_inc = 10
filename = 'cv_cover.jpg'

# ===== your code here! =====
# TODO: Call briefRotInvEc and visualize

# ==== end of code ====

```

Plot Histogram

```python
dispBriefRotHist(matches_path=ROT_INV_MATCHES_PATH)
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-24-8890031f161e> in <cell line: 1>()
----> 1 dispBriefRotHist(matches_path=ROT_INV_MATCHES_PATH)

<ipython-input-19-b92561582089> in dispBriefRotHist(matches_path)
     47     # Check if pickle file exists
     48     if not os.path.exists(matches_path):
---> 49       raise RuntimeError('matches_path does not exist. did you call briefRot?')
     50 
     51     # Read from pickle file

RuntimeError: matches_path does not exist. did you call briefRot?
```

Compare the histograms with an without rotation invariance. Explain your rotation invariant design and how you selected any parameters that you used: YOUR ANSWER HERE...

Q2.1.7.2 (Extra Credit - 5 points):

Design a fix to make BRIEF more scale invariant. Feel free to make any helper functions as necessary. But you cannot use any additional OpenCV or Scikit-Image functions.

```python
# ===== your code here! =====
# TODO: Define any helper functions here
# (Feel free to put anything in its own cell)

# TODO: Modify the inputs and the function body as necessary
def briefScaleInvEc(ratio, sigma, filename):

    #Read the image and convert bgr to rgb
    image_path = os.path.join(DATA_DIR, filename)
    image = cv2.imread(image_path)
    if len(image.shape) == 3 and image.shape[2] == 3:
      image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    match_scales = [] # stores the scaling factors
    match_counts = [] # stores the number of matches at each scaling factor

    for i in [1]:
        # Scale Image
        image_scale = cv2.resize(image,(int(image.shape[1]/(2**i)),
                                        int(image.shape[0]/(2**i))),
                                 interpolation = cv2.INTER_AREA)

        # TODO: Brief matcher that is scale invariant
        # Feel free to define additional helper functions as necessary

        # Compare to regular matchPics
        matches_orig, locs1_orig, locs2_orig = matchPics(image,
                                                         image_scale,
                                                         ratio, sigma)

        print('plotting non-scale invariant scale: ', 2**i)
        plotMatches(image, image_scale, matches_orig, locs1_orig,
                    locs2_orig)
        print('plotting scale-invariant: ', 2**i)
        plotMatches(image, image_scale, matches, locs1, locs2)

# ==== end of code ====
```

Visualize your implemented function

```python
# ===== your code here! =====
# TODO: Call briefScaleInvEc and visualize
# You may change any parameters and the function body as necessary

filename = 'cv_cover.jpg'

ratio = 0.7
sigma = 0.15

briefScaleInvEc(ratio, sigma, filename)
# ==== end of code ====
```

Explain your scale invariant design and how you selected any parameters that you used: YOUR ANSWER HERE...

Q2.2 Homography Computation

Q2.2.1 (15 Points): Compute H

Implement the function computeH

```python
def computeH(x1, x2):
    """
    Compute the homography between two sets of points

    Input
    -----
    x1, x2: Sets of points

    Returns
    -------
    H2to1: 3x3 homography matrix that best transforms x2 to x1
    """

    if x1.shape != x2.shape:
        raise RuntimeError('number of points do not match')

    # ===== your code here! =====
    # TODO: Compute the homography between two sets of points

    # Construct matrix A

    N = x1.shape[0]
    A = np.zeros((2 * N, 9))

    for i in range(N):
        A[2 * i] = [x2[i, 0], x2[i, 1], 1, 0, 0, 0, -x2[i, 0] * x1[i, 0], -x2[i, 1] * x1[i, 0], -x1[i, 0]]
        A[2 * i + 1] = [0, 0, 0, x2[i, 0], x2[i, 1], 1, -x2[i, 0] * x1[i, 1], -x2[i, 1] * x1[i, 1], -x1[i, 1]]

    # Perform SVD on A
    _, _, V = np.linalg.svd(A)

    # Extract homography matrix H2to1 from the last column of V
    H2to1 = V[-1].reshape(3, 3)

    return H2to1
```

Q2.2.2 (10 points): ComputeH_norm

Implement the function computeH_norm

```python
def computeH_norm(x1, x2):
    """
    Compute the homography between two sets of points using normalization

    Input
    -----
    x1, x2: Sets of points

    Returns
```

```
    -------
    H2to1: 3x3 homography matrix that best transforms x2 to x1
    """

    # ===== your code here! =====

    # TODO: Compute the centroid of the points
    centroid1 = np.mean(x1, axis=0)
    centroid2 = np.mean(x2, axis=0)

    # TODO: Shift the origin of the points to the centroid
    x1_centered = x1 - centroid1
    x2_centered = x2 - centroid2

    # TODO: Normalize the points so that the largest distance from the
    # origin is equal to sqrt(2)
    max_dist = np.sqrt(2)
    scale1 = max_dist / np.max(np.linalg.norm(x1_centered, axis=1))
    scale2 = max_dist / np.max(np.linalg.norm(x2_centered, axis=1))


    # TODO: Similarity transform 1
    T1 = np.array([[scale1, 0, -scale1 * centroid1[0]],
                   [0, scale1, -scale1 * centroid1[1]],
                   [0, 0, 1]])

    # TODO: Similarity transform 2
    T2 = np.array([[scale2, 0, -scale2 * centroid2[0]],
                   [0, scale2, -scale2 * centroid2[1]],
                   [0, 0, 1]])

    x1_centered = np.column_stack((x1_centered, np.ones(x1_centered.shape[0])))
    x2_centered = np.column_stack((x2_centered, np.ones(x2_centered.shape[0])))

    x1_norm = T1 @ x1_centered.T
    x2_norm = T2 @ x2_centered.T
    # TODO: Compute homography
    H2to1 = computeH(x1, x2)

    # TODO: Denormalization\
    #H2to1 = np.linalg.inv(T1) @ H2to1 @ T2
    H2to1 /= H2to1[2, 2]


    # ==== end of code ====

    return H2to1
```

Q2.2.3 (25 points): ComputeH_ransac

Implement RANSAC

```
def computeH_ransac(locs1, locs2, max_iters, inlier_tol):
    """
    Estimate the homography between two sets of points using ransac

    Input
    -----
    locs1, locs2: Lists of points
    max_iters: the number of iterations to run RANSAC for
    inlier_tol: the tolerance value for considering a point to be an inlier

    Returns
    -------
    bestH2to1: 3x3 homography matrix that best transforms locs2 to locs1
    inliers: indices of RANSAC inliers

    """

    # ===== your code here! =====

    # TODO:
    # Compute the best fitting homography using RANSAC
    # given a list of matching points locs1 and loc2
    bestH2to1 = None
    best_inliers = []

    flag=False
    print("Max Iterations: "+str(max_iters))
    for m in range(max_iters):

      locs1_swapped = np.column_stack((locs1[:, 1], locs1[:, 0]))
      locs2_swapped = np.column_stack((locs2[:, 1], locs2[:, 0]))

      rand_indices = np.random.choice(len(matches), 4, replace=False)
      rand_locs1 = [locs1_swapped[i] for i in rand_indices]
      rand_locs2 = [locs2_swapped[i] for i in rand_indices]

      rand_locs1=np.array(rand_locs1)
      rand_locs2=np.array(rand_locs2)

      H = computeH_norm(rand_locs1, rand_locs2)
      #print(H)
      #bestH2to1= H


      inliers = []
      for i in range(len(matches)):
          p1 = np.append(locs1_swapped[i], 1)
          p2 = np.append(locs2_swapped[i], 1)
          p2_transformed = H @ p2
          print(np.linalg.norm(p1 - p2_transformed))
          if np.linalg.norm(p1 - p2_transformed) < inlier_tol:
            print("inlier!!")
            inliers.append(i)

      if len(inliers) > len(best_inliers):
        flag=True
        print("good H found")
        best_inliers = inliers
        bestH2to1 = H

    if flag==False:
        bestH2to1 = H
```

```
    # ==== end of code ====

    return bestH2to1, best_inliers
```

Q2.2.4 (10 points): compositeH

Implement the function compositeH

```
def compositeH(H2to1, template, img):
    """
    Returns the composite image.

    Input
    -----
    H2to1: Homography from image to template
    template: template image to be warped
    img: background image

    Returns
    -------
    composite_img: Composite image

    """

    # ===== your code here! =====
    # TODO: Create a composite image after warping the template image on top
    # of the image using the homography

    h, w, _ = img.shape
    plt.imshow(template)
    warped_template = cv2.warpPerspective(template, H2to1, (w, h))
    composite_img = np.where(warped_template != 0, warped_template, img)

    # ==== end of code ====

    return warped_template
```

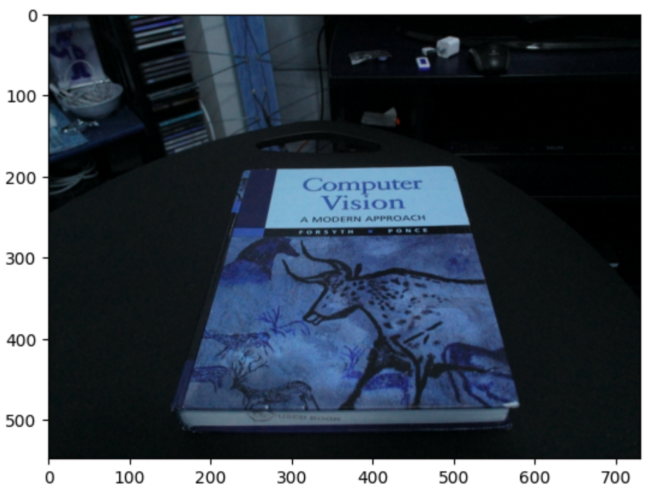
Implement the function warpImage

```
hp_cover = skimage.io.imread(os.path.join(DATA_DIR, 'hp_cover.jpg'))
cv_cover = skimage.io.imread(os.path.join(DATA_DIR, 'cv_cover.jpg'))
cv_desk = skimage.io.imread(os.path.join(DATA_DIR, 'cv_desk.png'))
cv_desk = cv_desk[:, :, :3]
```

```
image1 = cv_cover
image2 = cv_desk
image3= hp_cover

#bgr to rgb
if len(image1.shape) == 3 and image1.shape[2] == 3:
  image1 = cv2.cvtColor(image1, cv2.COLOR_BGR2RGB)

if len(image2.shape) == 3 and image2.shape[2] == 3:
  image2 = cv2.cvtColor(image2, cv2.COLOR_BGR2RGB)
```

```
plt.imshow(image2)
```

```
<matplotlib.image.AxesImage at 0x787fc42cdfc0>
```




```
matches, locs1, locs2 = matchPics(cv_cover, cv_desk, ratio, sigma)
```

```
Image1 Shape: (440, 350)
Image2 Shape: (548, 731, 3)
Image1 Shape: (440, 350)
Image2 Shape: (548, 731)
<ipython-input-10-2fa2590324ba>:75: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from your array before performin
  return 1 if img[int(center[0]+row1)][int(center[1]+col1)] < img[int(center[0]+row2)][int(center[1]+col2)] else 0
```

```
matched_locs1 = locs1[matches[:, 0]] #y,x

scaling_factor = [hp_cover.shape[0] / cv_cover.shape[0], hp_cover.shape[1] / cv_cover.shape[1]] #y,x
print(scaling_factor)
scaled_locs1 = np.round(matched_locs1 * scaling_factor).astype(int) #y,x

matched_locs2 =locs2[matches[:,1]]
```

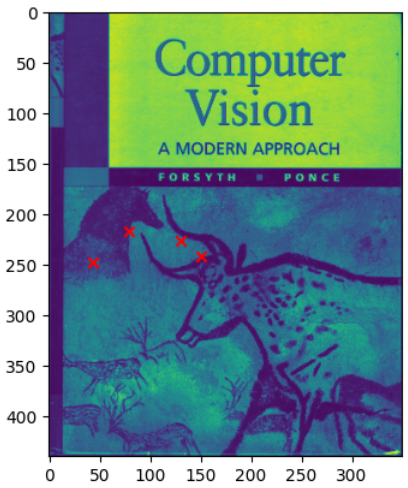
```
[0.6704545454545454, 0.5714285714285714]
```

```
locs1_swapped = np.column_stack((scaled_locs1[:, 1], scaled_locs1[:, 0]))
locs2_swapped = np.column_stack((matched_locs2[:, 1], matched_locs2[:, 0]))
```

```
rand_indices = np.random.choice(len(matches), 4, replace=False)

rand_locs1 = [locs1_swapped[i] for i in rand_indices]
rand_locs2 = [locs2_swapped[i] for i in rand_indices]
```

```
rand_locs1=np.array(rand_locs1)
rand_locs2=np.array(rand_locs2)
```

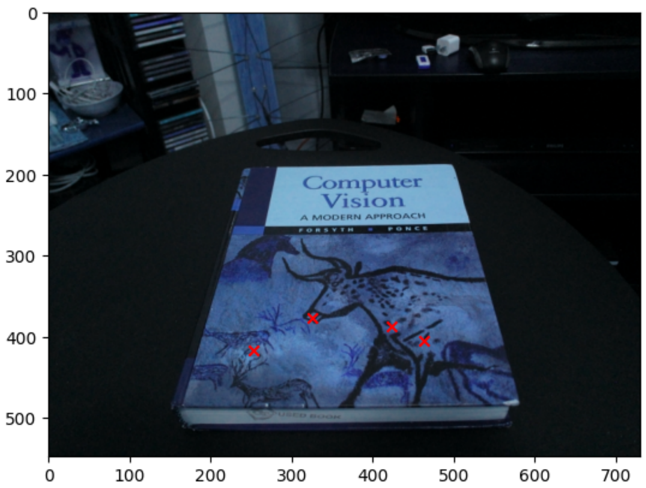
```
plt.imshow(image1)
plt.scatter(rand_locs1[:, 0], rand_locs1[:, 1], c='r', marker='x')
plt.show()
```




```
rand_locs2[:, 0]
```

```
array([253, 423, 326, 464])
```

```
plt.imshow(image2)
plt.scatter(rand_locs2[:, 0], rand_locs2[:, 1], c='r', marker='x')
plt.show()
```




```
x1
```

```
array([[ 43, 248],
       [130, 226],
       [ 79, 217],
       [150, 241]])
```

```
x1 = rand_locs1
x2 = rand_locs2

centroid1 = np.mean(x1, axis=0)
centroid2 = np.mean(x2, axis=0)
print(centroid1)
print(centroid2)
```

```
[100.5 233. ]
[366.5  396.25]
```

```
x1_centered = x1 - centroid1
x2_centered = x2 - centroid2
```

```
max_dist = np.sqrt(2)
scale1 = max_dist / np.max(np.linalg.norm(x1_centered, axis=1))
scale2 = max_dist / np.max(np.linalg.norm(x2_centered, axis=1))
```

```
# TODO: Similarity transform 1
T1 = np.array([[scale1, 0, -scale1 * centroid1[0]],
               [0, scale1, -scale1 * centroid1[1]],
               [0, 0, 1]])

# TODO: Similarity transform 2
T2 = np.array([[scale2, 0, -scale2 * centroid2[0]],
               [0, scale2, -scale2 * centroid2[1]],
               [0, 0, 1]])
```

```
x1_centered = np.column_stack((x1, np.ones(x1_centered.shape[0])))
x2_centered = np.column_stack((x2, np.ones(x2_centered.shape[0])))
```

```
x1_centered
```

```
array([[ 43., 248.,   1.],
       [130., 226.,   1.],
       [ 79., 217.,   1.],
       [150., 241.,   1.]])
```

```
x1_norm = T1 @ x1_centered.T
print(x1_norm)
```

```
[[-1.36841747  0.70205766 -0.51166914  1.17802895]
 [ 0.35697847 -0.16658995 -0.38077704  0.19038852]
 [ 1.          1.          1.          1.        ]]
```

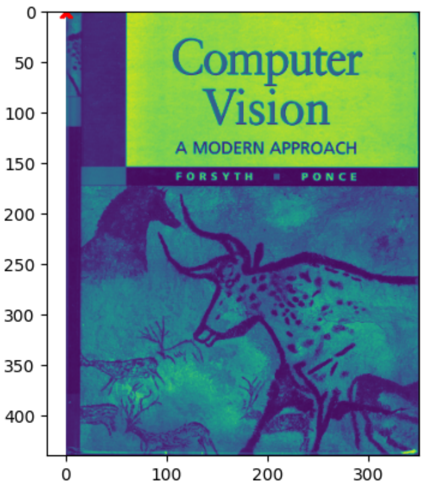
```
x2_centered
```

```
array([[253., 417.,   1.],
       [423., 387.,   1.],
       [326., 376.,   1.],
       [464., 405.,   1.]])
```

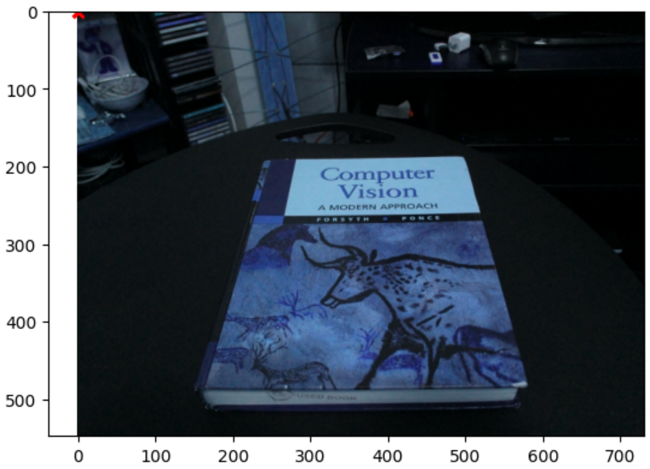
```
x2_norm = T2 @ x2_centered.T
print(x2_norm)
```

```
[[-1.39115638  0.69251397 -0.49640382  1.19504623]
 [ 0.25433035 -0.11337618 -0.24820191  0.10724774]
 [ 1.          1.          1.          1.        ]]
```

```python
plt.imshow(image1)
plt.scatter(x1_norm[:, 0], x1_norm[:, 1], c='r', marker='x')
plt.show()
```




```python
plt.imshow(image2)
plt.scatter(x2_norm[:, 0], x2_norm[:, 1], c='r', marker='x')
plt.show()
```




```python
H2to1 = computeH(x1, x2)
H2to1 /= H2to1[2, 2]
```

```python
H2to1
```

```
array([[ 9.44905656e-01,  1.66259041e-01, -2.13384591e+02],
       [-2.32861805e-01,  2.42027131e+00, -4.02394287e+02],
       [-1.03264707e-03,  3.52689352e-03,  1.00000000e+00]])
```

```python
H2to1 = np.linalg.inv(T1) @ H2to1 @ T2
H2to1 /= H2to1[2, 2]
```

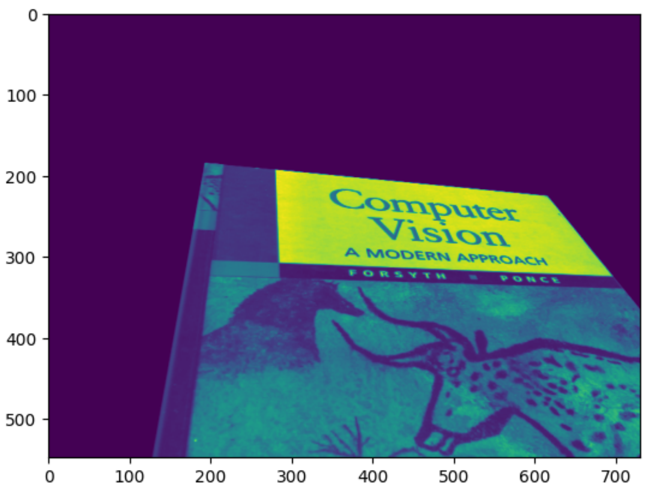
```python
H2to1
```

```
array([[ 3.19066558e+00, -7.75134174e-01,  1.58908963e+04],
       [ 5.44528878e-01,  5.86821347e-01,  1.51848292e+04],
       [ 1.70581448e-05, -3.56233972e-05,  1.00000000e+00]])
```

```python
height, width = image2.shape[:2]
warped_image2 = cv2.warpPerspective(image1, np.linalg.inv(H2to1), (width, height))
```

```python
plt.imshow(warped_image2)
```

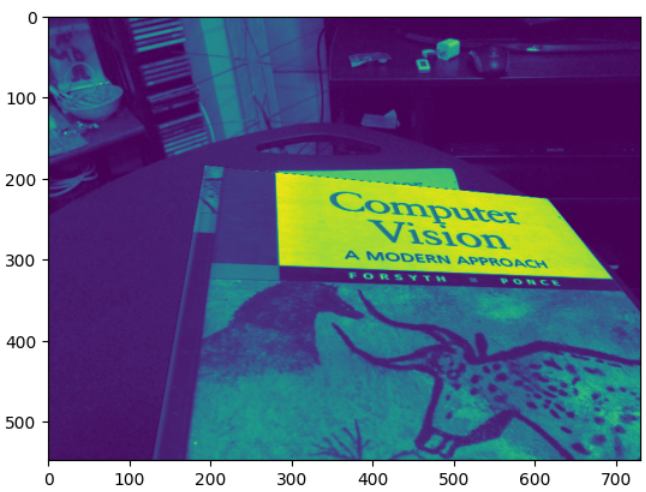
```
<matplotlib.image.AxesImage at 0x787fc3adb4f0>
```




```python
composite_img = np.where(warped_image2 != 0, warped_image2, image2.mean(2))
```

```python
plt.imshow(composite_img)
```

```
<matplotlib.image.AxesImage at 0x787fc39a9b40>
```



```
def warpImage(ratio, sigma, max_iters, inlier_tol):
    """
    Warps hp_cover.jpg onto the book cover in cv_desk.png.

    Input
    -----
    ratio: ratio for BRIEF feature descriptor
    sigma: threshold for corner detection using FAST feature detector
    max_iters: the number of iterations to run RANSAC for
    inlier_tol: the tolerance value for considering a point to be an inlier

    """

    hp_cover = skimage.io.imread(os.path.join(DATA_DIR, 'hp_cover.jpg'))
    cv_cover = skimage.io.imread(os.path.join(DATA_DIR, 'cv_cover.jpg'))
    cv_desk = skimage.io.imread(os.path.join(DATA_DIR, 'cv_desk.png'))
    cv_desk = cv_desk[:, :, :3]

    # ===== your code here! =====

    # TODO: match features between cv_desk and cv_cover using matchPics
    matches, locs1, locs2 = matchPics(cv_cover, cv_desk, ratio, sigma)

    # TODO: Scale matched pixels in cv_cover to size of hp_cover
    matched_locs1 = locs1[matches[:, 0]]
    scaling_factor = [hp_cover.shape[0] / cv_cover.shape[0], hp_cover.shape[1] / cv_cover.shape[1]]
    scaled_locs1 = np.round(matched_locs1 * scaling_factor).astype(int)
    matched_locs2 =locs2[matches[:,1]]

    # TODO: Get homography by RANSAC using computeH_ransac
    H, _ = computeH_ransac(matched_locs1, matched_locs2, max_iters, inlier_tol)
    print("Compute H ransac done")
    print(H.dtype)

    # TODO: Overlay using compositeH to return composite_img
    composite_img = compositeH(H, hp_cover, cv_desk)

    # ==== end of code ====

    plt.imshow(composite_img)
    plt.show()
```

Visualize composite image

```
# defaults are:
# ratio = 0.7
# sigma = 0.15
# max_iters = 600
# inlier_tol = 1.0

# (no need to change this but can if you want to experiment)
ratio = 0.7
sigma = 0.15
max_iters = 600
inlier_tol = 50.0

warpImage(ratio, sigma, max_iters, inlier_tol)
```

Q2.2.5 (10 points):

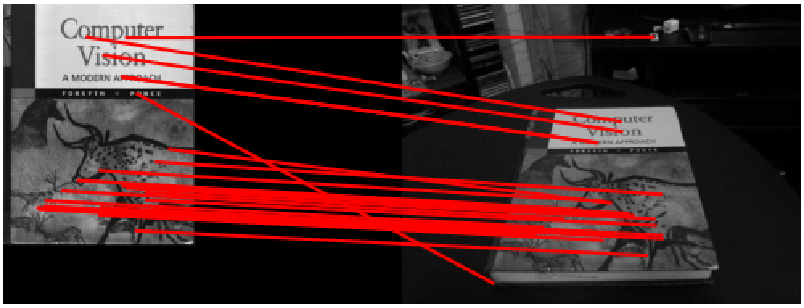
Conduct ablation study with various max_iters and inlier_tol values. Plot the result images and explain the effect of these two parameters respectively.

```
# ===== your code here! =====
# Experiment with different max_iters and inlier_tol values.
# Include the result images in the write-up.

# ==== end of code ====
```

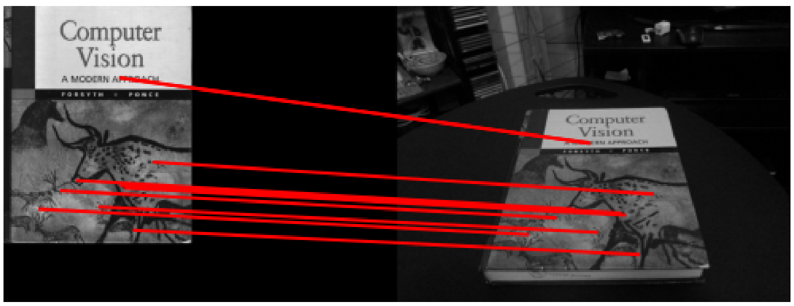
Explain the effect of max_iters and inlier_tol: YOUR ANSWER HERE...

Q3 Create a Simple Panorama

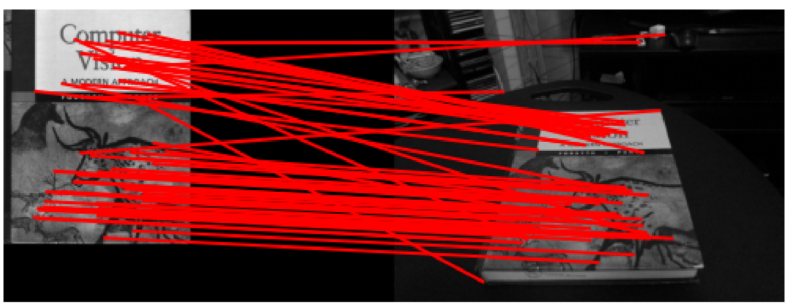
Original (Sigma=0.15, Ratio=0.7)



(Sigma=0.15, Ratio=0.6)

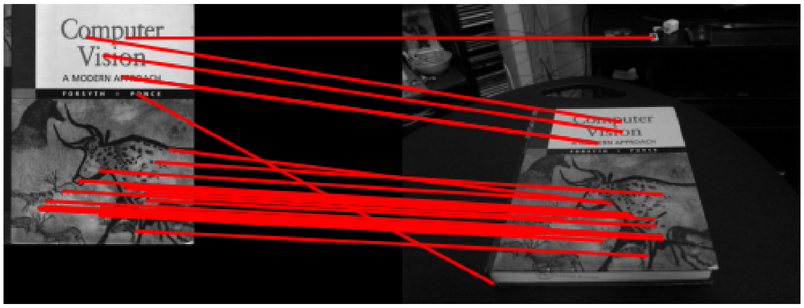


(Sigma=0.15, Ratio=0.8)

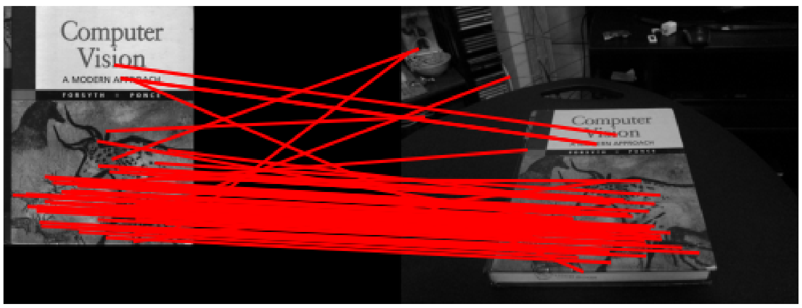


(Sigma=0.15, Ratio=0.9)

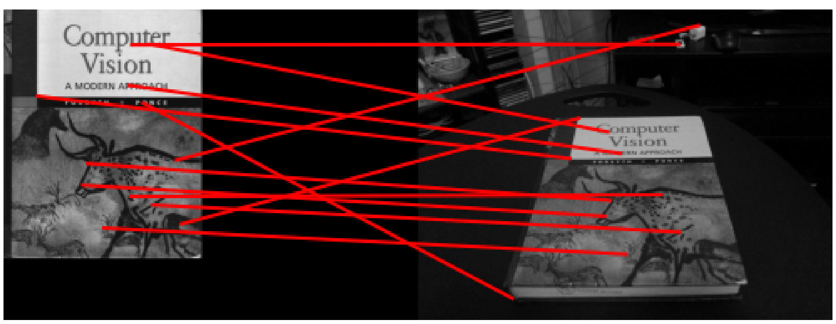
Original (Sigma=0.15, Ratio=0.7)



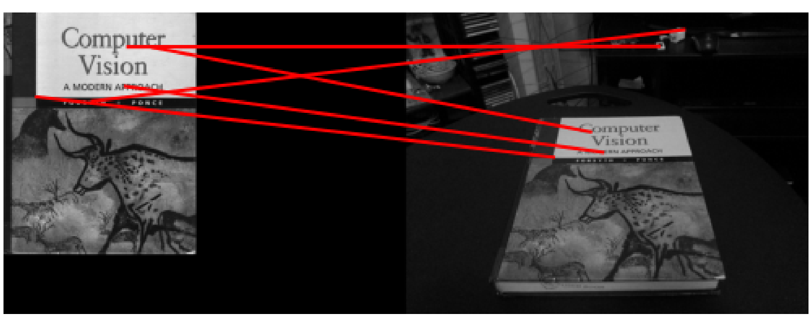
(Sigma=0.1, Ratio=0.7)



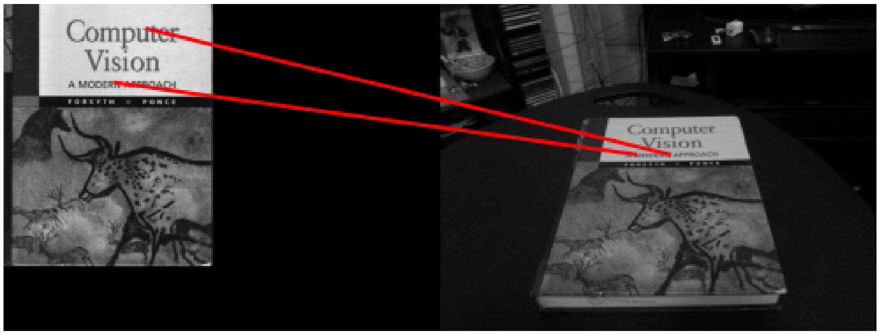
(Sigma=0.2, Ratio=0.7)



(Sigma=0.3, Ratio=0.7)



(Sigma=0.5, Ratio=0.7)



1.201) $$x_1 = K_1 [I\ 0] X$$

$$x_2 = K_2 [R\ 0] X$$

$$X = K_1^{-1} x_1.$$

So, $$x_2 = K_2 [R\ 0] K^{-1} x_1$$

$$x_2 = \underline{\underline{K_2 R K_1}^{-1} x_1}$$

$$\underline{\underline{x_2 \equiv H x_1}}$$

Let's define h as $\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$

Let's use the points (x_1^i, y_1^i) and (x_2^i, y_2^i) such that $x_2^i \equiv H x_1^i$ $(i \in \{1 \dots N\})$

To compute h,

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

$$x_2^i \overbrace{(g x_1^i + h y_1^i + i)}^{\lambda} = a x_1^i + b y_1^i + c$$

$$y_2^i (g x_1^i + h y_1^i + i) = d x_1^i + e y_1^i + f$$

These can be written as:

$$a x_1^i + b y_1^i + c - g x_1^i x_2^i - h x_2^i y_1^i - i x_2^i = 0$$

$$d x_1^i + e y_1^i + f - g x_1^i y_2^i - h y_1^i y_2^i - i y_2^i = 0$$

$$A_i = \begin{bmatrix} x_1^i & y_1^i & 1 & 0 & 0 & 0 & -x_1^i x_2^i & -x_2^i y_1 & -x_2^i \\ 0 & 0 & 0 & x_1^i & y_1^i & 1 & -x_1^i y_2^i & -y_1^i y_2^i & -y_2^i \end{bmatrix}$$