Homework 2: 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 it has 9 elements (3X3 matrix) it is agnostic of scaling.

Q1.1.2 (2 points)

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

---

4 points giving us 8 equations in total to solve for the 8 degrees of freedom of H

Q1.1.3 (5 points)

Derive A_i

We are working with two sets of corresponding points, $\mathbf{x}_1^{(i)}$ and $\mathbf{x}_2^{(i)}$, taken from two images captured by different cameras. The relation between these two sets of points is governed by a homography matrix H which is a 3×3 transformation matrix. The points $\mathbf{x}_1^{(i)}$ and $\mathbf{x}_2^{(i)}$ are in homogeneous coordinates.

Given that:

$$x_1^{(i)} \equiv H\, x_2^{(i)}$$

we aim to derive the matrix A_i such that:

$$A_i h = 0$$

holds for each point pair, where h is the vector (9×1) containing the elements of matrix H.

We can express the homogeneous coordinates and the transformation in matrix form as:

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

Let $H = \begin{pmatrix} h_1^\top \\ h_2^\top \\ h_3^\top \end{pmatrix}$. We can now break this equation into two scalar equations for $x_1^{(i)}$ and $y_1^{(i)}$ as:

$$x_1^{(i)} = \frac{h_1^\top x_2^{(i)}}{h_3^\top x_2^{(i)}}$$

$$y_1^{(i)} = \frac{h_2^\top x_2^{(i)}}{h_3^\top x_2^{(i)}}$$

Rearranging these equations gives:

$$x_1^{(i)} \cdot h_3^\top x_2^{(i)} - h_1^\top x_2^{(i)} = 0$$

$$y_1^{(i)} \cdot h_3^\top x_2^{(i)} - h_2^\top x_2^{(i)} = 0$$

These two equations represent constraints on the elements of H for each point pair. To linearize these constraints, we stack them into the matrix equation:

$$\begin{pmatrix} -x_2^{(i)} & 0 & x_i^{(1)} x_2^{(i)} \\ 0 & -x_2^{(i)} & y_i^{(1)} x_2^{(i)} \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ h_3 \end{pmatrix} = 0$$

Expanding this matrix gives:

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

Thus, the matrix $ A_i $ for the Direct Linear Transform (DLT) is:

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

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 $\mathrm{A}^T \mathrm{A}$)?

A is 8X9 matrix. It is not full rank since h is the solution in the null space of A since we find the eigenvector with zero(least) eigen value such that Ah = λ h = 0 . A can not be full rank but since it is given if A is full rank then null space is ϕ and non zero singular values is possible then only trivial solution is possible i.e. $h = 0_{3X3}$.

Q1.2 Homography Theory Questions

Q1.2.1 (5 points)

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

We want to prove that there exists a homography matrix H such that:

$$x_2 = H x_1$$

Let the homography matrix H be:

$$H = K_1 R^{\top} K_2^{-1}$$

We start by substituting the expression for x_1 and x_2:

$$x_1 = K_1 \begin{bmatrix} I & 0 \end{bmatrix} X$$

and

$$x_2 = K_2 \begin{bmatrix} R & 0 \end{bmatrix} X$$

Now, we multiply x_2 by the homography H:

$$H x_2 = K_1 R^\top K_2^{-1} x_2$$

Substitute the expression for x_2:

$$H x_2 = K_1 R^\top K_2^{-1} \left(K_2 \begin{bmatrix} R & 0 \end{bmatrix} X \right)$$

By simplifying the right-hand side, we get:

$$H x_2 = K_1 R^\top \begin{bmatrix} R & 0 \end{bmatrix} X = K_1 \begin{bmatrix} R^\top R & 0 \end{bmatrix} X = K_1 \begin{bmatrix} I & 0 \end{bmatrix} X = x_1$$

This matches the original equation for x_1, thus proving that:

$$x_1 = H x_2$$

Q1.2.2 (5 points):

Show that H^2 is the homography corresponding to a rotation of 2θ.

Since the intrinsic parameters are the same on one rotation by θ to get rotation R. $ x_2 = K_1 \begin{bmatrix} R & 0 \end{bmatrix} X $ and $x_1 = K_1 \begin{bmatrix} I & 0 \end{bmatrix} X $

From this the homography matrix is:

$$H = K_1 R K_1^{-1}$$

i.e $ x_2 = H x_1$

For again rotation by θ to reapply R to get a total rotation of 2 $ \theta $. we get

$$x_3 = H x_2 = H H x_1 = H^2 x_1$$

Hence the new homography matrix for 2 $ \theta $ rotation is $ H^2$.

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

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)
    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.z
ip' -O $ZIP_PATH
  !unzip -qq $ZIP_PATH -d $DATA_PARENT_DIR
```

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?

FAST (Features from Accelerated Segment Test) uses a circle of 16 pixels around a central pixel and classifies a corner if a certain number of surrounding pixels are significantly brighter or darker than the center. FAST optimizes speed by first checking four key points and skipping further checks if these points don’t indicate a corner, making it much faster than traditional methods like Harris, which rely on more computationally intensive operations.

The Harris Corner Detector computes image gradients to form a structure tensor, whose eigenvalues indicate whether a pixel is a corner, an edge, or a flat region. Instead of directly computing eigenvalues, it uses a corner response function based on the determinant and trace of the matrix. $ R(x,y) = \text{det}(M(x,y)) - k \cdot (\text{trace}(M(x,y)))^2$ where $M(x,y)$ is the matrix of second moment computed at a patch at a given pixel location (x,y). Pixels with strong corner responses are selected after thresholding and non-maximum suppression. Though computationally heavier than other methods, it is highly accurate for corner detection in computer vision tasks.

Since FAST has an early exit based on just 4 comparisons of intensity value, it is much faster compared to Harris.

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 generates descriptions at keypoints in an image by comparing pixel intensities within a patch around each keypoint. After detecting keypoints using algorithms like FAST or Harris, BRIEF performs random pixel pair intensity comparisons within the patch, generating a binary string (descriptor) based on the results. This process effectively encodes the template of the patch into a string of 1s and 0s. This descriptor is compact and can be matched quickly using Hamming distance.

Filter banks that we used describe the patch interms of generic functions that compute the rate change and direction of patterns in a given image primarily. If we considered each operation as a basis function, the set of image patches that would get a unique descriptor with sufficient seperation in the representation space is limited since we are working with a finite set of filter banks. In case of BRIEF each possible patch gets its template encoded in bindary and can be controlled by the number of random comparisons that we wish to perform.

Yes we could use the LoG DoGx and DoGy filters here since keypoints are corners, these would capture the intensity and direction of the edges of the corners involved. The Gaussian filter would allow us to distinguish based on color. Note that the filter size should be kept sufficiently large and the patches might need to be locally normalised for this to be applicable since these are images taken from 2 different views that we are working with.

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?

Given a descriptor for each keypoint location, we can find the nearest neighbour using a distance function. Generally the distance function should be the one corresponding to the space in which the decriptors lie. Since for a BRIEF descriptor lies in the space fo strings $\{0,1\}^N$ Hamming is a natural distance function that we can use. We could still use L1 euclidean distance but that would give us the same a hamming distance. For L_p distance function is just $\left(\mathrm{Hamm}\left(s_1, s_2\right)\right)^{1/p}$. Since there is a monotonic relationship between hamming and euclidean distance, any ranking or nearest neighbour comparison should not be affected but still hamming should be preferred since the distance function tells us how many bits we are off. Hamming distance is robust in discrete spaces and maintains performance despite small bit-level differences. This aligns with binary descriptors' tolerance to small variations, which can occur due to noise or slight changes in image alignment.

We can match a point p to its nearest neighbour in a set S as:

$$

p_{\text{nearest}} = \arg \min_{s\in S} \text{Hamm}(\text{BRIEF}(s), \text{BRIEF}(p)) $$

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
    # convert I1 and I2 to numpy arrays
    # Convert images to numpy arrays and uint8 format
    # Convert images to numpy arrays and uint8 format
    # Debugging - Check image shapes and types

    # Check if images are in RGB format, and convert to grayscale if
necessary
    if len(I1.shape) == 3 and I1.shape[2] == 3:
        I1 = skimage.color.rgb2gray(I1)
    if len(I2.shape) == 3 and I2.shape[2] == 3:
        I2 = skimage.color.rgb2gray(I2)

    # Normalize images to 0.0 - 1.0 range
    I1 = I1 / 255.0
    I2 = I2 / 255.0

    # rescale sigma since we are rescaling images
```

```
    sigma = sigma/255.0

    # Detect features in both images
    locs1 = corner_detection(I1, sigma)
    locs2 = corner_detection(I2, sigma)

    # Obtain descriptors for the computed feature locations
    desc1, locs1 = computeBrief(I1, locs1)
    desc2, locs2 = computeBrief(I2, locs2)

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

    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

    matches, locs1, locs2 = matchPics(I1, I2, ratio, sigma)
    plotMatches(I1, I2, matches, locs1, locs2)
    return
    # ==== 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:

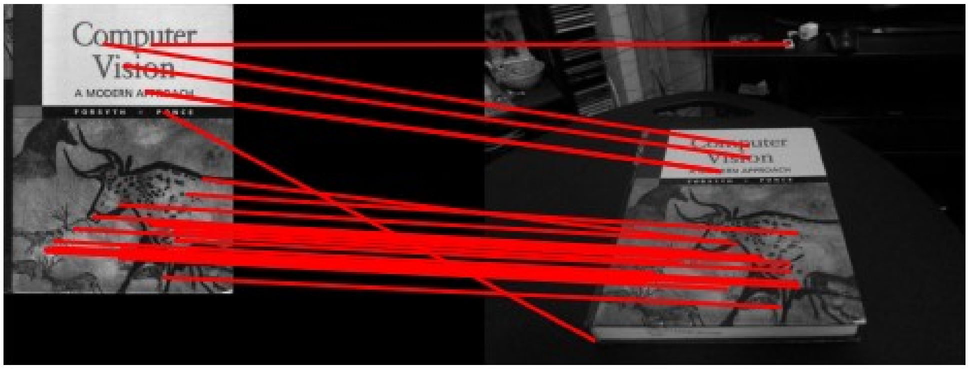
- image1_name = "cv_cover.jpg"
- image1_name = "cv_desk.png"

- 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 = 0.15

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:  0.15
```

```
/var/folders/43/5971cqb109bfv3wb3b47x9tc0000gn/T/
ipykernel_34345/3301853510.py:58: 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
performing this operation. (Deprecated NumPy 1.25.)
  return 1 if img[int(center[0]+row1)][int(center[1]+col1)] <
img[int(center[0]+row2)][int(center[1]+col2)] else 0
/var/folders/43/5971cqb109bfv3wb3b47x9tc0000gn/T/ipykernel_34345/33018
53510.py:35: FutureWarning: `plot_matches` is deprecated since version
0.23 and will be removed in version 0.25. Use
`skimage.feature.plot_matched_features` instead.
  skimage.feature.plot_matches(ax,img1,img2,locs1,locs2,
```



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.

---

1. **sigma(threshold)**: Threshold used in deciding whether the pixels on the circle are brighter, darker or similar w.r.t. the test pixel. Decrease the threshold when more corners are desired and vice-versa. A point c on the circle is darker w.r.t test pixel p if Ic < Ip - threshold and brighter if Ic > Ip + threshold. Also stands for the n in FAST-n corner detector. increasing sigma decreases the number of candidate corner points

2. **Ratio**: Maximum ratio of distances between first and second closest descriptor in the second set of descriptors. This threshold is useful to filter ambiguous matches between the two descriptor sets. If the ratio is small (e.g., 0.7), it means the nearest neighbor is significantly closer than the second-nearest, indicating a good match.If the ratio is larger, it means the nearest and second-nearest matches are very close in terms of distance, so it might be ambiguous and is typically rejected resulting in fewer matches

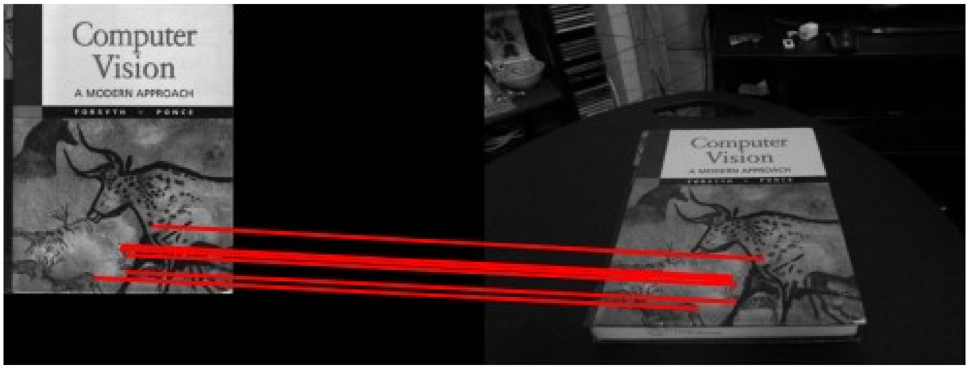
Ref: skimage documentation (https://scikit-image.org/)

```
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
```

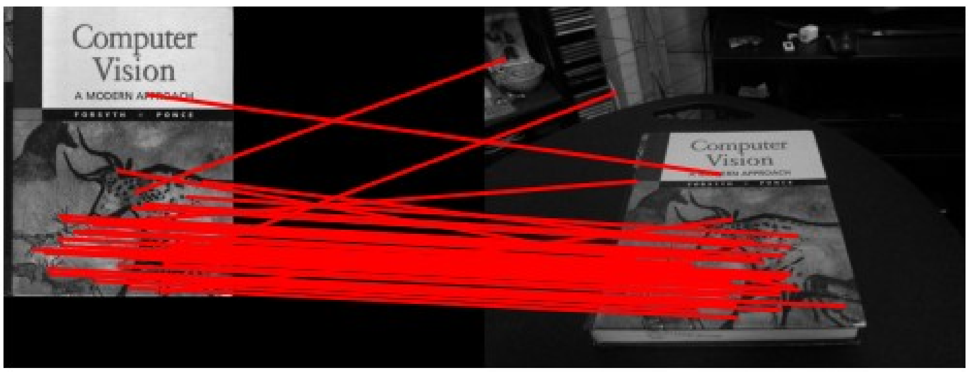
```python
if len(image1.shape) == 3 and image1.shape[2] == 3:
  image1 = cv2.cvtColor(image1, cv2.COLOR_BGR2RGB)

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

# ===== your code here! =====
# Experiment with different sigma and ratio values.
# Use displayMatches to visualize.
# Include the matched feature figures in the write-up.

# Display matches for different sigma and ratio values
for sigma in [0.1, 0.25, 0.5]:
    for ratio in [0.33, 0.5, 0.67, 0.9]:
        displayMatched(image1, image2, ratio, sigma)

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

```
Displaying matches for ratio:  0.33  and sigma:  0.1

/var/folders/43/5971cqb109bfv3wb3b47x9tc0000gn/T/
ipykernel_34345/3301853510.py:58: 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
performing this operation. (Deprecated NumPy 1.25.)
  return 1 if img[int(center[0]+row1)][int(center[1]+col1)] <
img[int(center[0]+row2)][int(center[1]+col2)] else 0
/var/folders/43/5971cqb109bfv3wb3b47x9tc0000gn/T/ipykernel_34345/33018
53510.py:35: FutureWarning: `plot_matches` is deprecated since version
0.23 and will be removed in version 0.25. Use
`skimage.feature.plot_matched_features` instead.
  skimage.feature.plot_matches(ax,img1,img2,locs1,locs2,
```

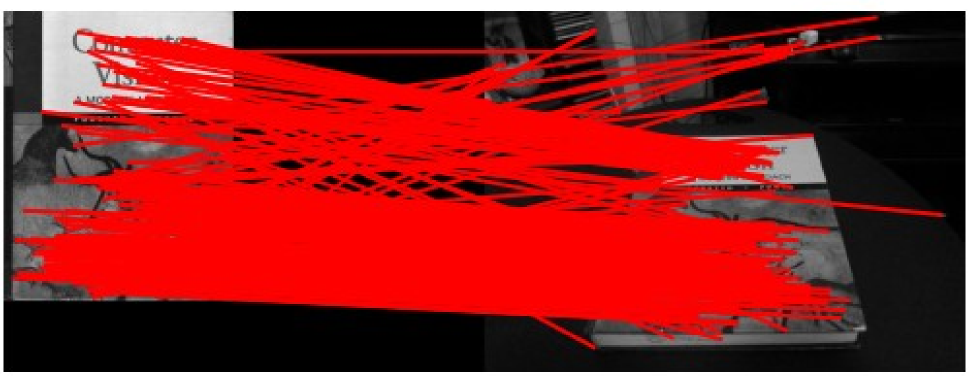

```
Displaying matches for ratio:  0.5  and sigma:  0.1
```




```
Displaying matches for ratio:  0.67  and sigma:  0.1
```



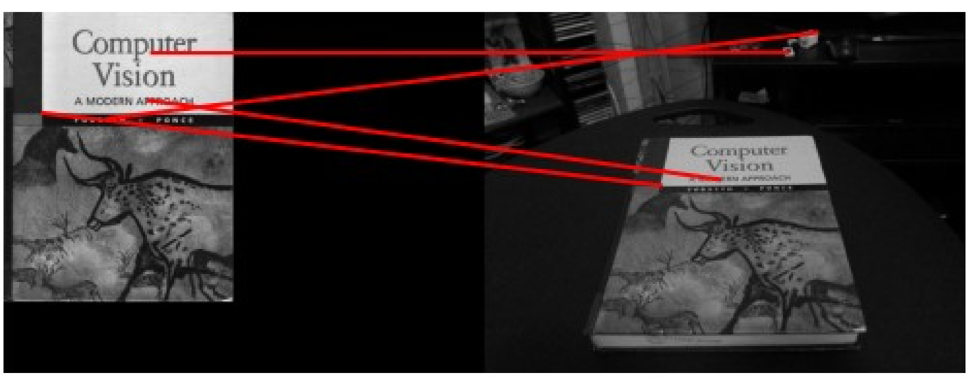

```
Displaying matches for ratio:  0.9  and sigma:  0.1
```



```
Displaying matches for ratio:  0.33  and sigma:  0.25
```


```
Displaying matches for ratio:  0.5  and sigma:  0.25
```


```
Displaying matches for ratio:  0.67  and sigma:  0.25
```

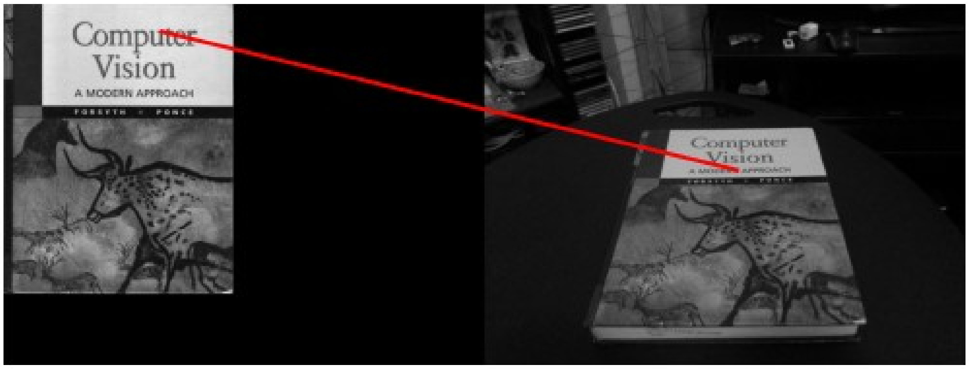



```
Displaying matches for ratio:  0.9  and sigma:  0.25
```

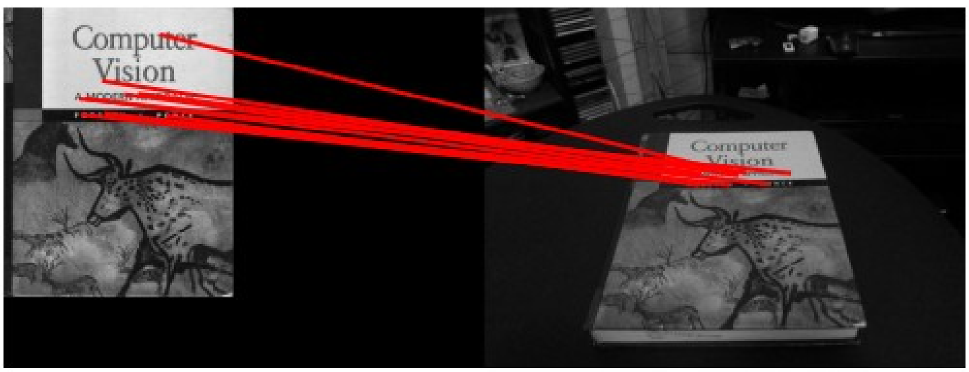
```
Displaying matches for ratio:  0.33  and sigma:  0.5
```


```
Displaying matches for ratio:  0.5  and sigma:  0.5
```


```
Displaying matches for ratio:  0.67  and sigma:  0.5
```




```
Displaying matches for ratio:  0.9  and sigma:  0.5
```



Q2.1.6 (10 points):

Implement the function briefRot

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

        image_rotated = scipy.ndimage.rotate(image, i, reshape=False)

        # TODO: Match features in images

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

        # TODO: visualizes matches at at least 3 different
orientations
        # to include in your report
        # (Hint: use plotMatches)
        plotMatches(image, image_rotated, matches, locs1, locs2)
        match_counts.append(len(matches))
        match_degrees.append(i)

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

        # ==== 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.

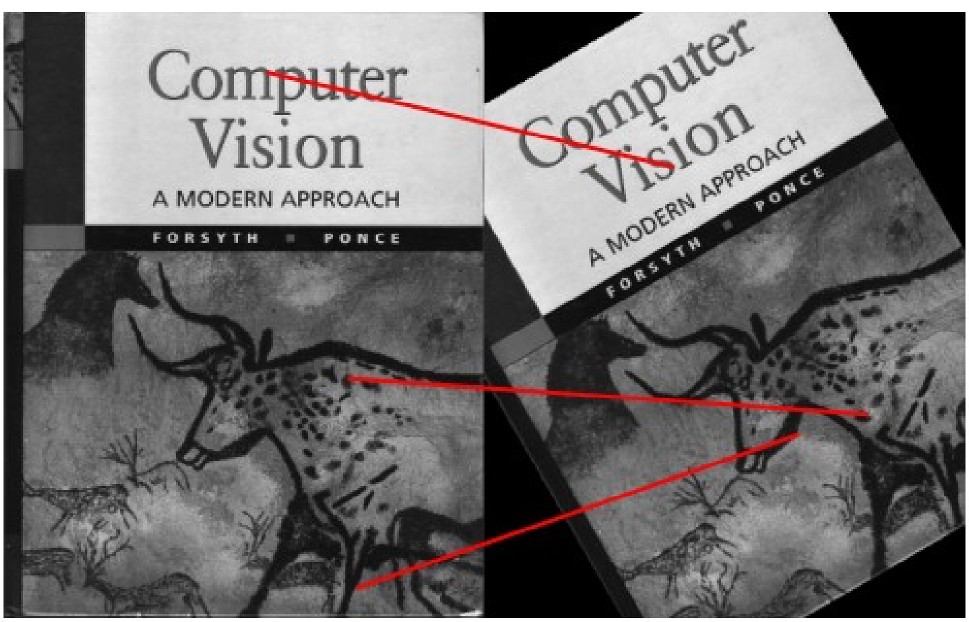
```
# defaults are:
# min_deg = 0
# max_deg = 360
# deg_inc = 10
# ratio = 0.7
# sigma = 0.15
# filename = 'cv_cover.jpg'

# Controls the rotation degrees
min_deg = 0
max_deg = 360
deg_inc = 30

# 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)
```

```
0
```

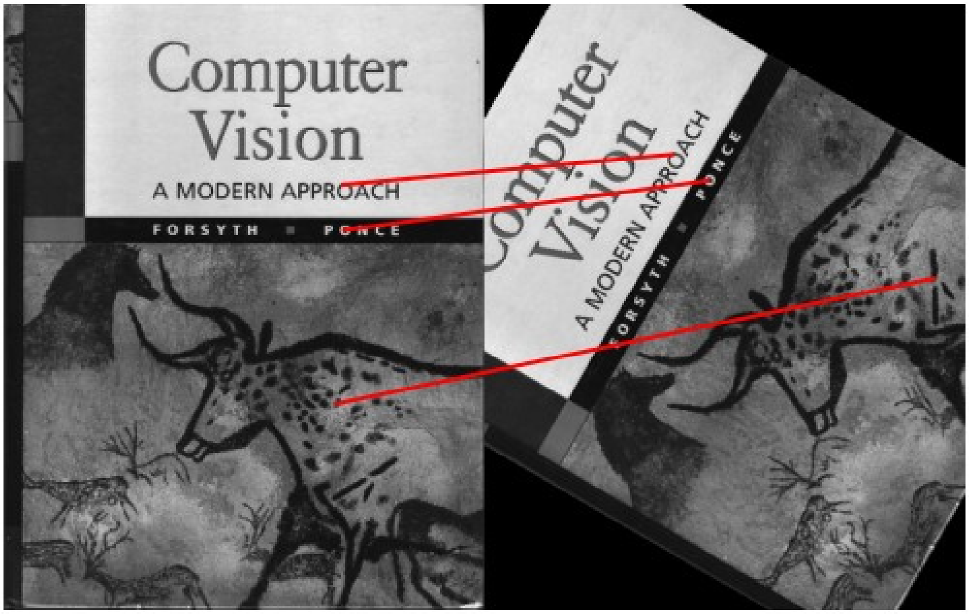
```
/var/folders/43/5971cqb109bfv3wb3b47x9tc0000gn/T/
ipykernel_34345/3301853510.py:58: DeprecationWarning: Conversion of an
array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
```

```
performing this operation. (Deprecated NumPy 1.25.)
  return 1 if img[int(center[0]+row1)][int(center[1]+col1)] <
img[int(center[0]+row2)][int(center[1]+col2)] else 0
/var/folders/43/5971cqb109bfv3wb3b47x9tc0000gn/T/ipykernel_34345/33018
53510.py:35: FutureWarning: `plot_matches` is deprecated since version
0.23 and will be removed in version 0.25. Use
`skimage.feature.plot_matched_features` instead.
  skimage.feature.plot_matches(ax,img1,img2,locs1,locs2,
```

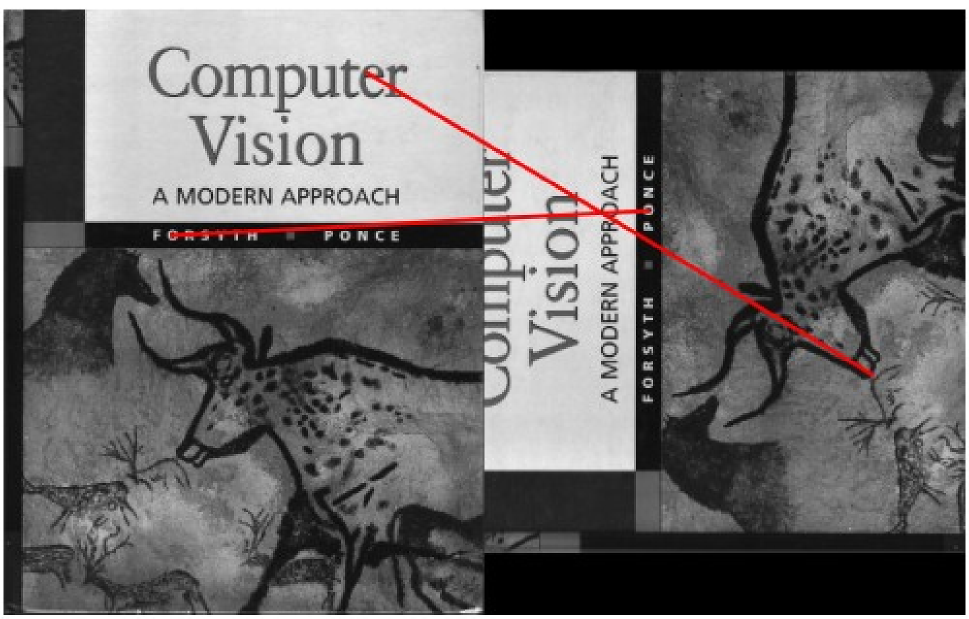
```
30
```



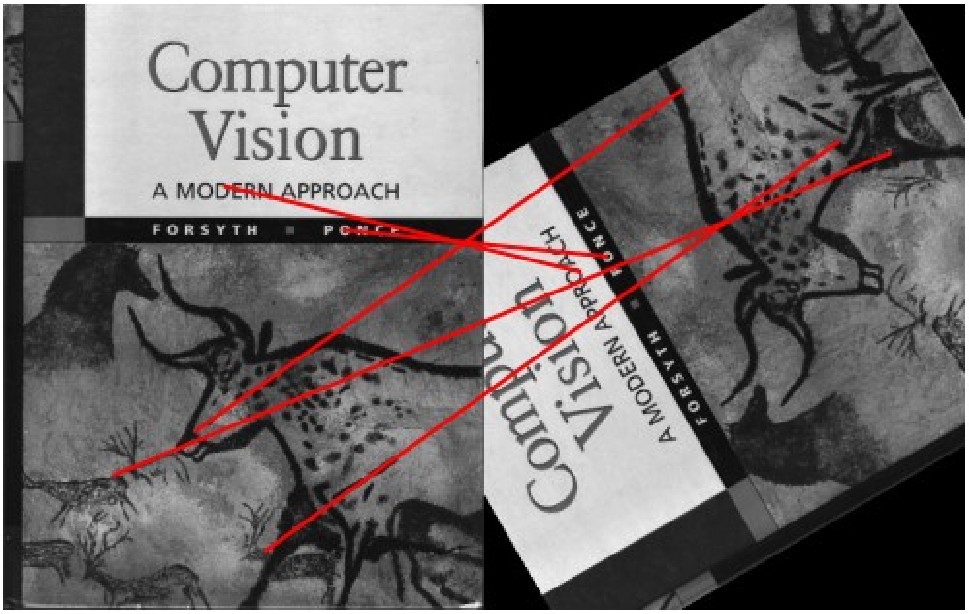
```
60
```


Computer Vision
A MODERN APPROACH
FORSYTH PONCE

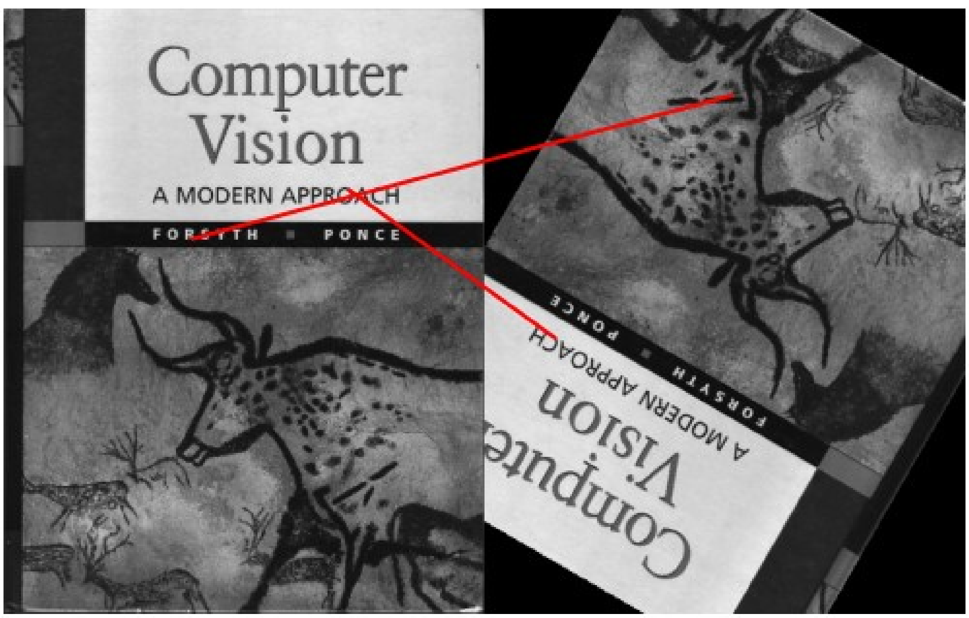
```
90
```


Computer Vision
A MODERN APPROACH
FORSYTH PONCE

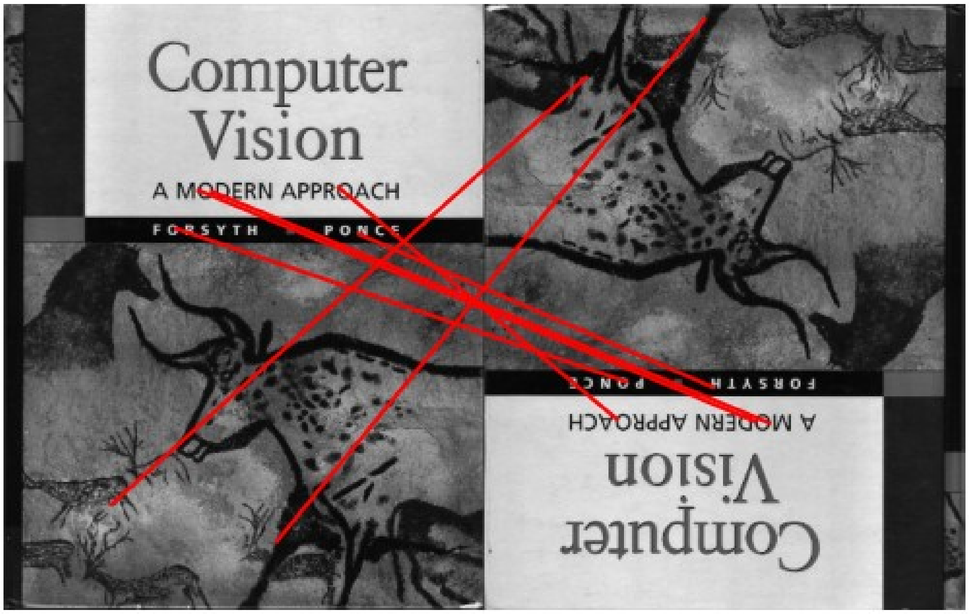
```
120
```


Computer
Vision
A MODERN APPROACH
FORSYTH ■ PONCE

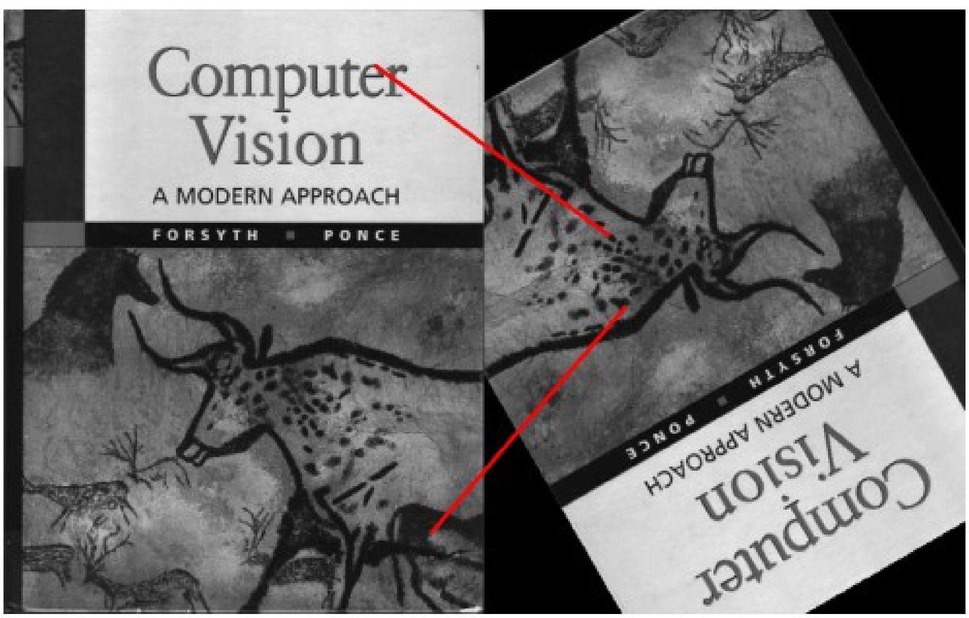
```
150
```


Computer
Vision
A MODERN APPROACH
FORSYTH ■ PONCE

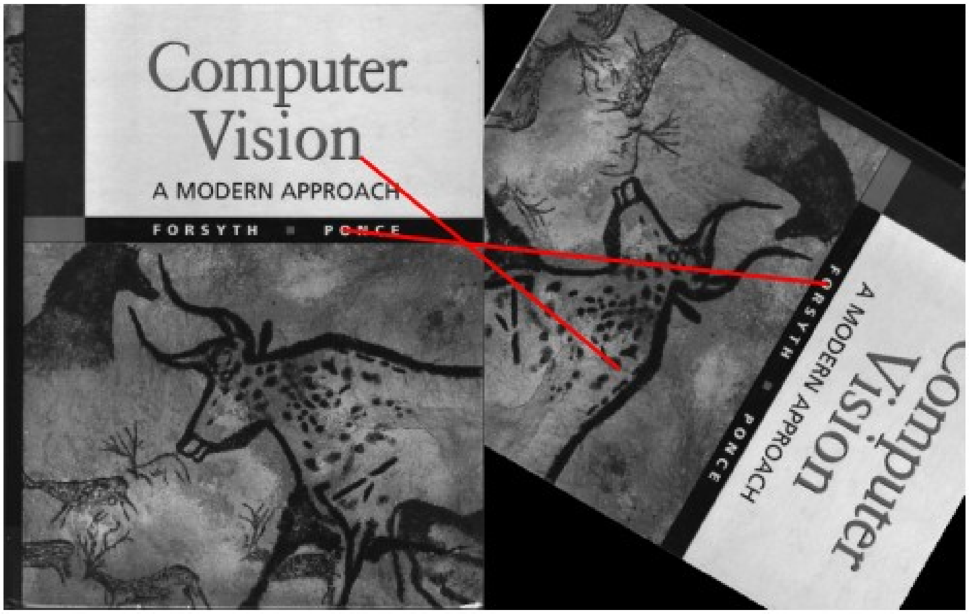
```
180
```


Computer
Vision
A MODERN APPROACH
FORSYTH ■ PONCE

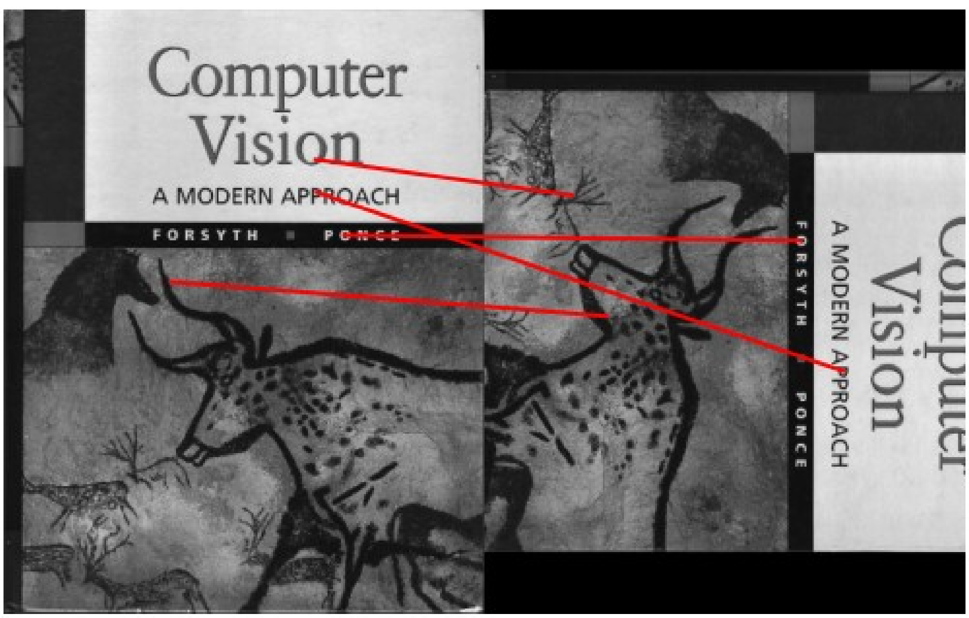
```
210
```


Computer
Vision
A MODERN APPROACH
FORSYTH ■ PONCE

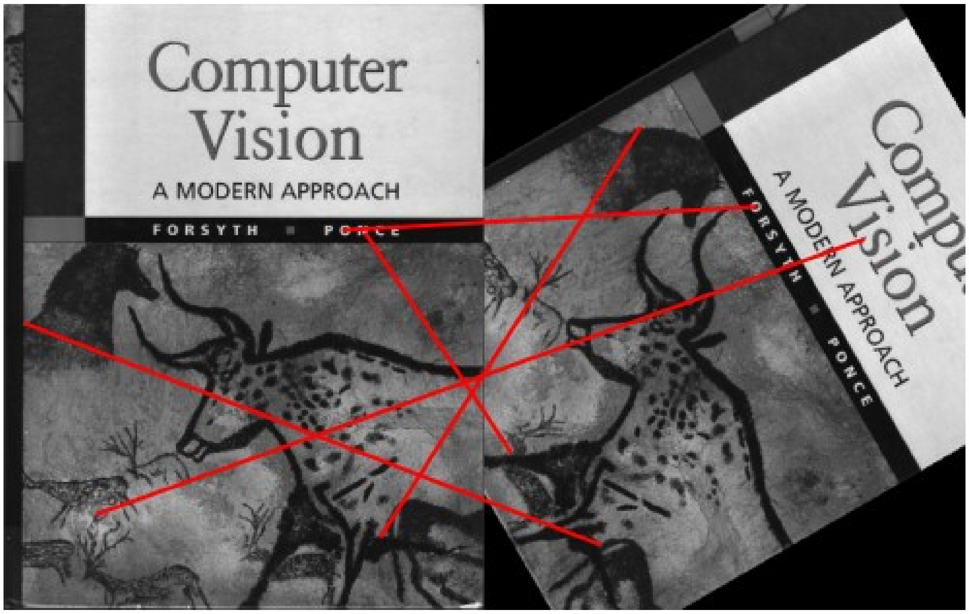
```
240
```


Computer
Vision
A MODERN APPROACH
FORSYTH ■ PONCE
Computer
Vision
A MODERN APPROACH
FORSYTH ■ PONCE

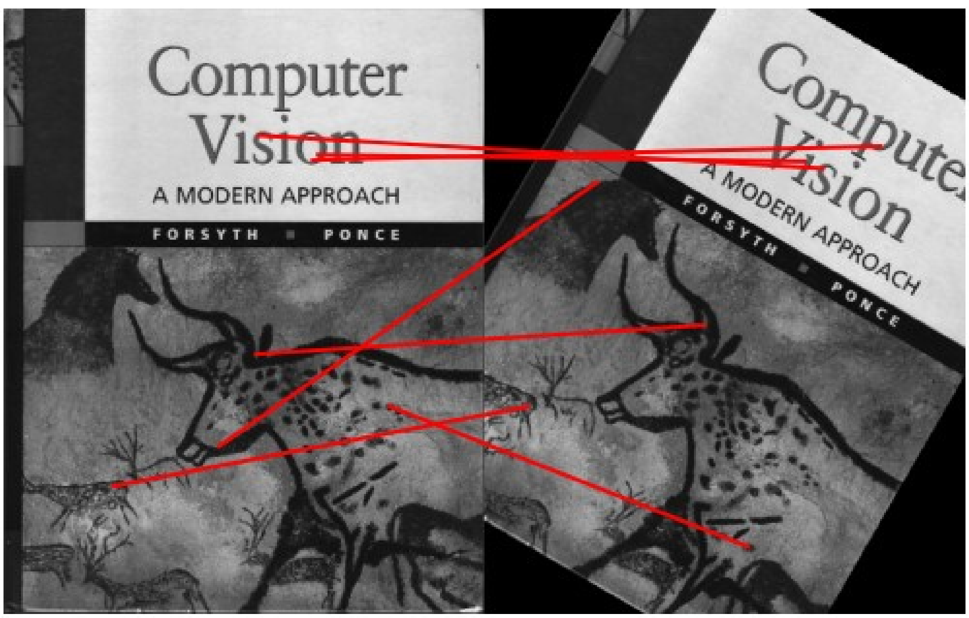
```
270
```


Computer
Vision
A MODERN APPROACH
FORSYTH ■ PONCE
Vision
A MODERN APPROACH
FORSYTH ■ PONCE

```
300
```



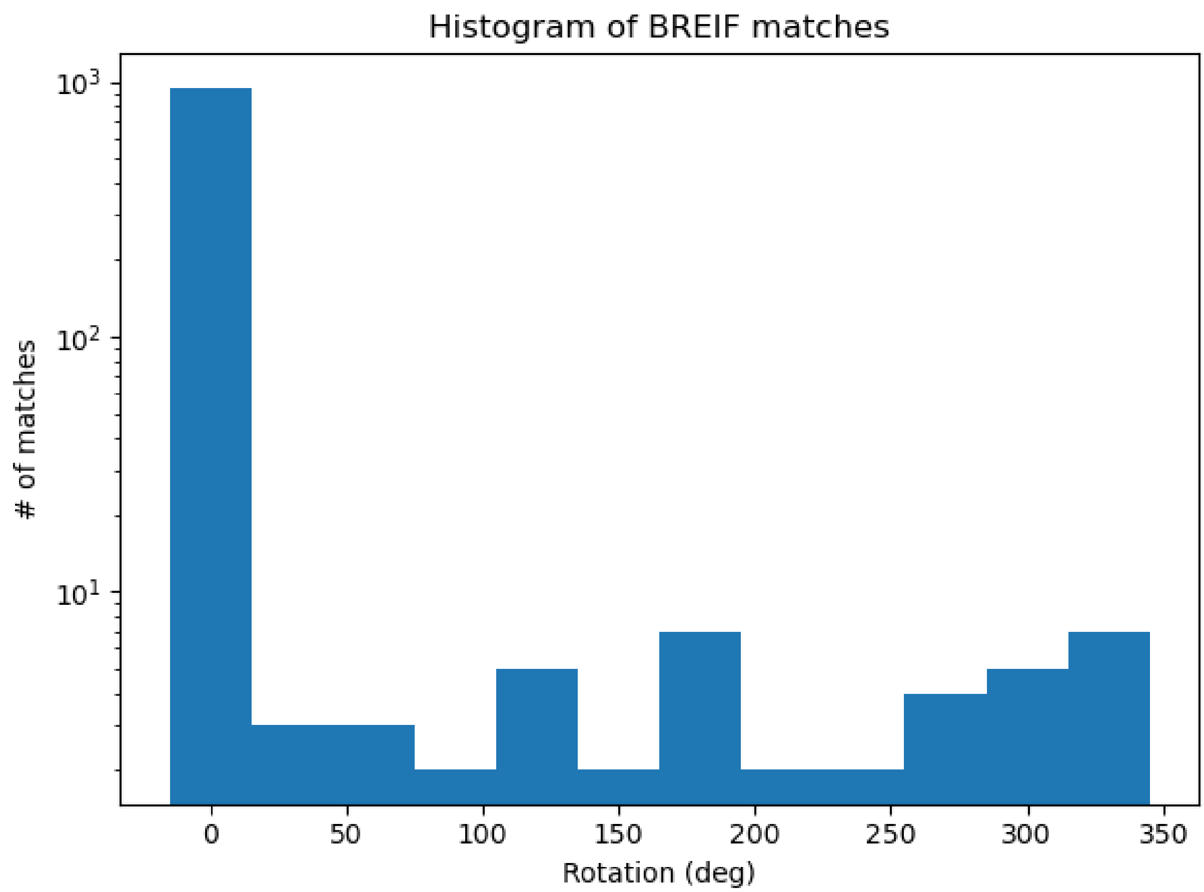

```
330
```



Plot the histogram

See debugging tips in handout.

```
dispBriefRotHist()
```



BRIEF generates a strign description of the oriented template of the patch. on rotating, the patch orientation changes and the corresponding brief descriptor also changes since the indices of comparison have rotated but the decriptors are being generated from the original unrotated location resulting in low similarity.

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.

```
import os
import cv2
import scipy.ndimage
import numpy as np
from sympy import deg

# Path for saving the rotation-invariant matches
ROT_INV_MATCHES_PATH = os.path.join(RES_DIR, 'rot_inv_matches.pkl')
```

```
def rotate_points(points, angle, shape):
    """
    Rotate points by a specified angle around the center of an image.

    Input
    -----
    points: Array of points (coordinates) to rotate
    angle: Angle to rotate by (in degrees)
    shape: Shape of the image (height, width)

    Returns
    -------
    rotated_points: Array of points rotated back to original
orientation
    """
    # Convert angle to radians
    angle_rad = np.deg2rad(angle)

    # Calculate center of the image
    center_x = shape[1] / 2
    center_y = shape[0] / 2

    # Create rotation matrix
    rotation_matrix = np.array([
        [np.cos(angle_rad), -np.sin(angle_rad)],
        [np.sin(angle_rad), np.cos(angle_rad)]
    ])

    # Rotate each point
    rotated_points = []
    for point in points:
        # Translate point to origin, apply rotation, then translate
back
        translated_point = point - np.array([center_x, center_y])
        rotated_point = rotation_matrix @ translated_point +
np.array([center_x, center_y])
        rotated_points.append(rotated_point)

    return np.array(rotated_points)


def briefRotBruteForce(image, image_rotated, ratio, sigma):
    """
    Perform a brute-force search for the rotation angle that maximizes
    matches between an image and its rotated versions.

    Input
    -----
    image: Original image
    image_rotated: The second image that will be rotated
```

```
    ratio: Ratio for BRIEF feature descriptor
    sigma: Threshold for corner detection using FAST feature detector

    Returns
    -------
    best_matches: The best matches obtained after rotation
    best_locs1: Locations in the original image
    best_locs2: Rotated locations back to original orientation in the
rotated image
    best_angle: The angle that maximizes matches
    max_matches: Maximum number of matches found
    """

    best_angle = 0
    max_matches = 0
    best_matches = None
    best_locs1 = None
    best_locs2_rotated = None  # Rotated back to match original
locations

    min_deg = 0
    deg_inc = 60
    max_deg = 360
    # Loop through rotation degrees
    for angle in range(min_deg, max_deg + 1, deg_inc):
        print(f"Rotating by {angle} degrees")

        # Rotate image_rotated by the current angle
        image_rotated_from_original =
scipy.ndimage.rotate(image_rotated, angle, reshape=False)

        # Match features between the original and rotated images
        matches, locs1, locs2 = matchPics(image,
image_rotated_from_original, ratio, sigma)

        # Check if this rotation results in the highest number of
matches
        num_matches = len(matches)
        if num_matches > max_matches:
            max_matches = num_matches
            best_angle = angle
            best_matches = matches
            best_locs1 = locs1
            best_locs2 = locs2

            # Rotate locs2 back to original coordinates
            best_locs2_rotated = rotate_points(best_locs2, -angle,
image_rotated.shape)

    return best_matches, best_locs1, best_locs2_rotated, best_angle,
```

```
max_matches

def briefRotInvEc(min_deg, max_deg, deg_inc, ratio, sigma, filename):
    """
    Perform a brute-force search to find the rotation angle that 
maximizes
    matches between an image and its rotated versions.

    Input
    -----
    min_deg: Minimum degree to rotate image
    max_deg: Maximum degree to rotate image
    deg_inc: Degree increment for each iteration
    ratio: Ratio for BRIEF feature descriptor
    sigma: Threshold for corner detection using FAST feature detector
    filename: Filename of image to rotate
    """

    # Check if results directory exists
    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 if necessary
    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)

    # Loop through rotation degrees (brute-force search)
    for angle in range(min_deg, max_deg + 1, deg_inc):
        print(f"Rotating by {angle} degrees")

        # Rotate the image using scipy's rotate function
        image_rotated = scipy.ndimage.rotate(image, angle, 
reshape=False)

        matches, locs1, locs2_rotated, best_angle, num_matches = 
briefRotBruteForce(image, image_rotated, ratio, sigma)

        # Update if this rotation has the most matches

        # Display and save results for the best rotation angle
        print(f"Best angle: {best_angle} degrees with {num_matches} 
matches")
        plotMatches(image, image_rotated, matches, locs1, 
locs2_rotated)
```

Visualize your implemented function

```
# Set parameters
min_deg = 0
max_deg = 360
deg_inc = 90
filename = 'cv_cover.jpg'
ratio = 0.9  # Set an appropriate ratio for feature matching
sigma = 0.15   # Set a sigma value for corner detection

# Call briefRotInvEc function to perform rotation-invariant BRIEF
matching
briefRotInvEc(min_deg, max_deg, deg_inc, ratio, sigma, filename)
```

```
Rotating by 0 degrees
Rotating by 0 degrees
```

```
/var/folders/43/5971cqb109bfv3wb3b47x9tc0000gn/T/
ipykernel_34345/3301853510.py:58: 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
performing this operation. (Deprecated NumPy 1.25.)
  return 1 if img[int(center[0]+row1)][int(center[1]+col1)] <
img[int(center[0]+row2)][int(center[1]+col2)] else 0
```

```
Rotating by 60 degrees
Rotating by 120 degrees
Rotating by 180 degrees
Rotating by 240 degrees
Rotating by 300 degrees
Rotating by 360 degrees
Best angle: 0 degrees with 945 matches
```

```
/var/folders/43/5971cqb109bfv3wb3b47x9tc0000gn/T/
ipykernel_34345/3301853510.py:35: FutureWarning: `plot_matches` is
deprecated since version 0.23 and will be removed in version 0.25. Use
`skimage.feature.plot_matched_features` instead.
  skimage.feature.plot_matches(ax,img1,img2,locs1,locs2,
```

```
Rotating by 90 degrees
Rotating by 0 degrees
Rotating by 60 degrees
Rotating by 120 degrees
Rotating by 180 degrees
Rotating by 240 degrees
Rotating by 300 degrees
Rotating by 360 degrees
Best angle: 240 degrees with 113 matches
```

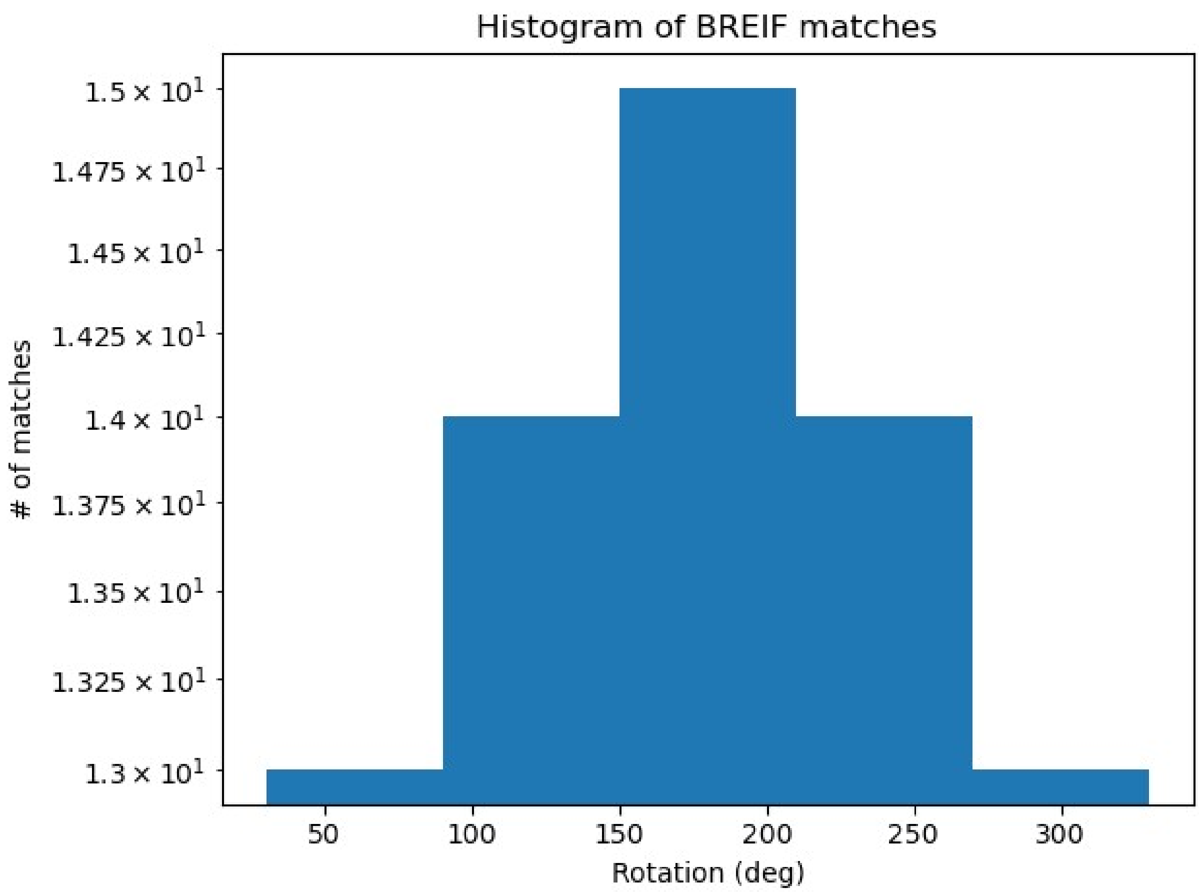
```
Rotating by 180 degrees
Rotating by 0 degrees
Rotating by 60 degrees
Rotating by 120 degrees
Rotating by 180 degrees
Rotating by 240 degrees
Rotating by 300 degrees
Rotating by 360 degrees
Best angle: 180 degrees with 945 matches
```


```
Rotating by 270 degrees
Rotating by 0 degrees
Rotating by 60 degrees
Rotating by 120 degrees
Rotating by 180 degrees
Rotating by 240 degrees
Rotating by 300 degrees
Rotating by 360 degrees
Best angle: 60 degrees with 113 matches
```


```
Rotating by 360 degrees
Rotating by 0 degrees
Rotating by 60 degrees
Rotating by 120 degrees
Rotating by 180 degrees
Rotating by 240 degrees
Rotating by 300 degrees
Rotating by 360 degrees
Best angle: 0 degrees with 945 matches
```

Plot Histogram

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



Compare the histograms with an without rotation invariance. Explain your rotation invariant design and how you selected any parameters that you used: This algorithm performs a brute-force search to determine the optimal rotation angle that maximizes feature matches between an original image and its rotated versions. The primary function, `briefRotInvEc`, iterates over a range of angles, rotating the image incrementally and then calling `briefRotBruteForce` to find the best matching angle. In `briefRotBruteForce`, for each angle, the rotated image is compared to the original image using feature descriptors (e.g., BRIEF) to identify and match distinctive points.

The number of matches for each angle is tracked, and the function updates the angle with the highest match count as the "best angle." To account for changes in coordinate positions due to rotation, the function also includes a helper function, `rotate_points`, which rotates matched points back to their original orientation, aligning them with the original image's coordinate system. Finally, the algorithm returns the highest match count, best angle, and visualizes the optimal matches between the two images. This approach is useful for applications requiring rotation-invariant feature matching, such as object recognition in rotated images.

For the rotation invariant version the histogram is relatively flat since now the decriptors are rotation agnostic by bruteforce search hence it is almost as good as aligned image comparison

---

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.

```
import os
import cv2
import numpy as np
from matplotlib import pyplot as plt

def briefScaleBruteForce(image, image2, ratio, sigma, min_scale=0.5,
max_scale=2.0, scale_inc=0.5):
    """
    Perform a brute-force search for the scaling factor that maximizes
    matches between an original image and its scaled versions.

    Input
    -----
    image: Original image
    scaled_image: The image after applying initial scaling
    ratio: Ratio for BRIEF feature descriptor
    sigma: Threshold for corner detection using FAST feature detector
    min_scale: Minimum scale factor
    max_scale: Maximum scale factor
    scale_inc: Scale increment for each iteration

    Returns
    -------
    best_matches: The best matches obtained after scaling
    best_locs1: Locations in the original image
    best_locs2_scaled: Scaled locations back to original coordinates
    best_scale: The scale factor that maximizes matches
    max_matches: Maximum number of matches found
    """

    best_scale = 1.0
    max_matches = 0
    best_matches = None
    best_locs1 = None
    best_locs2_scaled = None

    # Loop through scale factors
    scale_factors = np.arange(min_scale, max_scale + scale_inc,
scale_inc)
    for scale in scale_factors:
```

```
        print(f"Testing scale factor: {scale}")

        # Scale the image
        scaled_shape = (int(image2.shape[1] * scale),
int(image2.shape[0] * scale))
        image_scaled = cv2.resize(image2, scaled_shape,
interpolation=cv2.INTER_AREA)

        # Match features between original and scaled images
        matches, locs1, locs2 = matchPics(image, image_scaled, ratio,
sigma)

        # Check if this scale results in the highest number of matches
        num_matches = len(matches)
        if num_matches > max_matches:
            max_matches = num_matches
            best_scale = scale
            best_matches = matches
            best_locs1 = locs1
            best_locs2 = locs2 / scale

            # Scale locs2 back to original coordinates
            best_locs2_scaled = locs2 / scale

    return best_matches, best_locs1, best_locs2_scaled, best_scale,
max_matches

def briefScaleInvEc(ratio, sigma, filename, initial_scales=[0.5, 1.0,
1.5, 2.5], min_scale=0.5, max_scale=2.0, scale_inc=0.5):
    """
    Scale image and use brute-force search to find the scale factor
that maximizes
    matches between an image and its scaled versions.

    Input
    -----
    ratio: Ratio for BRIEF feature descriptor
    sigma: Threshold for corner detection using FAST feature detector
    filename: Filename of the image to scale
    initial_scales: List of initial scaling factors for brute-force
search
    min_scale: Minimum scale factor
    max_scale: Maximum scale factor
    scale_inc: Scale increment for each iteration
    """

    # Read the image and convert BGR to RGB if necessary
    image_path = os.path.join(DATA_DIR, filename)
    image = cv2.imread(image_path)
    if len(image.shape) == 3 and image.shape[2] == 3:
```

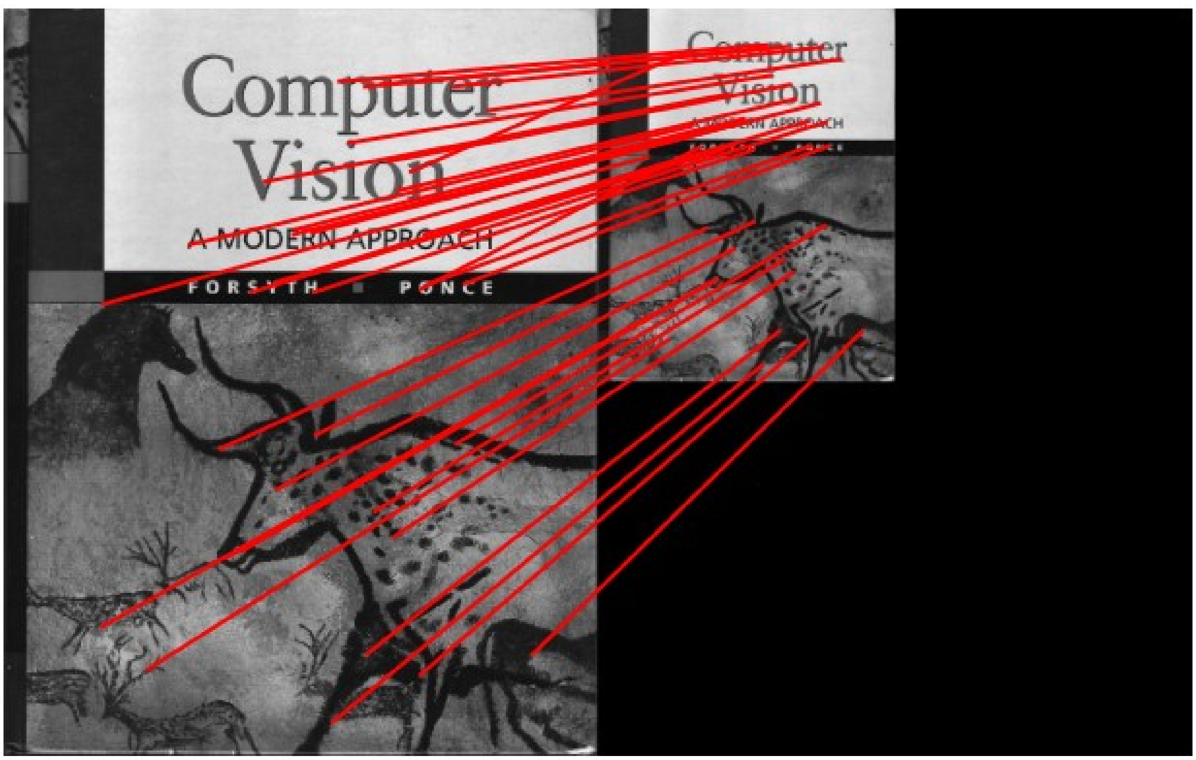
```
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # Loop through each initial scale and perform brute-force search
    global_best_scale = 1.0
    global_max_matches = 0
    global_best_matches = None
    global_best_locs1 = None
    global_best_locs2_scaled = None

    for init_scale in initial_scales:
        print(f"\nStarting brute-force search with initial scale: 
{init_scale}")

        # Create an initial scaled version of the image
        initial_scaled_image = cv2.resize(image, (int(image.shape[1] * 
init_scale), int(image.shape[0] * init_scale)))

        # Run brute-force scale search starting from this initial 
scale
        matches, locs1, locs2_scaled, best_scale, max_matches = 
briefScaleBruteForce(
            image, initial_scaled_image, ratio, sigma, min_scale, 
max_scale, scale_inc)

        # Display and save results for the best scale factor found 
across all initial scales
        print(f"\nBest scale across all initial scales: 
{global_best_scale} with {global_max_matches} matches")
        print(best_scale, max_matches)
        plotMatches(image, initial_scaled_image,
                    matches, locs1, locs2_scaled)
```

Visualize your implemented function

```
# ===== 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 ====
```

```
Starting brute-force search with initial scale: 0.5
Testing scale factor: 0.5
```

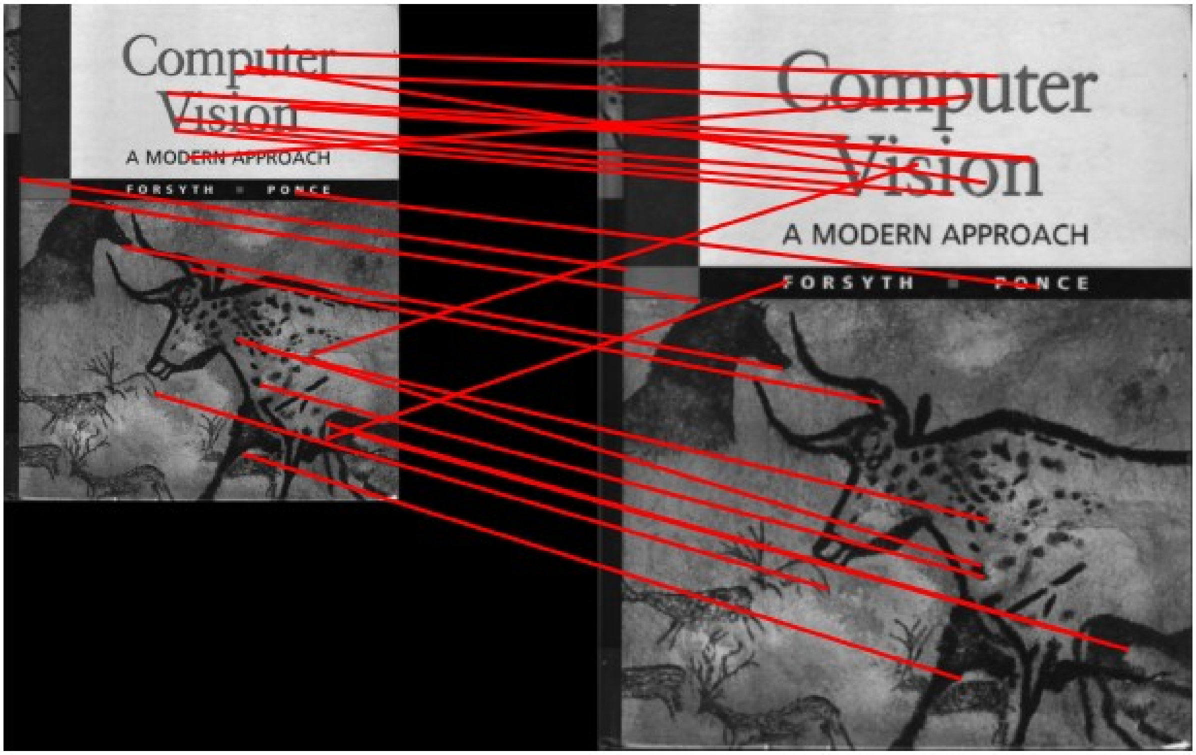
```
/var/folders/43/5971cqb109bfv3wb3b47x9tc0000gn/T/
ipykernel_34345/3301853510.py:58: DeprecationWarning: Conversion of an
array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  return 1 if img[int(center[0]+row1)][int(center[1]+col1)] <
img[int(center[0]+row2)][int(center[1]+col2)] else 0

Testing scale factor: 1.0
Testing scale factor: 1.5
Testing scale factor: 2.0

Best scale across all initial scales: 1.0 with 0 matches
2.0 38

/var/folders/43/5971cqb109bfv3wb3b47x9tc0000gn/T/
ipykernel_34345/1842640990.py:116: FutureWarning: `plot_matches` is
deprecated since version 0.23 and will be removed in version 0.25. Use
`skimage.feature.plot_matched_features` instead.
  skimage.feature.plot_matches(ax, img1_gray, img2_gray, locs1, locs2,
```



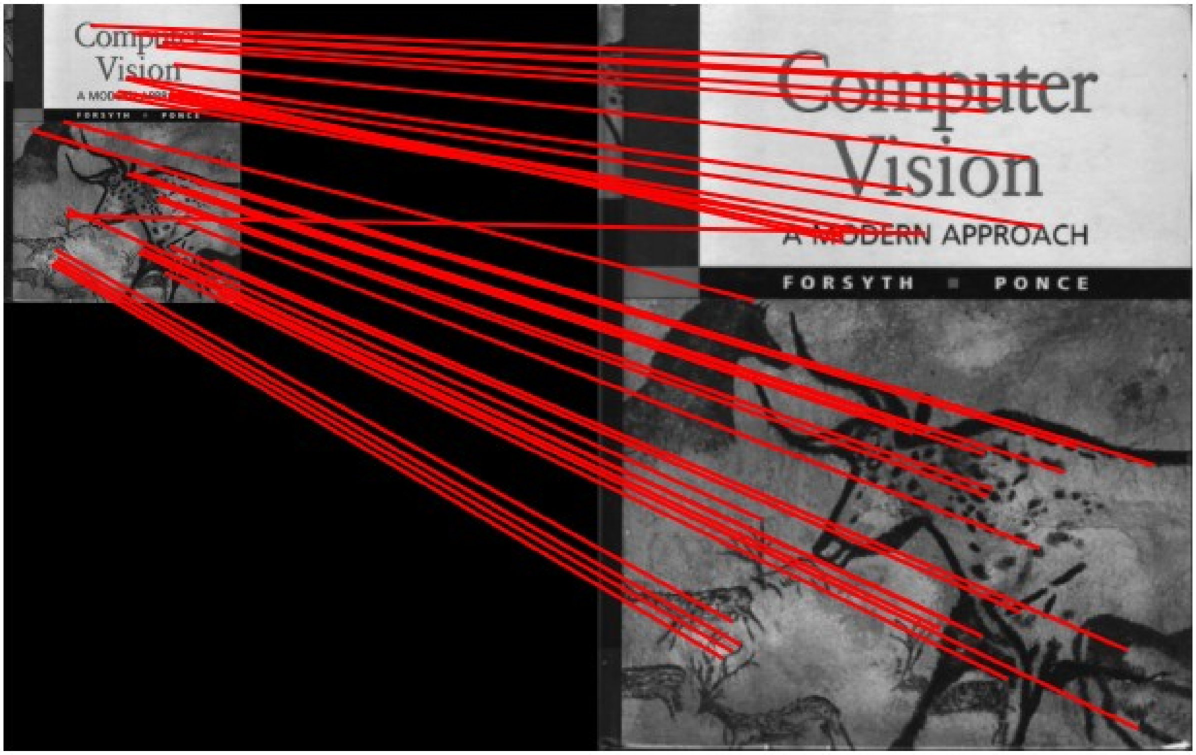

```
Starting brute-force search with initial scale: 1.0
Testing scale factor: 0.5
Testing scale factor: 1.0
Testing scale factor: 1.5
```

```
Testing scale factor: 2.0

Best scale across all initial scales: 1.0 with 0 matches
1.0 945
```

```


Starting brute-force search with initial scale: 1.5
Testing scale factor: 0.5
Testing scale factor: 1.0
Testing scale factor: 1.5
Testing scale factor: 2.0

Best scale across all initial scales: 1.0 with 0 matches
0.5 24
```




```
Starting brute-force search with initial scale: 2.5
Testing scale factor: 0.5
Testing scale factor: 1.0
Testing scale factor: 1.5
Testing scale factor: 2.0

Best scale across all initial scales: 1.0 with 0 matches
0.5 38
```



Explain your scale invariant design and how you selected any parameters that you used: This code implements a brute-force search algorithm to find the best scale factor that maximizes feature matches between an original image and scaled versions of a second image using the BRIEF (Binary Robust Independent Elementary Features) feature descriptor. The `briefScaleBruteForce` function takes two images (`image` and `image2`) and iterates through a range of scaling factors for `image2`, resizing it each time to calculate the number of feature matches with the original image. For each scale factor, it identifies the locations of matched features, adjusts them back to the original scale, and tracks the best scale that provides the highest match count. The `briefScaleInvEc` function extends this search by initializing `image2` at several predefined scale factors (e.g., `[0.5, 1.0, 1.5, 2.5]`) before passing each initialization to `briefScaleBruteForce`. This allows a multi-level brute-force approach, where each initial scaling factor is explored over a finer range of scales, increasing accuracy in finding the best match. After each initialization, it visualizes the matches, giving insight into how well features align across scales.

Q2.2 Homography Computation

Q2.2.1 (15 Points):

Implement the function computeH

```python
import numpy as np

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

    Input
    -----
    x1, x2: Nx2 arrays of corresponding points from two images.

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

    if x1.shape != x2.shape:
        raise ValueError('The number of points in x1 and x2 must match.')

    # Ensure that both x1 and x2 are 2D points
    assert x1.shape[1] == 2 and x2.shape[1] == 2, "Input points must be 2D coordinates."

    # Convert points to homogeneous coordinates if needed
    if x1.shape[1] == 2:
        x1 = np.hstack((x1, np.ones((x1.shape[0], 1))))
    if x2.shape[1] == 2:
        x2 = np.hstack((x2, np.ones((x2.shape[0], 1))))

    # Initialize the matrix A for the system of equations A*h = 0
    A = np.zeros((2 * x1.shape[0], 9))

    # Construct the A matrix using the point correspondences
    for i in range(x1.shape[0]):
        x, y, z = x1[i]
        x_, y_, z_ = x2[i]

        A[2 * i] = [-x_, -y_, -1, 0, 0, 0, x_ * x, y_ * x,  x]
        A[2 * i + 1] = [0, 0, 0, -x_, -y_, -1, x_ * y, y_ * y,  y]

    # Perform SVD (Singular Value Decomposition) on the A matrix
    U, S, VT = np.linalg.svd(A)
    h = VT[-1]  # The last row of V^T (or last column of V) gives the solution

    # Reshape the resulting 9x1 vector into the 3x3 homography matrix
    H2to1 = h.reshape((3, 3))

    # Normalize H so that H[2,2] = 1 (this is optional but standard practice)
```

```
    H2to1 = H2to1 / H2to1[2, 2]

    return H2to1
```

Q2.2.2 (10 points):

Implement the function computeH_norm

```
import numpy as np

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

    Input
    -----
    x1, x2: Nx2 arrays of corresponding points from two images.

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

    if x1.shape != x2.shape:
        raise ValueError('The number of points in x1 and x2 must
match.')
    if x1.shape[0] < 4:
        raise ValueError('At least 4 points are required to compute a
homography.')

    # Compute the centroids of the points
    x1_centroid = np.mean(x1, axis=0)
    x2_centroid = np.mean(x2, axis=0)

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

    # Compute the mean distance of the points from the origin
    x1_mean_dist = np.mean(np.linalg.norm(x1_shifted, axis=1))
    x2_mean_dist = np.mean(np.linalg.norm(x2_shifted, axis=1))

    # Similarity transform for x1
    T1 = np.array([[np.sqrt(2) / x1_mean_dist, 0, -(np.sqrt(2) /
x1_mean_dist) * x1_centroid[0]],
                   [0, np.sqrt(2) / x1_mean_dist, -(np.sqrt(2) /
x1_mean_dist) * x1_centroid[1]],
                   [0, 0, 1]])
```

```
    # Similarity transform for x2
    T2 = np.array([[np.sqrt(2) / x2_mean_dist, 0, -(np.sqrt(2) /
x2_mean_dist) * x2_centroid[0]],
                   [0, np.sqrt(2) / x2_mean_dist, -(np.sqrt(2) /
x2_mean_dist) * x2_centroid[1]],
                   [0, 0, 1]])

    # Convert points to homogeneous coordinates
    x1_h = np.hstack((x1, np.ones((x1.shape[0], 1))))
    x2_h = np.hstack((x2, np.ones((x2.shape[0], 1))))

    # Apply the similarity transforms
    x1_normalized = (T1 @ x1_h.T).T
    x2_normalized = (T2 @ x2_h.T).T

    # Compute homography using the normalized points
    H2to1 = computeH(x1_normalized[:, :2], x2_normalized[:, :2])  #
Use the first two columns

    # Denormalize the homography
    H2to1 = np.linalg.inv(T1) @ H2to1 @ T2
    H2to1 = H2to1 / H2to1[2, 2]
    return H2to1
```

Q2.2.3 (25 points):

Implement RANSAC

```
import numpy as np
import cv2

def computeH_ransac(locs1, locs2, max_iters=1000, inlier_tol=5.0):
    """
    Estimate the homography between two sets of points using RANSAC.

    Parameters:
    -----------
    locs1, locs2: Nx2 arrays of corresponding points (source and
destination points).
    max_iters: Number of iterations to run RANSAC.
    inlier_tol: Tolerance value for considering a point to be an
inlier.

    Returns:
    --------
    bestH2to1: 3x3 homography matrix that best transforms locs2 to
locs1.
    best_inliers: Indices of RANSAC inliers.
    """
```

```
    num_points = locs1.shape[0]
    bestH2to1 = None
    best_inliers = []
    max_inliers = 0

    # Convert the points to float32 for OpenCV
    locs1 = locs1.astype(np.float32)
    locs2 = locs2.astype(np.float32)

    for i in range(max_iters):
        # Step 1: Randomly sample 4 points to compute the homography
        indices = np.random.choice(num_points, 4, replace=False)
        locs1_sample = locs1[indices]
        locs2_sample = locs2[indices]

        # Step 2: Compute the homography matrix using the 4 sample
points
        H = computeH_norm(locs1_sample, locs2_sample)  # 0 means use
Direct Linear Transform (DLT)

        # Step 3: Transform all locs2 points using the computed
homography
        locs2_transformed =
cv2.perspectiveTransform(np.expand_dims(locs2, axis=1), H)
        locs2_transformed = locs2_transformed.squeeze()  # Remove
unnecessary dimensions

        # Step 4: Compute the distances between transformed locs2 and
locs1
        distances = np.linalg.norm(locs2_transformed - locs1, axis=1)

        # Step 5: Determine the inliers (distances less than
inlier_tol)
        inliers = np.where(distances < inlier_tol)[0]

        # Step 6: Update if this set of inliers is the largest found
so far
        if len(inliers) > max_inliers:
            max_inliers = len(inliers)
            best_inliers = inliers
            bestH2to1 = H

    return bestH2to1, best_inliers

# Example usage:
# locs1 and locs2 are Nx2 arrays of corresponding points
# max_iters is the number of RANSAC iterations, inlier_tol is the
distance tolerance for inliers
```

Q2.2.4 (10 points):

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
    """

    # Ensure homography matrix is float and 3x3
    H2to1 = H2to1.astype(np.float32)  # Convert homography matrix to
float if necessary

    # Warp the template image to the image using the homography H2to1
    template_warped = cv2.warpPerspective(template, H2to1,
(img.shape[1], img.shape[0]))

    # Create a binary mask where the warped template is not black
(non-zero values in all channels)
    mask = np.any(template_warped != 0, axis=-1).astype(np.uint8)  #
Shape (height, width), single channel

    # Expand mask to match the image channels
    mask = np.repeat(mask[:, :, np.newaxis], 3, axis=2)  # Shape
(height, width, 3)

    # Create a masked version of the background image where the
template will be placed
    img_masked = img * (1 - mask)

    # Add the warped template to the background
    composite_img = img_masked + template_warped

    return composite_img
```

Implement the function warpImage

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

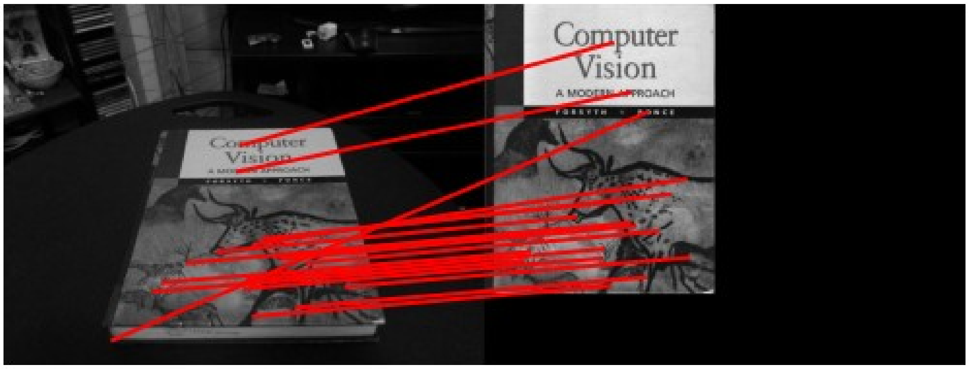
```
    Input
    -----
    ratio: ratio for BRIEF feature descriptor
    sigma: threshold for corner detection using FAST feature detector
    max_iters: the number of iterations to run RANSAC for
    inlier_tol: the tolerance value for considering a point to be an
inlier

    """

    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! =====
    # if image is greascale convert to rgb
    if len(hp_cover.shape) == 2:
        hp_cover = cv2.cvtColor(hp_cover, cv2.COLOR_GRAY2RGB)
    if len(cv_cover.shape) == 2:
        cv_cover = cv2.cvtColor(cv_cover, cv2.COLOR_GRAY2RGB)
    if len(cv_desk.shape) == 2:
        cv_desk = cv2.cvtColor(cv_desk, cv2.COLOR_GRAY2RGB)
    # TODO: match features between cv_desk and cv_cover using
matchPics
    matches, locs1, locs2 = matchPics(cv_desk, cv_cover, ratio, sigma)

    plotMatches(cv_desk, cv_cover, matches, locs1, locs2)

    # TODO: Scale matched pixels in cv_cover to size of hp_cover
    # locs2 = locs2 * (hp_cover.shape[1] / cv_cover.shape[1])

    # TODO: Get homography by RANSAC using computeH_ransac

    # swap columns of corners to follow cv2 convention
    # swap columns of locs1 and locs2
    locs1 = locs1[:, ::-1]
    locs2 = locs2[:, ::-1]

    # locations of harry potter cover
    locs_hp = np.zeros_like(locs2)
    locs_hp[:, 0] = locs2[:, 0] * (hp_cover.shape[1] /
cv_cover.shape[1])
    locs_hp[:, 1] = locs2[:, 1] * (hp_cover.shape[0] /
cv_cover.shape[0])


    H2to1, inliers = computeH_ransac(locs1[matches[:,0]],
```

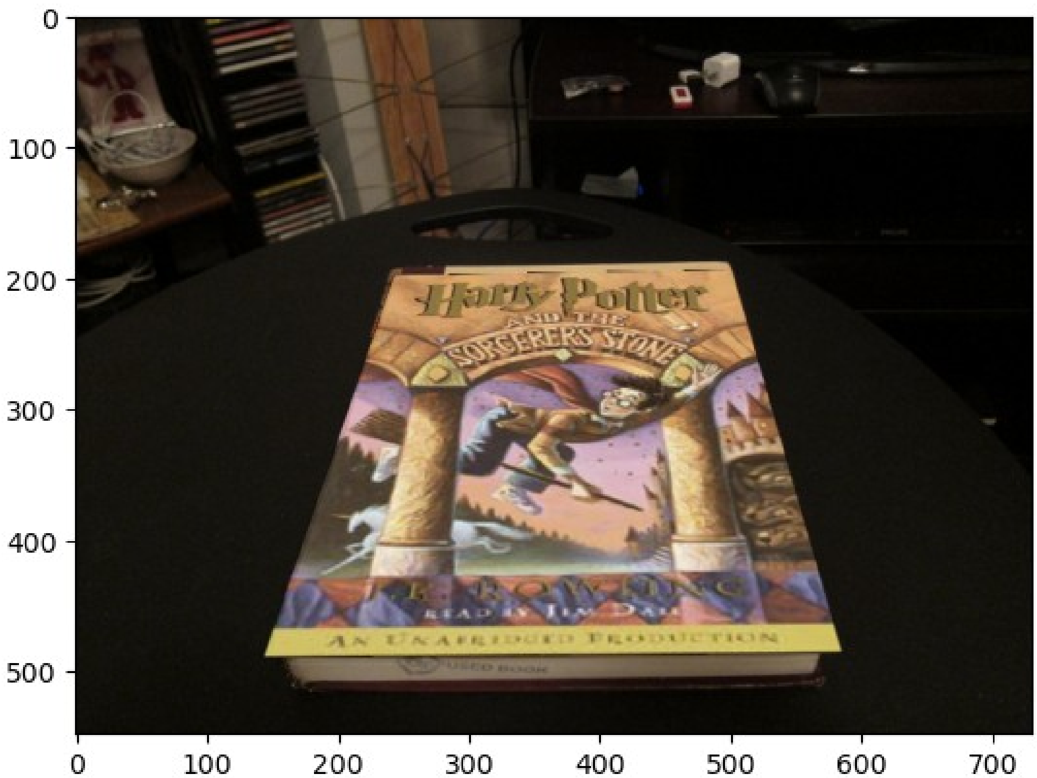
```
locs_hp[matches[:,1]], max_iters, inlier_tol)
    # H2to1, inliers = computeH_ransac(corners2, corners, max_iters,
inlier_tol)

    # TODO: Overlay using compositeH to return composite_img
    composite_img = compositeH(H2to1, hp_cover, cv_desk)
    # print(locs2[inliers])
    # plotMatches(cv_desk, cv_cover, inliers, locs1, locs2)
    # ==== 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 = 1.0

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

```
/var/folders/43/5971cqb109bfv3wb3b47x9tc0000gn/T/
ipykernel_72687/3301853510.py:58: 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
performing this operation. (Deprecated NumPy 1.25.)
  return 1 if img[int(center[0]+row1)][int(center[1]+col1)] <
img[int(center[0]+row2)][int(center[1]+col2)] else 0
/var/folders/43/5971cqb109bfv3wb3b47x9tc0000gn/T/ipykernel_72687/33018
53510.py:35: FutureWarning: `plot_matches` is deprecated since version
0.23 and will be removed in version 0.25. Use
`skimage.feature.plot_matched_features` instead.
  skimage.feature.plot_matches(ax,img1,img2,locs1,locs2,
```





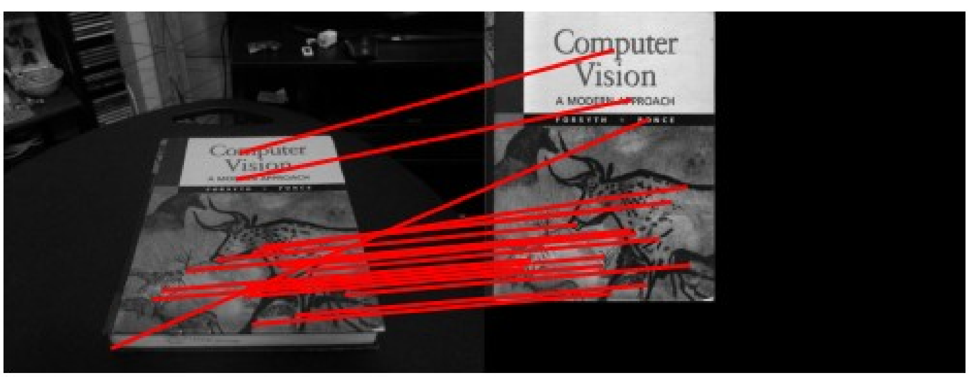
Q2.2.5 (10 points):

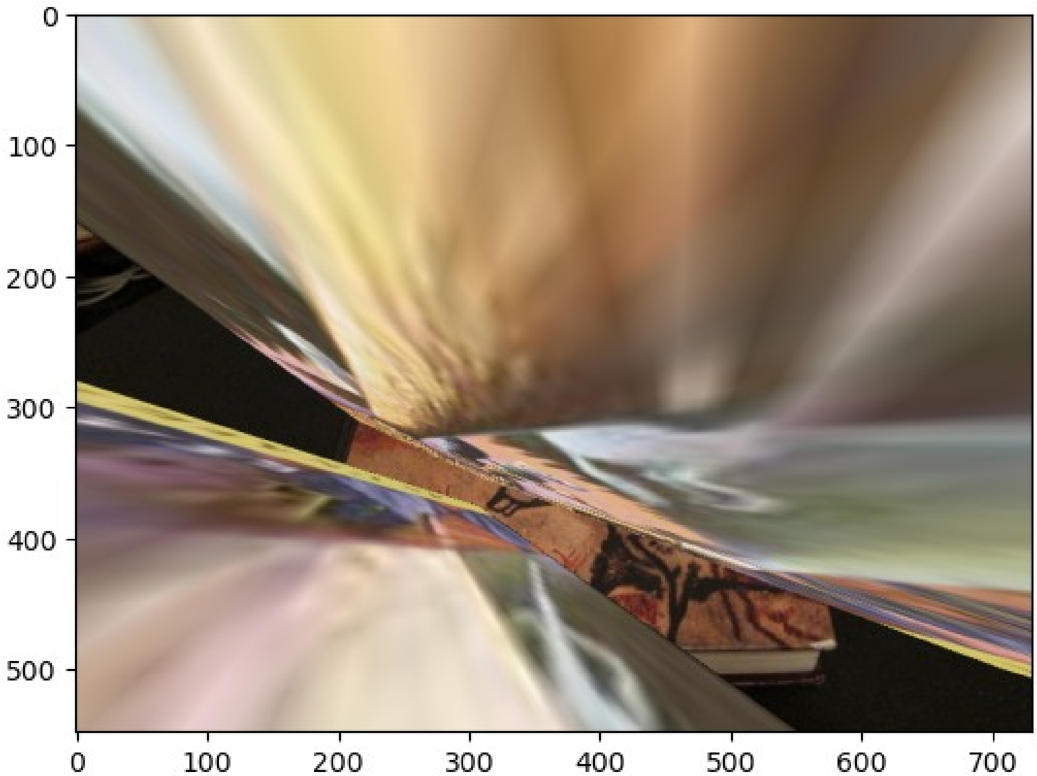
Conduct ablation study with various max_iters and inlier_tol values. Plot the result images and explain the effect of these two parameters respectively.

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


ratio = 0.7
sigma = 0.15
max_iters = 600
```

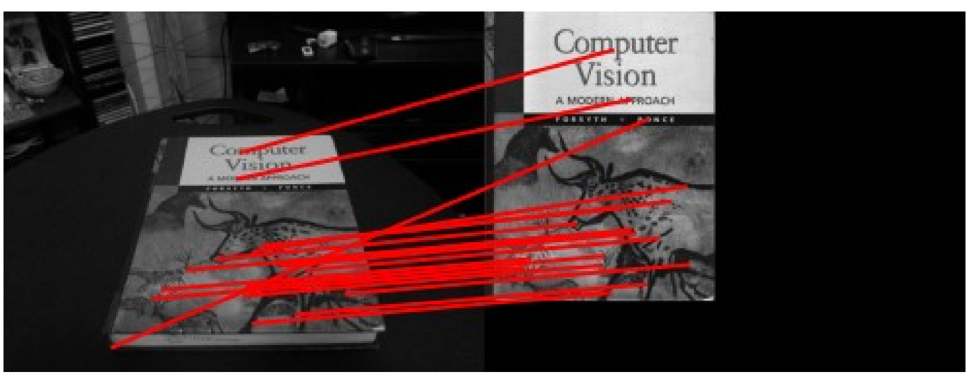
```
inlier_tol = 1.0

for max_iters in [200, 600, 1800]:
    for inlier_tol in [0.1, 0.5, 1.0, 5.0, 20.0]:
        print("=====================================================")
        print('max_iters: ', max_iters, ' inlier_tol: ', inlier_tol)
        warpImage(ratio, sigma, max_iters, inlier_tol)
        print("=====================================================")

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

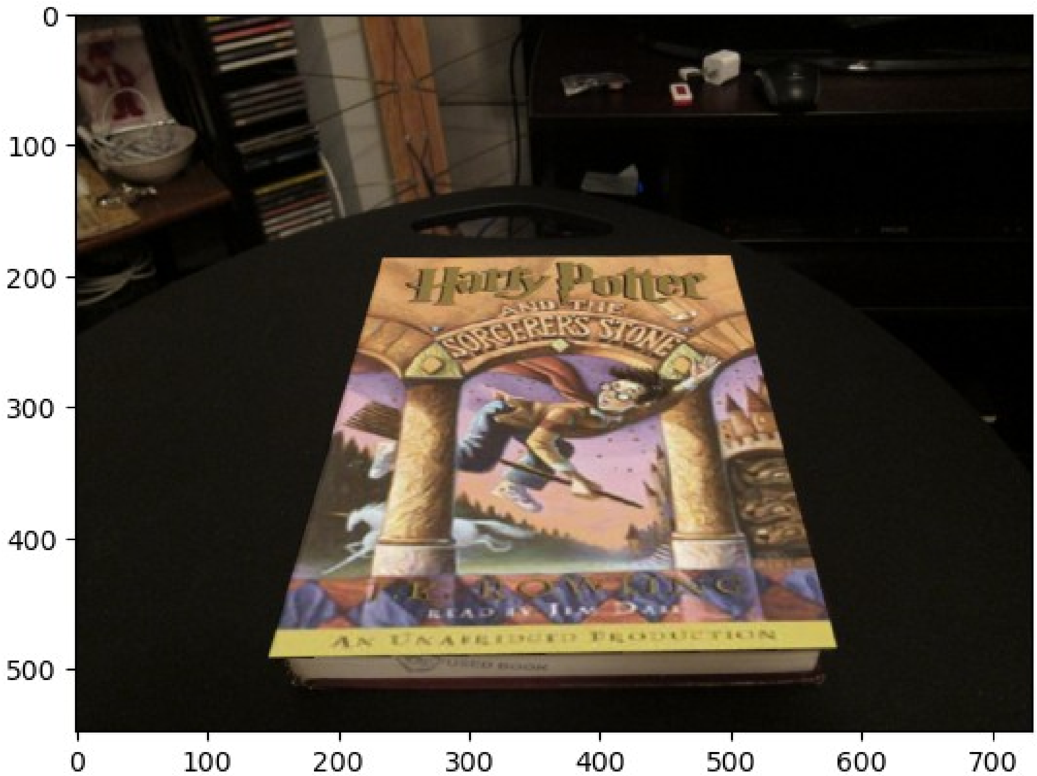
```
=====================================================
max_iters:  200  inlier_tol:  0.1

/var/folders/43/5971cqb109bfv3wb3b47x9tc0000gn/T/
ipykernel_72687/3301853510.py:58: 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
performing this operation. (Deprecated NumPy 1.25.)
  return 1 if img[int(center[0]+row1)][int(center[1]+col1)] <
img[int(center[0]+row2)][int(center[1]+col2)] else 0
/var/folders/43/5971cqb109bfv3wb3b47x9tc0000gn/T/ipykernel_72687/33018
53510.py:35: FutureWarning: `plot_matches` is deprecated since version
0.23 and will be removed in version 0.25. Use
`skimage.feature.plot_matched_features` instead.
  skimage.feature.plot_matches(ax,img1,img2,locs1,locs2,
```



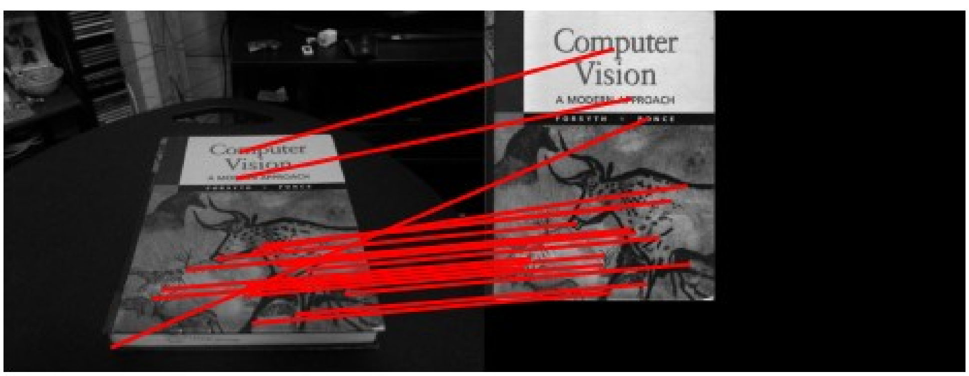


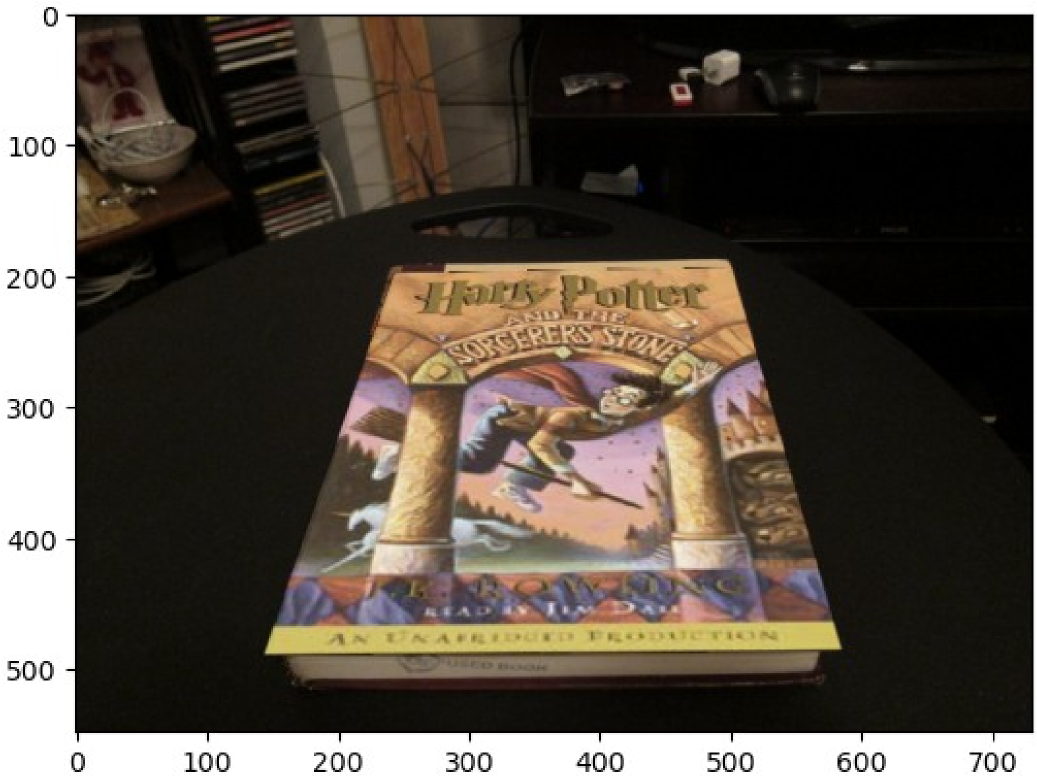
```
==================================================
==================================================
max_iters:  200  inlier_tol:  0.5
```



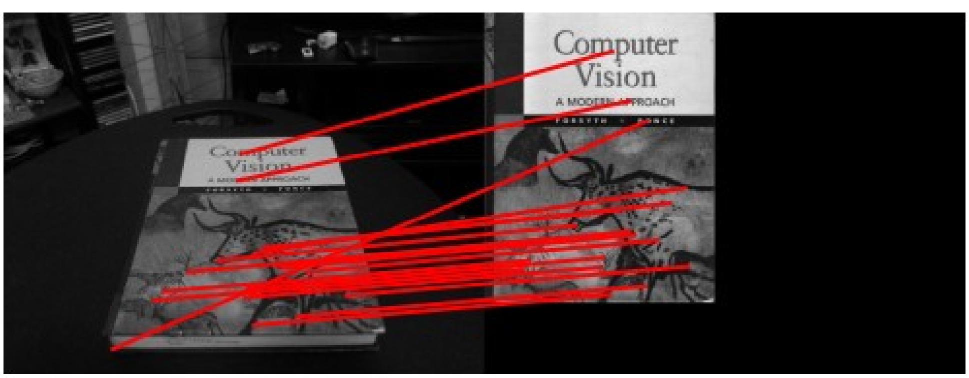


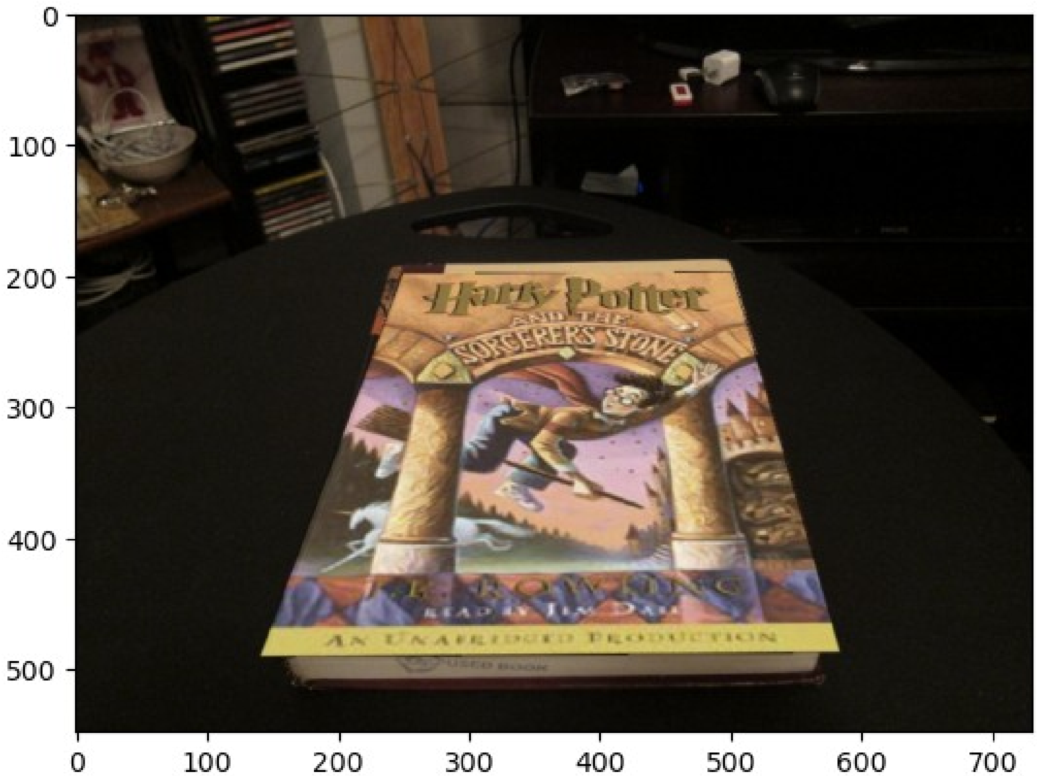
```
=============================================================
=============================================================
max_iters:  200  inlier_tol:  1.0
```



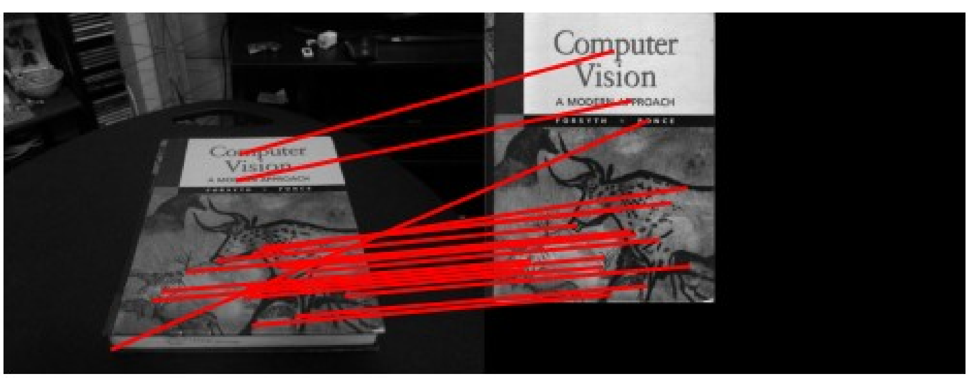


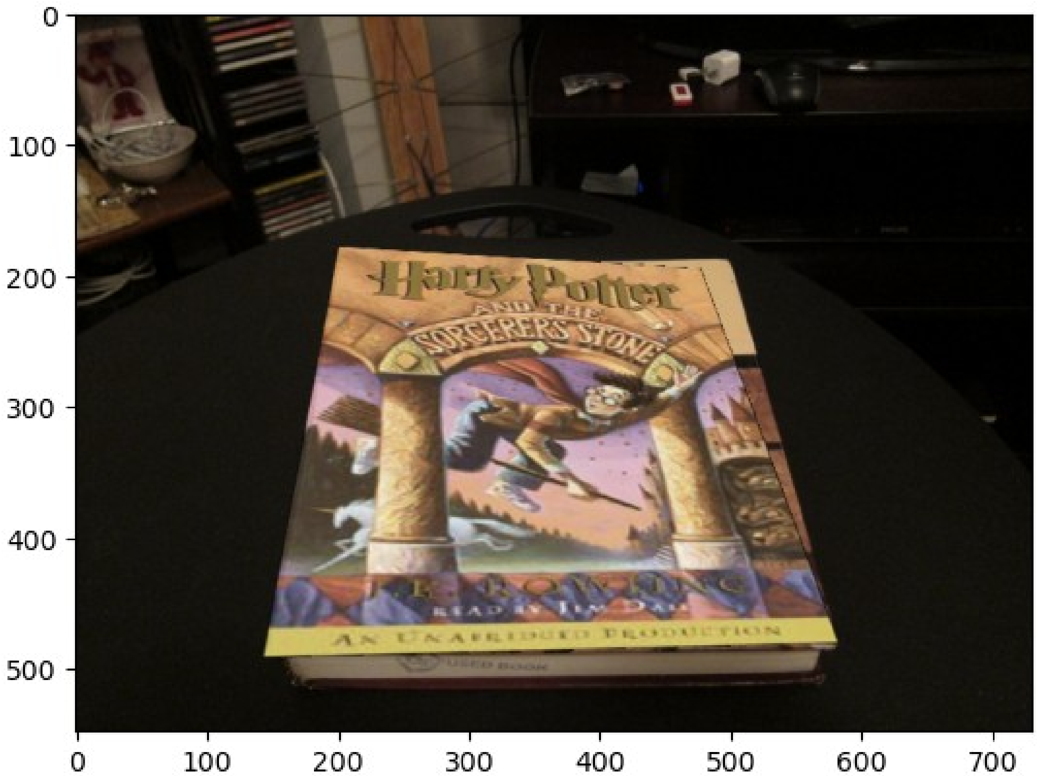

```
=================================================================
=================================================================
max_iters:  200  inlier_tol:  5.0
```



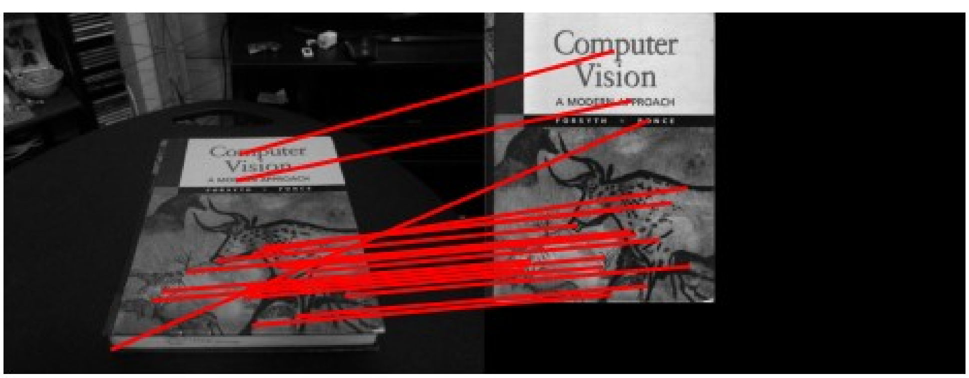


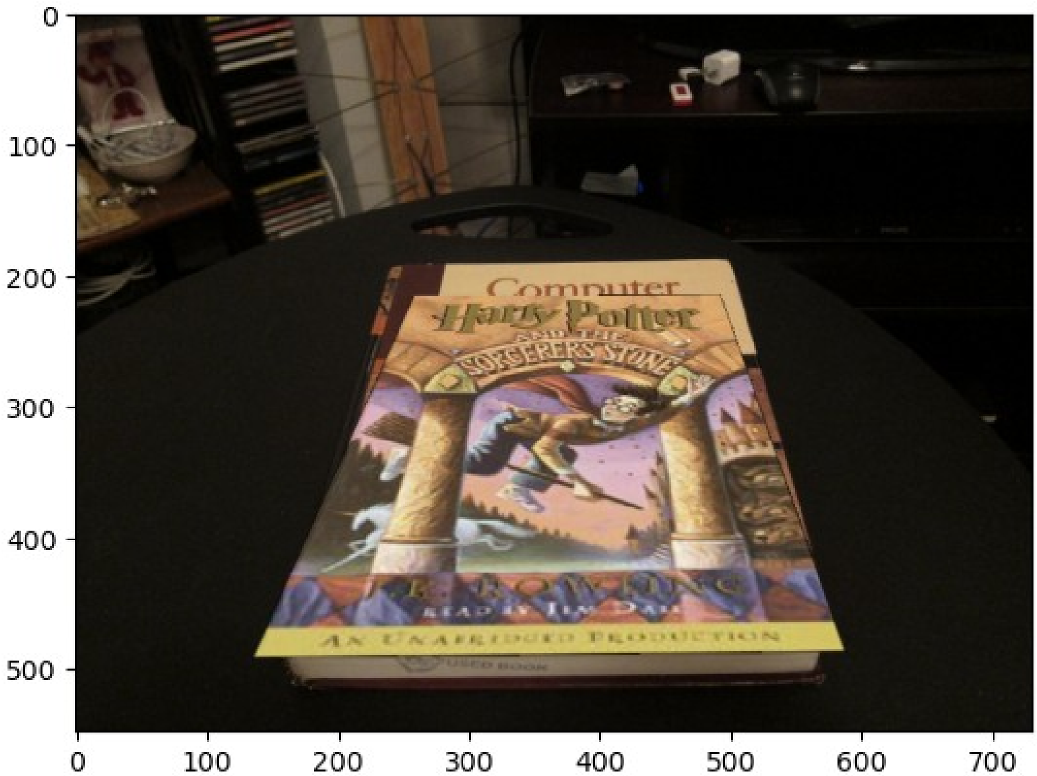

```
==================================================
==================================================
max_iters:  200  inlier_tol:  20.0
```



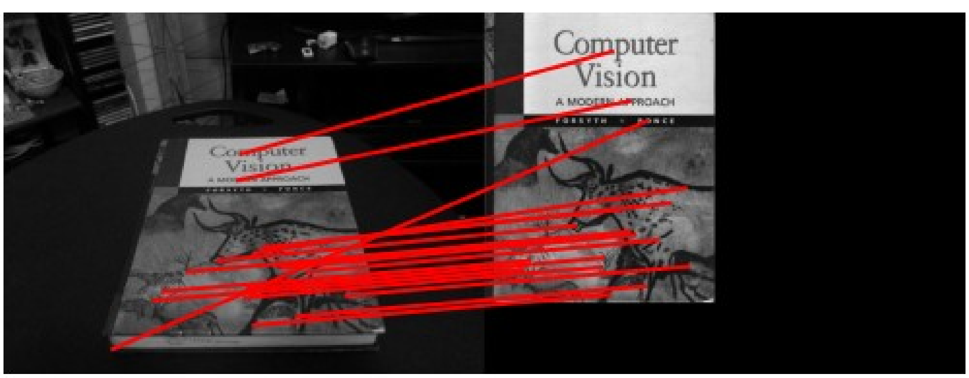


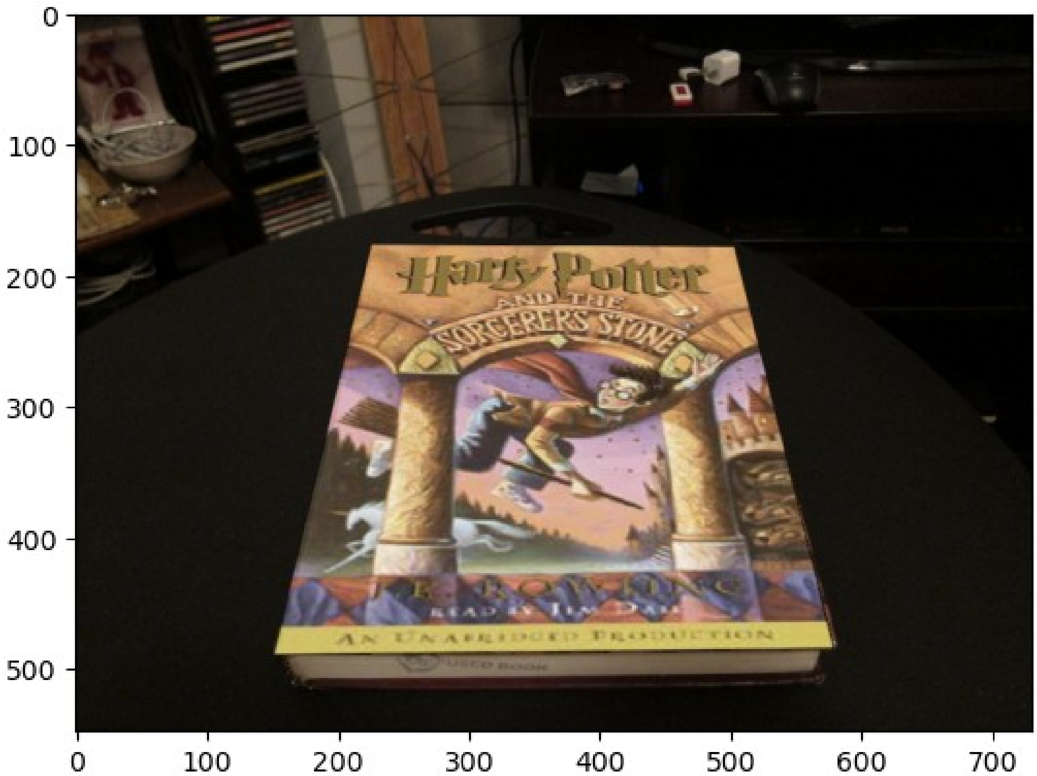

```
============================================================
============================================================
max_iters:  600  inlier_tol:  0.1
```



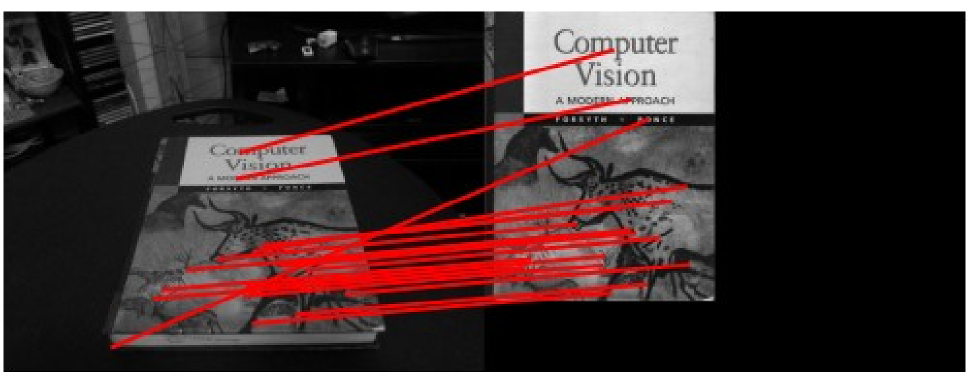


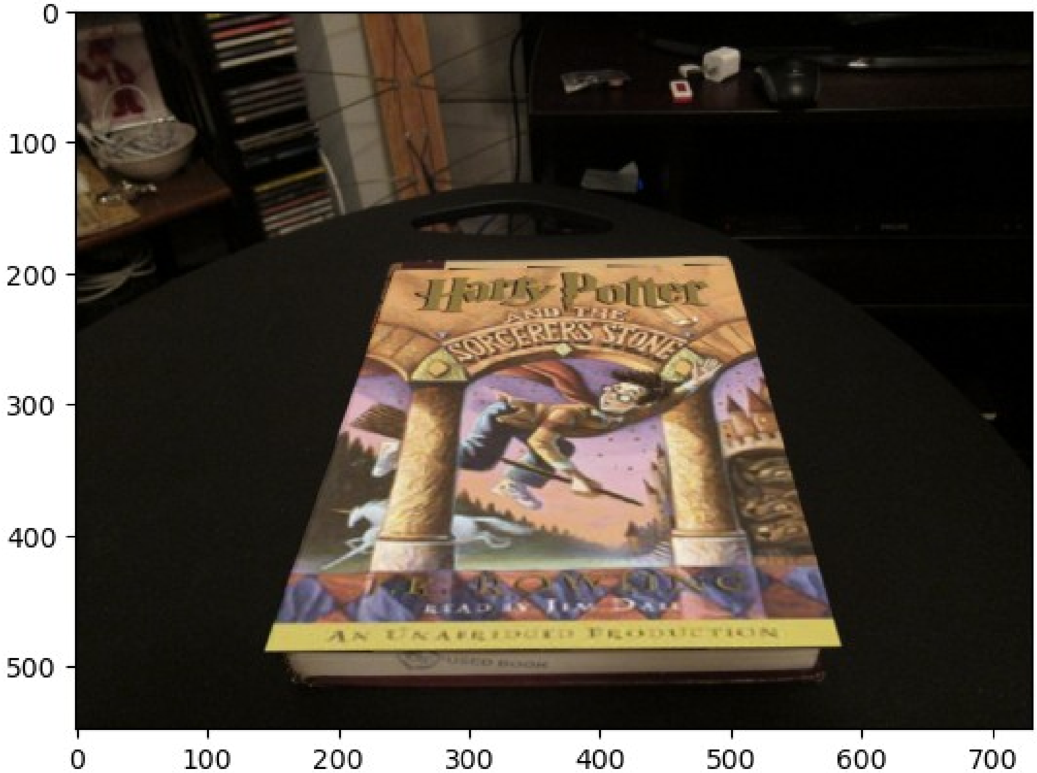

```
=====================================================
=====================================================
max_iters:  600  inlier_tol:  0.5
```



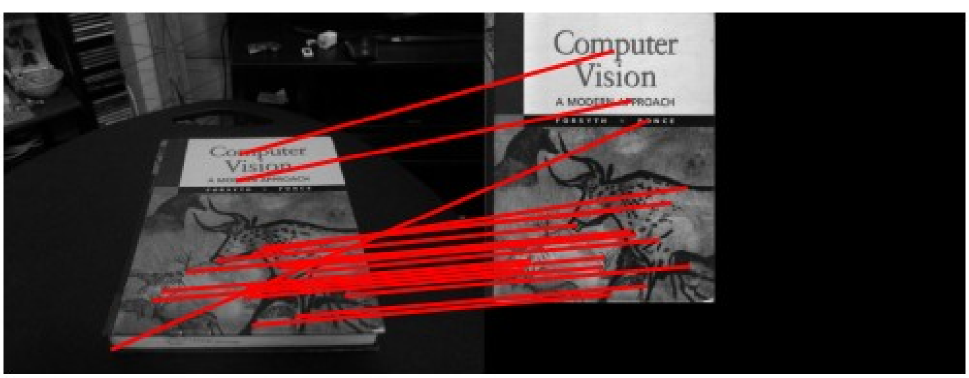


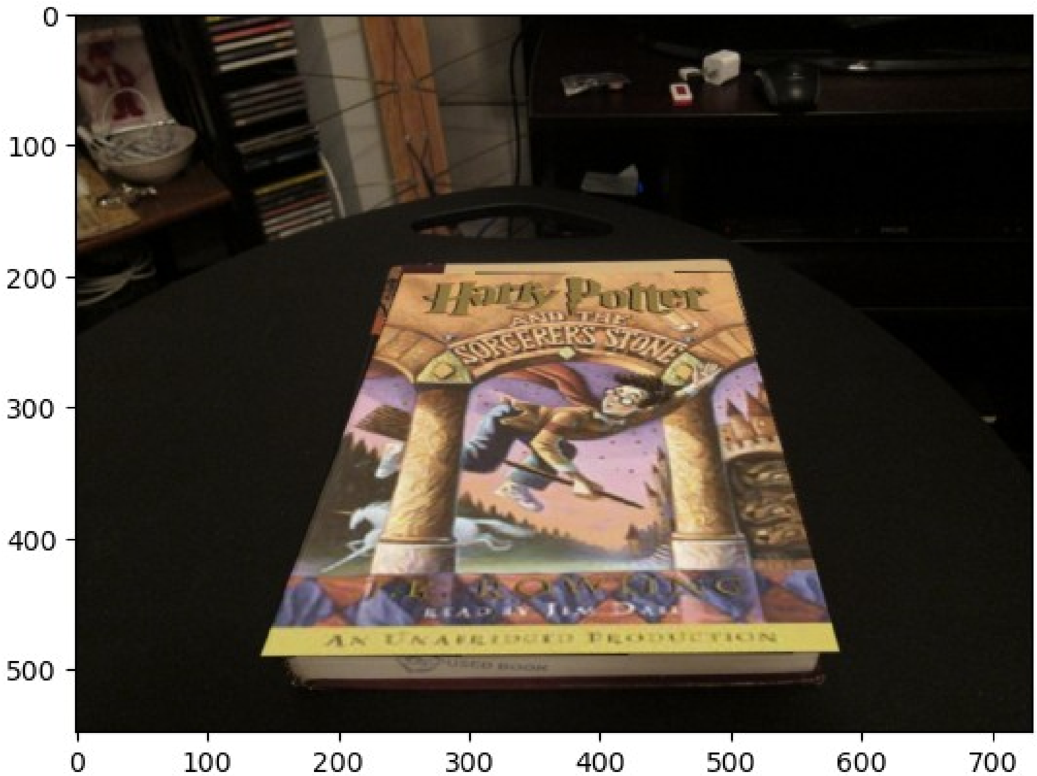

```
============================================================
============================================================
max_iters:  600  inlier_tol:  1.0
```



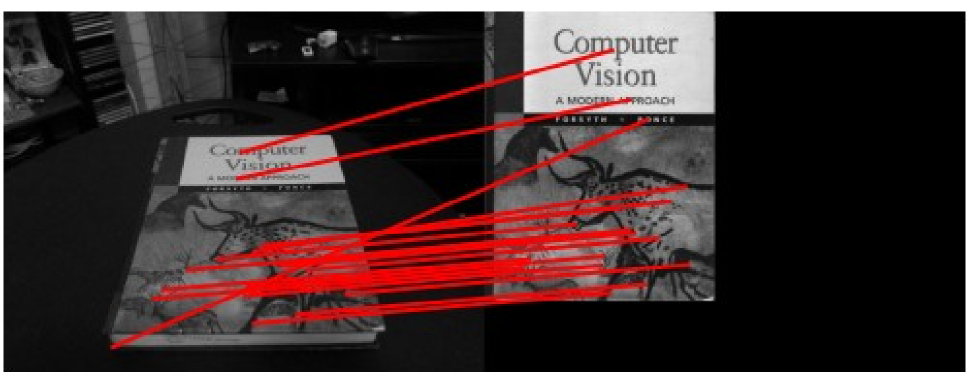


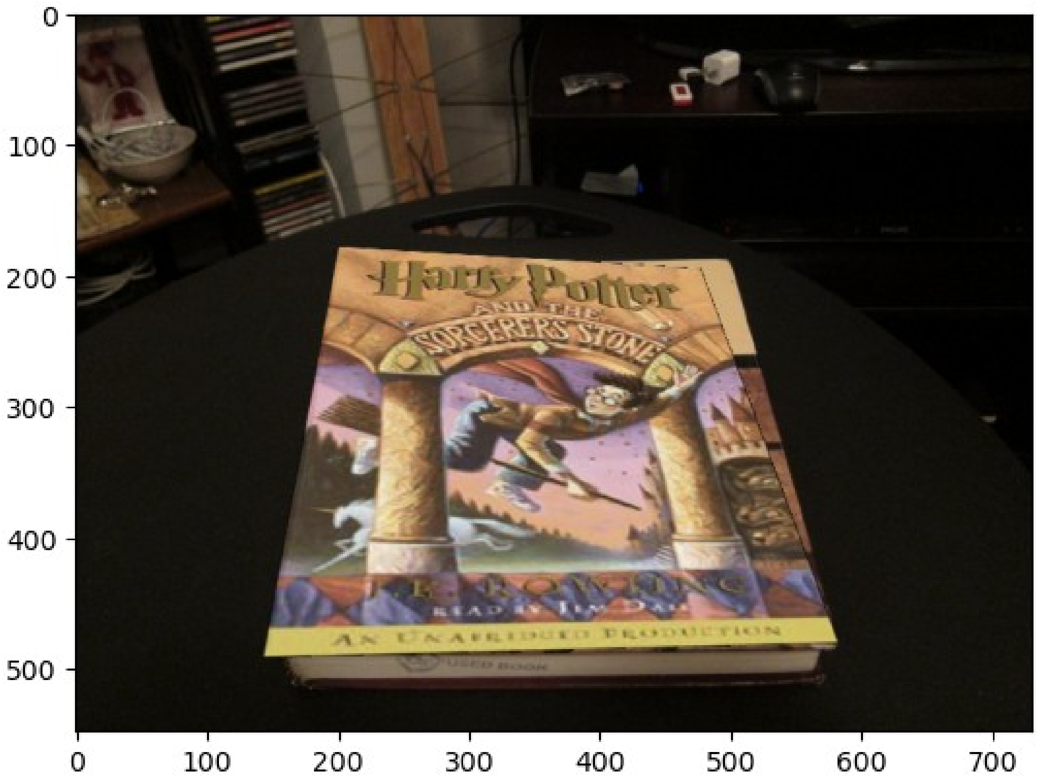

```
==================================================
==================================================
max_iters:  600  inlier_tol:  5.0
```



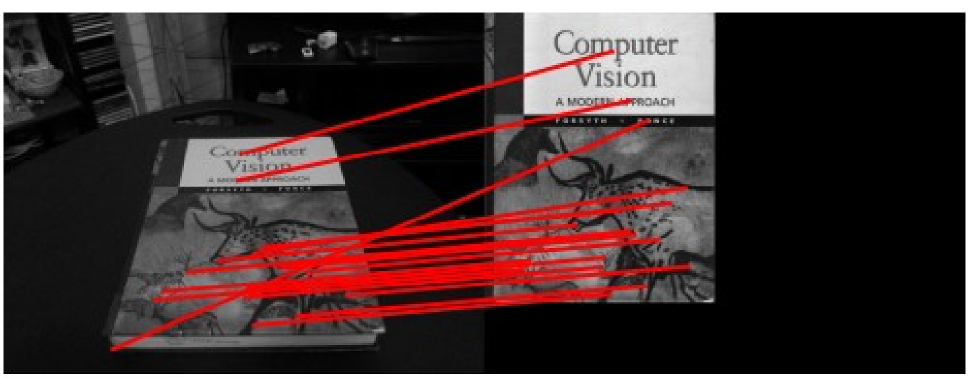


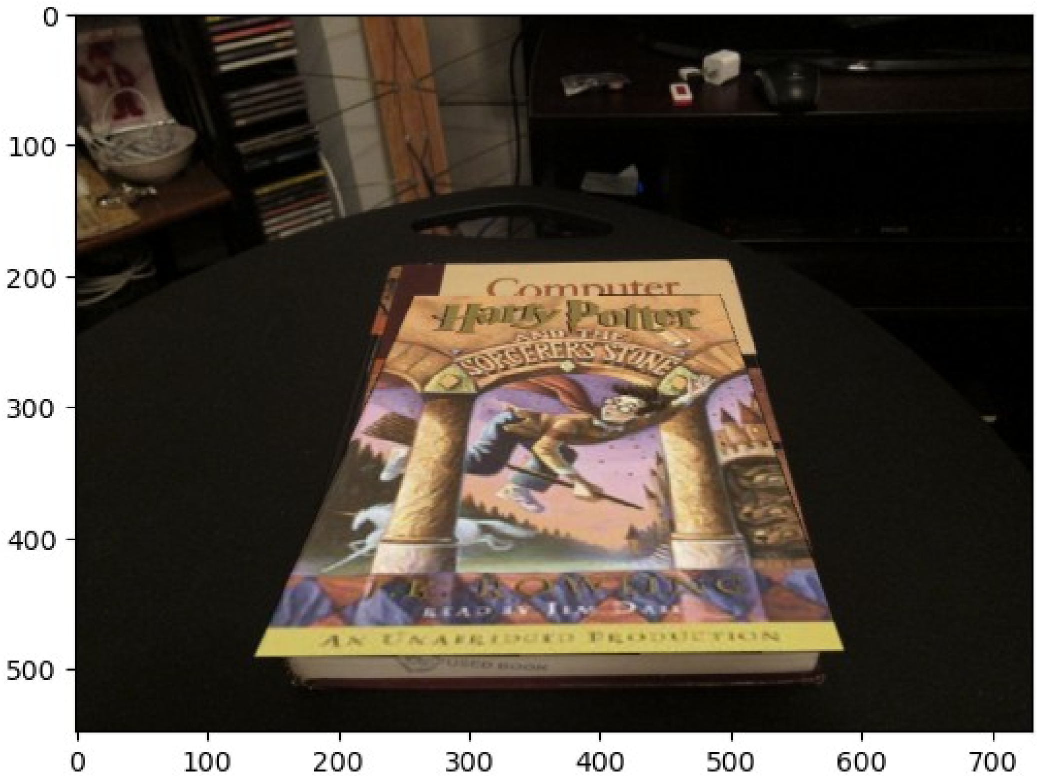

```
==================================================
==================================================
max_iters:  600  inlier_tol:  20.0
```



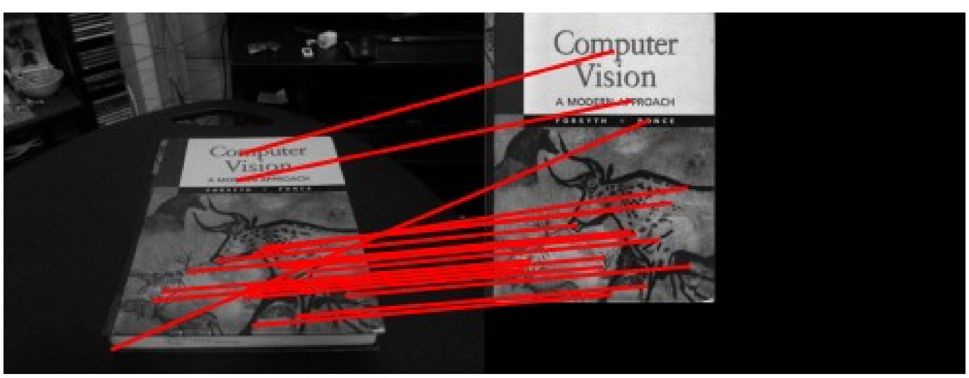


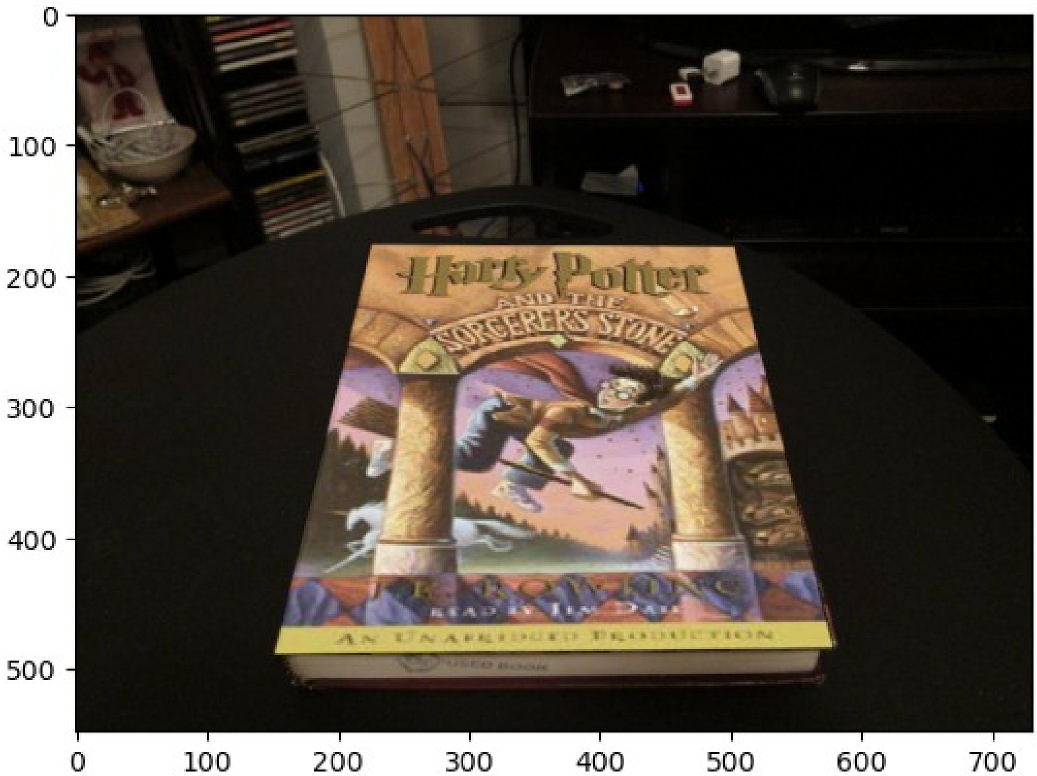

```
==============================================================================
==============================================================================
max_iters:  1800  inlier_tol:  0.1
```



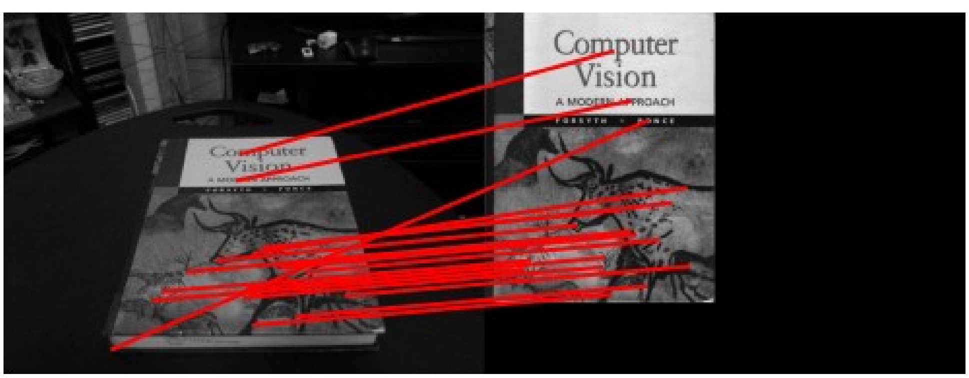


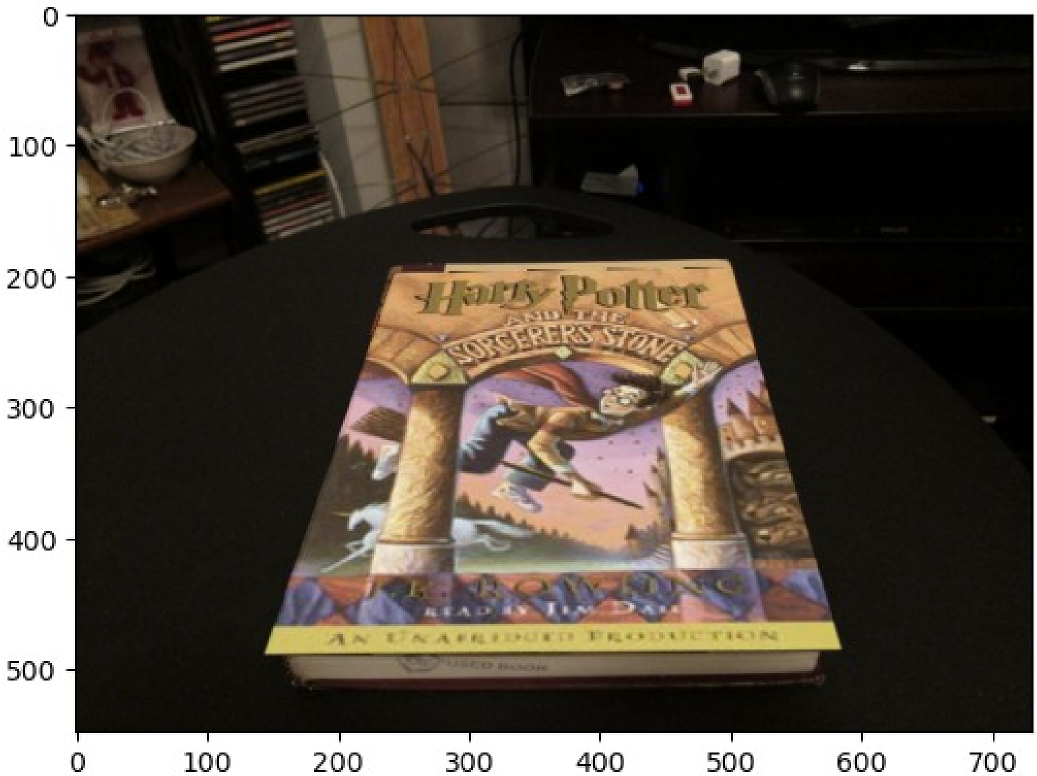
```
============================================================
============================================================
max_iters:  1800  inlier_tol:  0.5
```



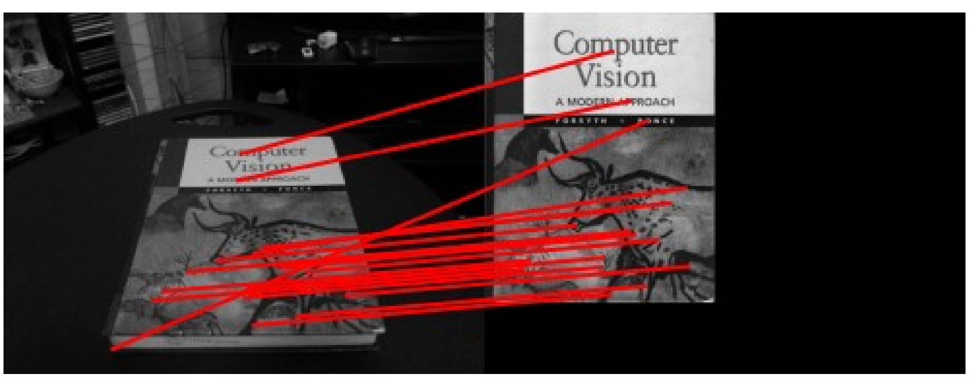


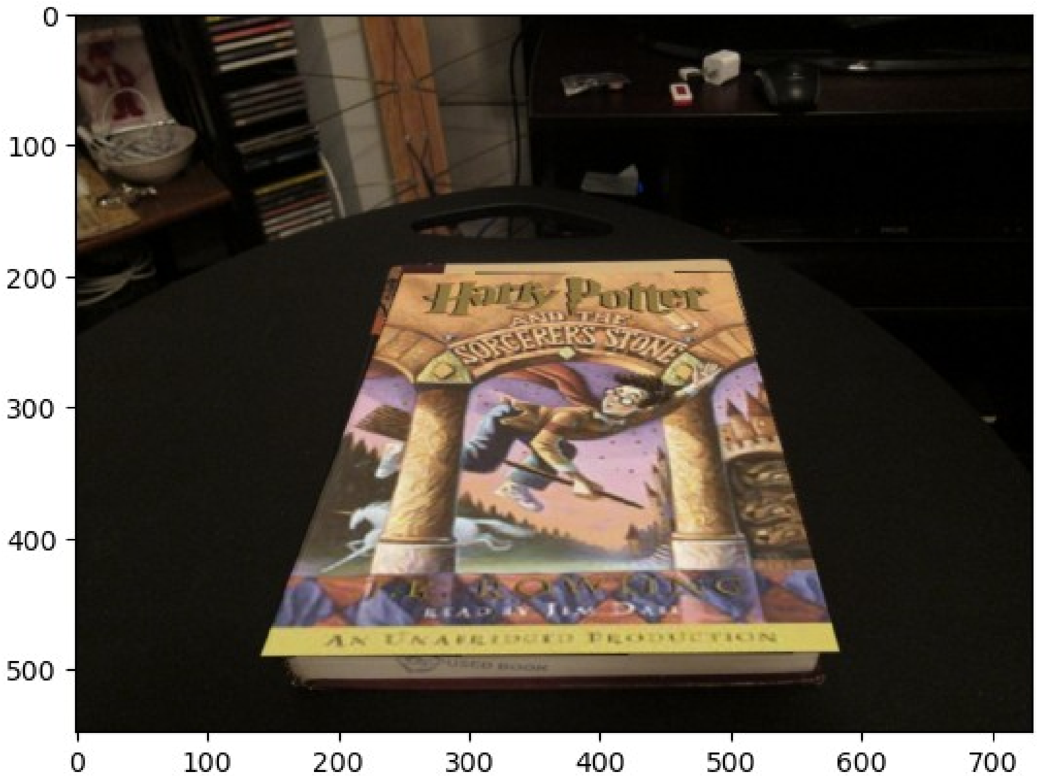

```
============================================================
============================================================
max_iters:  1800  inlier_tol:  1.0
```



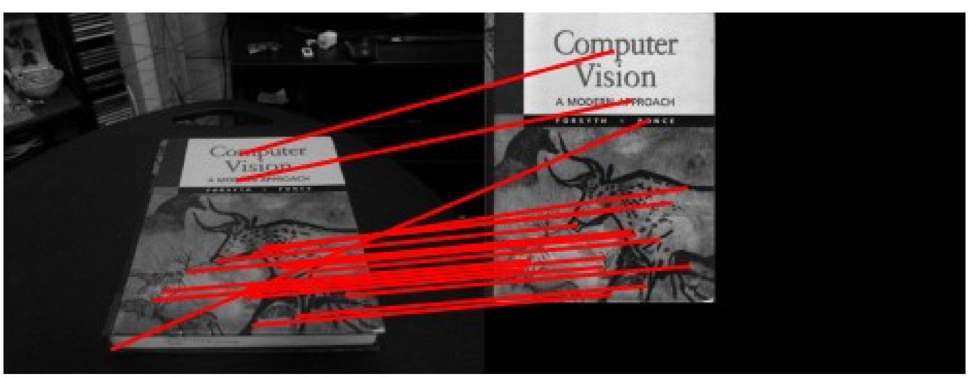


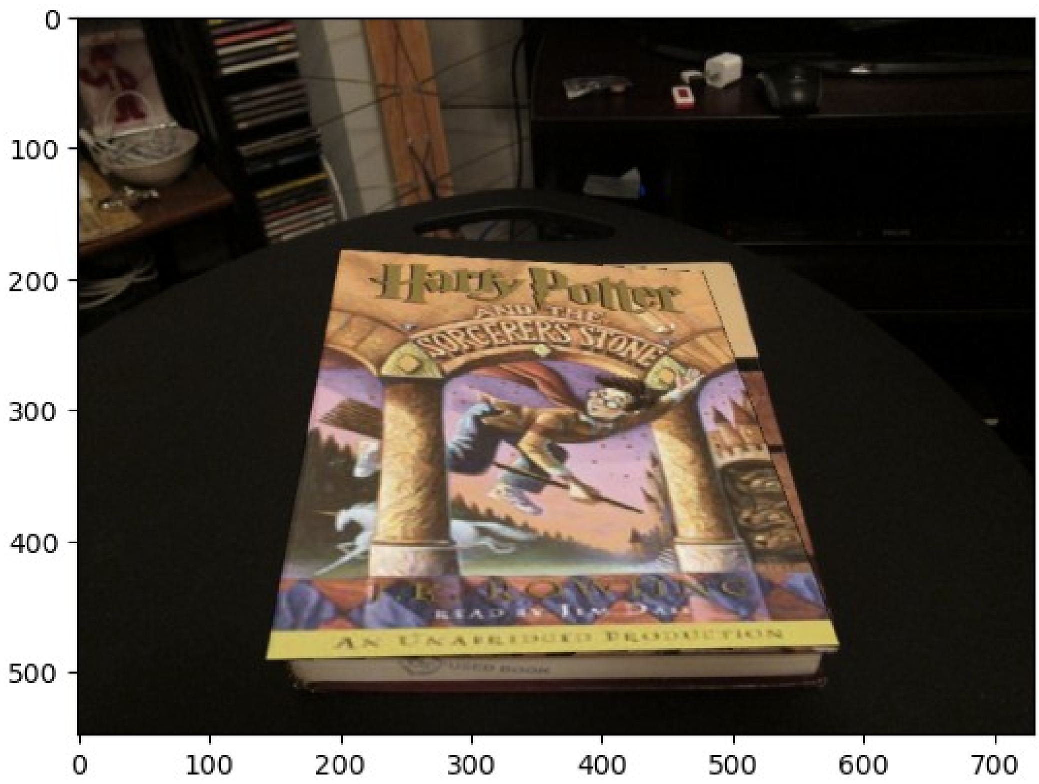

```
==================================================
==================================================
max_iters:  1800  inlier_tol:  5.0
```






```
=============================================================
=============================================================
max_iters:  1800  inlier_tol:  20.0
```






```
==========================================================================
```

Explain the effect of max_iters and inlier_tol:

In RANSAC, max_iters and inlier_tol are key for balancing model accuracy, robustness, and efficiency. max_iters sets the maximum iterations, with higher values increasing the likelihood of finding a good model in noisy data but also raising computation time. inlier_tol defines the distance threshold for classifying points as inliers. A low inlier_tol may exclude valid points, while a high one may wrongly include outliers, reducing model accuracy. Carefully tuning these parameters allows RANSAC to identify a robust model by maximizing inliers while controlling computation and minimizing outlier influence.

Increasing inlier tol increases number of consensus points but may result in bad candidate hypothesis. Decreasing inlier toll results in good enough matches being discarded due to small amount of error causing many points to result as outlier.

Increasing max_iters allows you to check as many possible hypothesis that maximise consensus. too small an you may end up with a bad match

---

Q3 Create a Simple Panorama

Q3.1 Create a panorama (10 points):

Implement the function createPanorama

```
from turtle import right
import numpy as np
import cv2
from skimage.feature import ORB, match_descriptors
from skimage.transform import ProjectiveTransform
from skimage.measure import ransac

def createPanorama(left_im, right_im, ratio, sigma, max_iters, 
inlier_tol):
    """
    Create a panorama augmented reality application by computing a 
homography
    and stitching together a left and right image.

    Input
    -----
    left_im: left image
    right_im: right image
    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

    Returns
    -------
    panorama_im: Stitched together panorama
    # """

    # # Step 1: Detect keypoints and compute descriptors using ORB 
(which is similar to BRIEF + FAST)
    # orb = ORB(n_keypoints=500, fast_threshold=sigma)

    # # Detect and extract features from the left image
    # orb.detect_and_extract(cv2.cvtColor(left_im, 
cv2.COLOR_BGR2GRAY))
    # keypoints1 = orb.keypoints
    # descriptors1 = orb.descriptors

    # # Detect and extract features from the right image
    # orb.detect_and_extract(cv2.cvtColor(right_im, 
cv2.COLOR_BGR2GRAY))
    # keypoints2 = orb.keypoints
```

```
    # descriptors2 = orb.descriptors

    # # Step 2: Match the descriptors using Hamming distance and the
ratio test
    # matches = match_descriptors(descriptors1, descriptors2,
metric='hamming', cross_check=True, max_ratio=ratio)

    matches, locs1, locs2 = matchPics(left_im, right_im, ratio, sigma)
    plotMatches(left_im, right_im, matches, locs1, locs2)

    locs1 = locs1[:, ::-1]
    locs2 = locs2[:, ::-1]

    # Step 3: Select the corresponding points from the matched
descriptors
    src_pts = locs1[matches[:, 0]]
    dst_pts = locs2[matches[:, 1]]

    # swap the x and y columns to follow cv2 convention
    # src_pts = src_pts[:, ::-1]
    # dst_pts = dst_pts[:, ::-1]


    # plotMatches(left_im, right_im, matches, keypoints1, keypoints2)
    # Step 4: Estimate homography using RANSAC
    model, inliers = ransac((dst_pts, src_pts), ProjectiveTransform,
min_samples=4,
                            residual_threshold=inlier_tol,
max_trials=max_iters)
    H2to1, inliers = computeH_ransac(locs1[matches[:,0]],
locs2[matches[:,1]], max_iters, inlier_tol)
    # H2to1, inliers = computeH_ransac(corners2, corners, max_iters,
inlier_tol)

    # TODO: Overlay using compositeH to return composite_img
    # composite_img = compositeH(H2to1, hp_cover, cv_desk)

    # Step 5: Warp the right image into the left image's perspective
    # Get the size of the left image and the right image
    height1, width1 = left_im.shape[:2]
    height2, width2 = right_im.shape[:2]

    # Calculate the canvas size to fit both images
    panorama_width = width1 + width2
    panorama_height = max(height1, height2)

    # Warp the right image using the computed homography
    warped_right_im = cv2.warpPerspective(right_im, H2to1,
(panorama_width, panorama_height))
```

```
    # Step 6: Create the panorama by placing the left image and
blending the right
    panorama_im = np.zeros((panorama_height, panorama_width, 3),
dtype=np.uint8)

    # Place the left image in the panorama
    panorama_im[:height1, :width1] = left_im

    # Blend the warped right image into the panorama
    # You can blend by averaging or by using a more sophisticated
blending technique
    mask = (warped_right_im > 0)  # Non-black pixels are part of the
right image
    panorama_im[mask] = warped_right_im[mask]

    return panorama_im.astype(np.uint8)
```

Visualize Panorama

Make sure to use **your own images** and **include them as well as the result** in the report.

```
left_im_path = os.path.join(DATA_DIR, 'pano_left.jpg')
left_im = skimage.io.imread(left_im_path)
right_im_path = os.path.join(DATA_DIR, 'pano_right.jpg')
right_im = skimage.io.imread(right_im_path)

# Feel free to adjust as needed
ratio = 0.7
sigma = 0.1
max_iters = 60
inlier_tol = 1.0

panorama_im = createPanorama(left_im, right_im, ratio, sigma,
max_iters, inlier_tol)

plt.imshow(panorama_im)
plt.axis('off')
plt.show()
```

```
/var/folders/43/5971cqb109bfv3wb3b47x9tc0000gn/T/
ipykernel_34345/3301853510.py:58: 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
performing this operation. (Deprecated NumPy 1.25.)
  return 1 if img[int(center[0]+row1)][int(center[1]+col1)] <
img[int(center[0]+row2)][int(center[1]+col2)] else 0
/var/folders/43/5971cqb109bfv3wb3b47x9tc0000gn/T/ipykernel_34345/33018
53510.py:35: FutureWarning: `plot_matches` is deprecated since version
0.23 and will be removed in version 0.25. Use
```

```
`skimage.feature.plot_matched_features` instead.
  skimage.feature.plot_matches(ax,img1,img2,locs1,locs2,
```

```
left_im_path = os.path.join(DATA_DIR, 'src_left.jpg')
left_im = skimage.io.imread(left_im_path)
right_im_path = os.path.join(DATA_DIR, 'src_right.jpg')
right_im = skimage.io.imread(right_im_path)

# Feel free to adjust as needed
ratio = 0.7
sigma = 0.1
max_iters = 60
inlier_tol = 1.0

panorama_im = createPanorama(left_im, right_im, ratio, sigma,
max_iters, inlier_tol)

plt.imshow(panorama_im)
plt.axis('off')
plt.show()
```

```
/var/folders/43/5971cqb109bfv3wb3b47x9tc0000gn/T/
ipykernel_34345/3301853510.py:58: 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
```

```
performing this operation. (Deprecated NumPy 1.25.)
  return 1 if img[int(center[0]+row1)][int(center[1]+col1)] <
img[int(center[0]+row2)][int(center[1]+col2)] else 0
/var/folders/43/5971cqb109bfv3wb3b47x9tc0000gn/T/ipykernel_34345/33018
53510.py:35: FutureWarning: `plot_matches` is deprecated since version
0.23 and will be removed in version 0.25. Use
`skimage.feature.plot_matched_features` instead.
  skimage.feature.plot_matches(ax,img1,img2,locs1,locs2,
```