16-820 Advanced Computer Vision: Homework 1 (Fall 2024)
Augmented Reality with Planar Homographies

Ryan Wu (Andrew ID: weihuanw)

Monday September 16, 2024

Q1.1
Prove that there exists a homography H that satisfies equation 1 given two 3×4 camera projection matrices P_1 and P_2 corresponding to the two cameras and a plane π.

Given Figure 1 and the two 3×4 camera projection matrices P_1 and P_2, the two cameras' projection can be represented as:

$$x = P_1 x_\pi$$

$$x' = P_2 x_\pi$$

By relating and substituting x' into x, we get:

$$x = P_1 P_2^{-1} x' = x_1 \equiv H x_2$$

From the above, we can define a homography H that satisfies $x_1 \equiv H x_2$ as:

$$H = P_1 P_2^{-1}$$

Q1.2

1. How many degrees of freedom does h have?

Since there exist a H in $x_1 \equiv Hx_2$, the full matrix representation can be expressed as:

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

After rearranging the terms and writing as a linear equation, h can be written as:

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

Lastly, with the scale factor normalization, h will have 8 degrees of freedom.

2. How many point pairs are required to solve h?

Since h has 8 degrees of freedom, 4 point pairs are required to solve h.

3. Derive A_i.

Recall, the full matrix representation can be expressed as:

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

Rearranging the terms and writing as linear equations, the full matrix can also be represented as:

$$x_1 = h_{11}x_2 + h_{12}y_2 + h_{13}$$

$$y_1 = h_{21}x_2 + h_{22}y_2 + h_{23}$$

$$1 = h_{31}x_2 + h_{32}y_2 + h_{33}$$

Further rearranging the terms as a system of linear equations:

$$x_1(h_{31}x_2 + h_{32}y_2 + h_{33}) = h_{11}x_2 + h_{12}y_2 + h_{13}$$
$$y_1(h_{31}x_2 + h_{32}y_2 + h_{33}) = h_{21}x_2 + h_{22}y_2 + h_{23}$$

Solving the system of equation with the condition $A_i h = 0$, we get the following solution:

$$\begin{bmatrix} -x_2 & -y_2 & -1 & 0 & 0 & 0 & x_1x_2 & x_1y_2 & x_1 \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & y_1x_2 & y_1y_2 & y_1 \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = 0$$

Thus,

$$\mathbf{A}_i = \begin{bmatrix} -x_2 & -y_2 & -1 & 0 & 0 & 0 & x_1x_2 & x_1y_2 & x_1 \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & y_1x_2 & y_1y_2 & y_1 \end{bmatrix}$$

4. When solving Ah = 0, in essence you're trying to find the h that exists in the null space of A. What that means is that there would be some non-trivial solution for h such that that product Ah turns out to be 0. What will be a 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 A^TA)?

The trivial solution for h would be h = 0. In our case, matrix A_i is a 2×9 with a possible full rank of 2. However, the two system of equations is not linearly independent, which indicates A_i not being full rank. Rank deficiency will lead to zero singular values and zero eigenvalues of A^TA for matrix A_i.

Q1.4.1: Homography under rotation
Prove that there exists a homography H that satisfies $x_1 \equiv Hx_2$, given two cameras separated by a pure rotation.

With the given parameters, the homography projection equation of camera 1 and 2 in the camera's local coordinates can be represented as:

$$x_1 = K_1[I \mid 0]X$$

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

Next, we project both cameras' projections to the world coordinate:

$$x_1 = K_1 X_{\text{world}}$$

$$x_2 = K_2 R X_{\text{world}}$$

By relating and substituting x_2 into x_1, we get:

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

From the above, we can define a homography H that satisfies $x_1 \equiv Hx_2$ as:

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

Q1.4.2: Understanding homographies under rotation
Show that H^2 is the homography corresponding to a rotation of 2θ.

Recall, the homography H for pure rotation can be represented as:

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

The inverse rotation matrix in the z-axis is:

$$R_z^{-1}(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

In the case of H^2, the homography becomes:

$$H^2 = (K_1 R_z^{-1}(\theta) K_2^{-1})(K_1 R_z^{-1}(\theta) K_2^{-1})$$

This can be further simplified to:

$$H^2 = K_1 R_z^{-1}(\theta) R_z^{-1}(\theta) K_2^{-1}$$

Note that:

$$R_z^{-1}(\theta) R_z^{-1}(\theta) = R_z^{-1}(2\theta)$$

Thus, we can prove that H^2 is the homography corresponding to a rotation of 2θ.

$$H^2 = K_1 R_z^{-1}(2\theta) K_2^{-1}$$

Q1.4.3: Limitations of the planar homography
Why is the planar homography not completely sufficient to map any arbitrary scene image to another viewpoint?

Planar homography is not completely sufficient to map any arbitrary scene image to another viewpoint because the assumption that all points lie on the same plane is not always true. There may be varying depths and 3D structures.

Q1.4.4: Behavior of lines under perspective projections
Verify algebraically that this is the case, i.e., verify that the projection P in x = PX preserves lines.

A line in 3D can be represented as:

$$\mathbf{X}(\lambda) = \mathbf{X}_0 + \lambda \mathbf{d_{3D}}$$

With proper subsitution, the projection of the 3D line in 2D space can be represented as:

$$\mathbf{x}(\lambda) = P\mathbf{X}(\lambda) = P(\mathbf{X}_0 + \lambda \mathbf{d_{3D}}) = P\mathbf{X}_0 + \lambda P \mathbf{d_{3D}}$$

From the above equation, we can further show that that the projection P in x = PX preserves lines with:

$$\mathbf{x}(\lambda) = \mathbf{x}_0 + \lambda \mathbf{d}_{2D}$$

where:

$$\mathbf{x}_0 = P\mathbf{X}_0$$

$$\mathbf{d}_{2D} = P\mathbf{d}_{3D}$$

Q2.1.1: FAST Detector
How is the FAST detector different from the Harris corner detector that you've seen in the lectures? Can you comment on its computational performance compared to the Harris corner detector?

The main difference between the FAST (features accelerated segment test) detector and the Harris corner detector is the method they use in determining corners. The FAST detector utilizes pixel intensity/intensity threshold as its corner-detecting logic. A candidate pixel will have a surrounding pixel ring, usually a circle of 16 pixels (Bresenham circle) for corner classification. Since the FAST detector uses a binary decision, it is more computationally efficient but less robust.

On the other hand, the Harris corner detector relies on matrix construction and computing eigenvalues for corner detection. For example, when both eigenvalues are large, the algorithm will classify it as a corner. The Harris corner detector method is more math-intensive and can be more robust but less computationally efficient.

When comparing computational performance between the two algorithms, the FAST detector is significantly faster than the Harris corner detector. The Harris corner detector will be better suited for applications that require high accuracy. The FAST detector is more beneficial for real-time implementations.

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

The main difference between the BRIEF (binary robust independent elementary features) and the filter banks is the underlying technique used in feature detection. The BRIEF descriptor uses pixel intensity differences to represent an image patch as a binary string. It yields higher recognition rates and is more computationally efficient. However, the BRIEF descriptor is less accurate when dealing with rotation and scaling.

On the other hand, filter banks apply linear filters (Gaussian, Laplacian of Gaussian, etc.) to extract unique features such as edges, gradients, orientations, etc. Filter banks are more robust and computationally heavy, better suited for images with more complex features or transformations. Any one of these filter banks can be used as a descriptor, but one must consider the trade-off between robustness and implementation time.

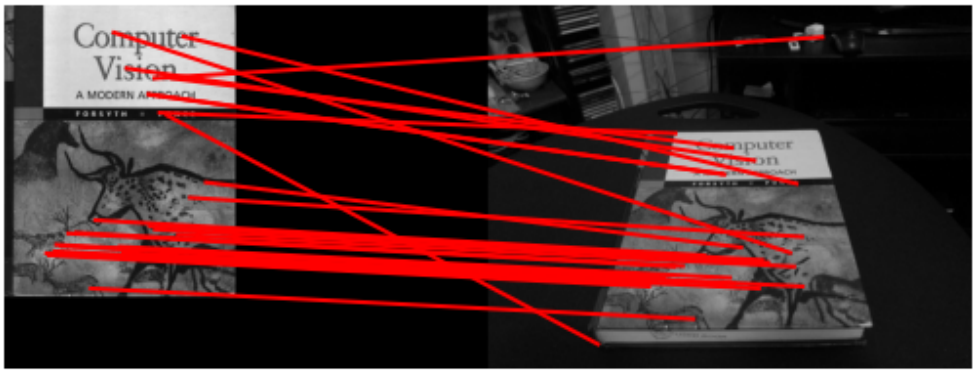
Q2.1.3: Matching Methods
Please search online to learn about Hamming distance and Nearest Neighbor, and describe how they can be used to match interest points with BRIEF descriptors. What benefits does the Hamming distance have over a more conventional Euclidean distance measure in our setting?

When matching interest points with BRIEF descriptors, the Hamming distance is more computationally and memory efficient compared to the conventional Euclidean distance. The faster implementation is due to the XOR operation in calculating Hamming distance compliments the binary string data used in BRIEF descriptors.

Q2.1.4: Feature Matching
Please implement a function in matchPics.py. Please include the code snippet and your best output of displayMatch in your write-up.

Here is the code snippet:

```
import numpy as np
import cv2
import skimage.color
from helper import briefMatch
from helper import computeBrief
from helper import corner_detection

# Q2.1.4

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

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

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

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

        # TODO: Convert Images to GrayScale
        'coverting-using-cv2'
        # I1_grayscale = cv2.cvtColor(I1, cv2.COLOR_BGR2GRAY)
        # I2_grayscale = cv2.cvtColor(I2, cv2.COLOR_BGR2GRAY)

        'coverting-using-skimage'
        I1_grayscale = skimage.color.rgb2gray(I1)
        I2_grayscale = skimage.color.rgb2gray(I2)

        # TODO: Detect Features in Both Images
        locs1 = corner_detection(I1_grayscale, sigma)
        locs2 = corner_detection(I2_grayscale, sigma)

        # TODO: Obtain descriptors for the computed feature locations
        desc1, locs1 = computeBrief(I1_grayscale, locs1)
        desc2, locs2 = computeBrief(I2_grayscale, locs2)

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

        return matches, locs1, locs2
```

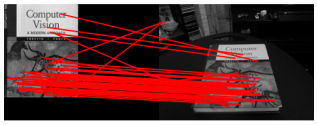


The default output visualization of my feature matching (displayMatch.py) implementation.

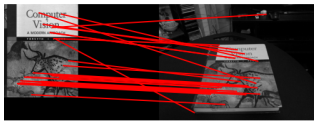
Q2.1.5: Feature Matching and Parameter Tuning
Conduct a small ablation study by running displayMatch.py with various sigma and ratio values. Include the figures displaying the matched features with various parameters in your write-up, and explain the effect of these two parameters respectively.

A small ablatioin study was conducted and the results are the shown below.

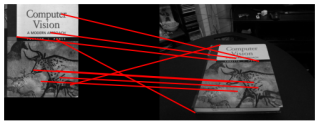
Varying sigma value with constant ratio value:



Sigma 0.1, ratio 0.7

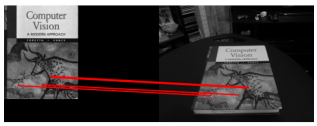


Sigma 0.15, ratio 0.7

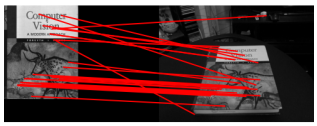


Sigma 0.20, ratio 0.7

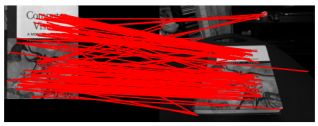
Varying ratio value with constant sigma value:



Sigma 0.15, ratio 0.5



Sigma 0.15, ratio 0.7

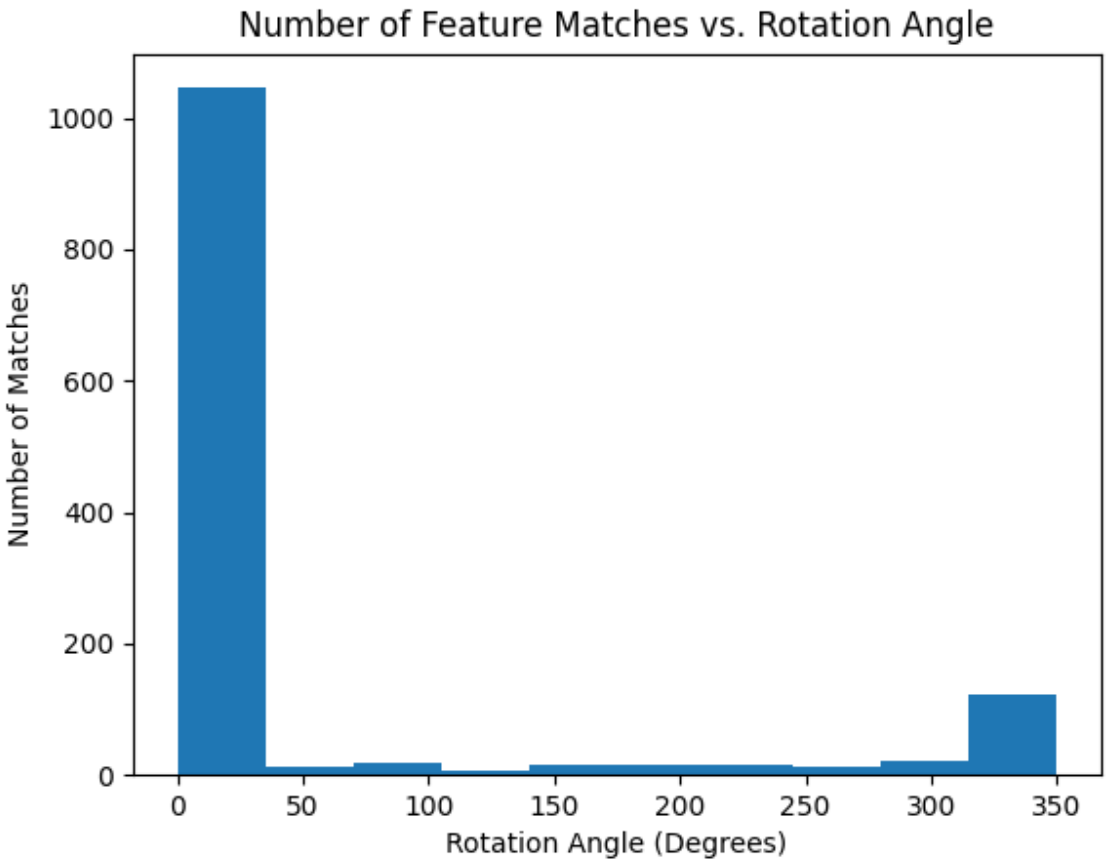


Sigma 0.15, ratio 0.9

From the study, we can observe that higher sigma values generates fewer feature matches. This is due to the higher sigma value applying more blurs during Gaussian smoothing, which can lead to less feature matches but more robustness. On the other hand, higher ratio values generates more feature matches. This is because higher ratio values are less strict in the differences between the best and the second-best match distance, in terms, decreases overall robustness.

Q2.1.6: BRIEF and Rotations
Write a script briefRotTest.py that: Visualize the histogram and the feature matching result at three sufficiently different orientations and include them in your write-up. Explain why you think the BRIEF descriptor behaves this way. Please include the code snippet in your write-up.

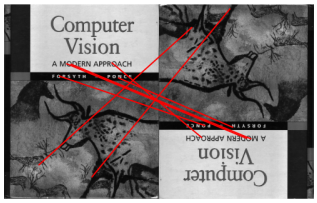
With the default parameters, the match distribution histogram is shown below:



The match distribution histogram.

The visualization of feature matching for varying rotation angles is shown below:

0 degree rotation



180 degree rotation

270 degree rotation

From the histogram and the visualization results, we can observe that with increased rotation, the number of matches dramatically decreased. This is because the BRIEF descriptor is not designed to be rotationally invariant. As we rotate the images, the keypoints computed by the BRIEF descriptors in the rotated images does not align well.

Here is the code snippet:

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

#Q2.1.6

def rotTest(opts):

    # TODO: Read the image and convert to grayscale, if necessary
    img = cv2.imread('../data/cv_cover.jpg')
    img_grayscale = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Initialize histogram data (rotation angle & number of matches)
    angles = []
    matches = []

    for i in range(36):

        # TODO: Rotate Image
        img_rotated = scipy.ndimage.rotate(img, i*10)

        # TODO: Compute features, descriptors and Match features
        matches_, locs1, locs2 = matchPics(img, img_rotated, opts)

        # TODO: Update histogram
        angles.append(i*10)
        matches.append(len(matches_))

    pass

    # TODO: Display histogram
    plt.hist(angles, weights=matches)
    plt.xlabel('Rotation Angle (Degrees)')
    plt.ylabel('Number of Matches')
    plt.title('Number of Feature Matches vs. Rotation Angle')
    plt.show()

    # visualize feature match results at 0, 90, 180, 270 degrees
    for angle in [0, 90, 180, 270]:
        img_rotated = scipy.ndimage.rotate(img, angle)
        matches_, locs1, locs2 = matchPics(img, img_rotated, opts)
        plotMatches(img, img_rotated, matches_, locs1, locs2)
        plt.show()

    return

if __name__ == "__main__":

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

Q2.1.7 (Extra Credit): Improving Performance
1. As we have seen, BRIEF is not rotation invariant. Design a simple fix to solve this problem using the tools you have developed so far (think back to edge detection and/or Harris corner's covariance matrix). You are not allowed to use any additional OpenCV or Scikit-Image functions. Include the code in your PDF, and explain your design decisions and how you selected any parameters that you use. Demonstrate the effectiveness of your algorithm on image pairs related by large rotation.

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

Did not attempt.

Q2.2.1: Computing the Homography
Write a function computeH in planarH.py that estimates the planar homography from a set of matched point pairs. Please include the code snippet in your write-up.

Here is the code snippet:

```
import numpy as np
import cv2


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

    # Number of points
    N = x1.shape[0]

    # Initialize A matrix
    A = np.zeros((2*N, 9))

    # Construct A matrix
    for i in range(N):
        x, y = x1[i]
        X, Y = 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]

    # Solve using SVD
    _, _, V = np.linalg.svd(A)

    # Homography
    H2to1 = V[-1, :].reshape(3, 3)


    return H2to1
```

Q2.2.2: Homography Normalization
Implement the function computeH norm. Please include the code snippet in your write-up.

Here is the code snippet:

```
def computeH_norm(x1, x2):
    #Q2.2.2
    # TODO: Compute the centroid of the points

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

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

    # TODO: Normalize the points so that the largest distance from the origin is
        equal to sqrt(2)
    x1_dist = np.mean(np.sqrt(np.sum(x1_shifted**2, axis=1)))
    x2_dist = np.mean(np.sqrt(np.sum(x2_shifted**2, axis=1)))

    # Scale factors
    x1_scale = np.sqrt(2) / x1_dist
    x2_scale = np.sqrt(2) / x2_dist

    # TODO: Similarity transform 1
    T1 = np.array([[x1_scale, 0, -x1_scale*x1_centroid[0]], [0, x1_scale, -x1_scale*
        x1_centroid[1]], [0, 0, 1]])

    # TODO: Similarity transform 2
    T2 = np.array([[x2_scale, 0, -x2_scale*x2_centroid[0]], [0, x2_scale, -x2_scale*
        x2_centroid[1]], [0, 0, 1]])

    # TODO: Compute homography

    # Convert to homogeneous coordinates
    x1_homogeneous = np.hstack((x1, np.ones((x1.shape[0], 1))))
    x2_homogeneous = np.hstack((x2, np.ones((x2.shape[0], 1))))

    # Normalize points
    x1_normalized = (T1 @ x1_homogeneous.T).T[:, :2]
    x2_normalized = (T2 @ x2_homogeneous.T).T[:, :2]

    # Compute homography between normalized points
    H2to1_normalized = computeH(x1_normalized, x2_normalized)

    # TODO: Denormalization
    H2to1 = np.linalg.inv(T1) @ H2to1_normalized @ T2


    return H2to1
```

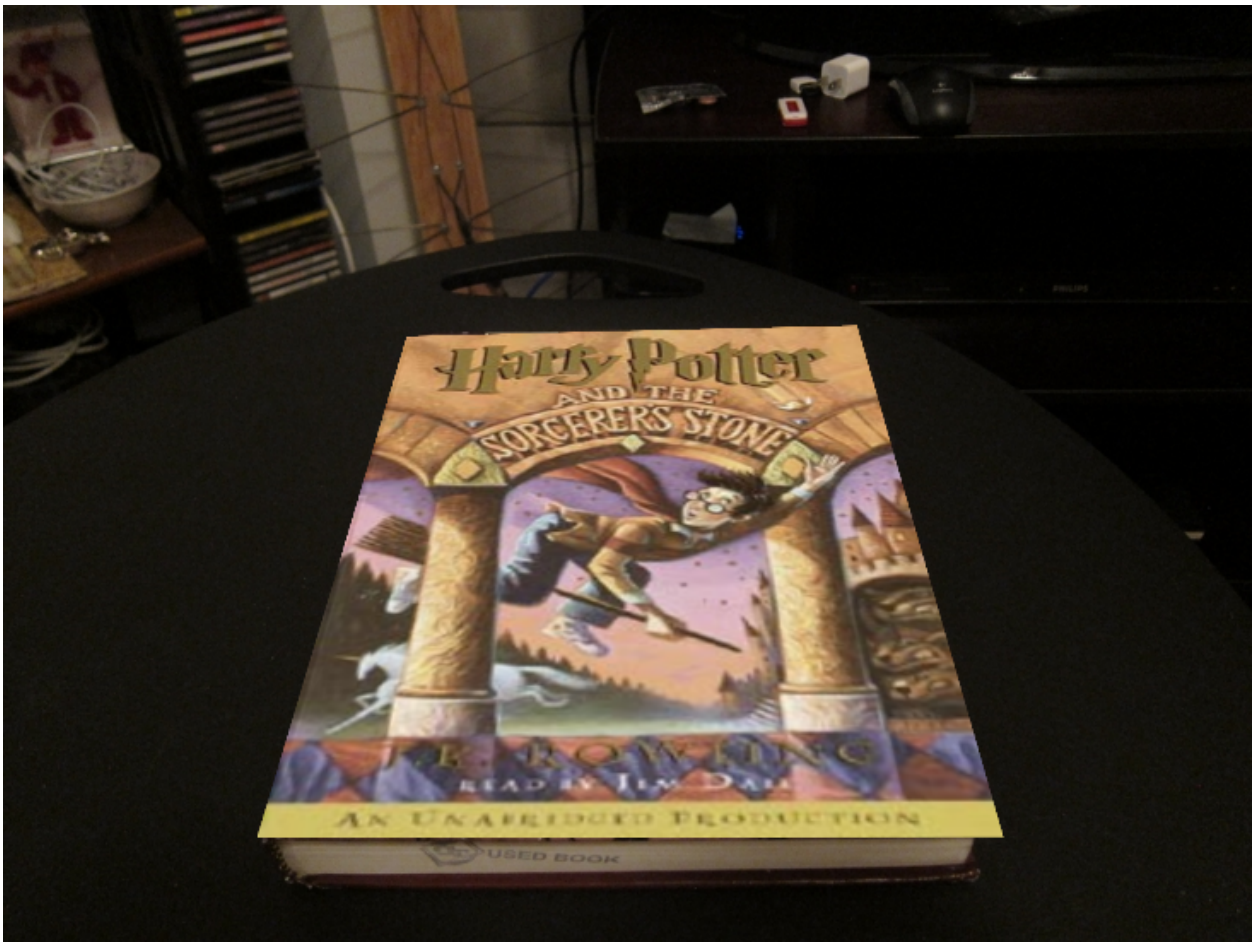
Q2.2.3: Implement RANSAC
Please include the code snippet in your write-up.

Here is the code snippet:

```
def computeH_ransac(locs1, locs2, opts):
    #Q2.2.3
    #Compute the best fitting homography given a list of matching points
    max_iters = opts.max_iters  # the number of iterations to run RANSAC for
    inlier_tol = opts.inlier_tol # the tolerance value for considering a point to be
        an inlier

    # Initialize variables
    num = locs1.shape[0]  # number of points
    # handle special case when number of points is less than 4
    if num < 4:
        raise ValueError("Number of points is less than 4")

    bestH2to1 = None
    best_inliers = np.zeros(len(locs1))
    max_inliers = 0

    for i in range(max_iters):
        # Sample 4 points randomly
        idx = np.random.choice(num, 4, replace=False)
        x1_sample = locs1[idx]
        x2_sample = locs2[idx]

        # Compute homography
        H2to1_sample = computeH_norm(x1_sample, x2_sample)

        # Transform points
        locs2_homogeneous = np.hstack((locs2, np.ones((num, 1))))
        locs1_project = (H2to1_sample @ locs2_homogeneous.T).T
        locs1_project /= locs1_project[:, 2:3]  # normalize points


        # Compute distance between points
        euclidean_distances = np.sqrt(np.sum((locs1 - locs1_project[:, :2])**2, axis
            =1))

        # Determine inliers
        inliers = euclidean_distances < inlier_tol
        inlier_count = np.sum(inliers)

        # Update best homography
        if inlier_count > max_inliers:
            max_inliers = inlier_count
            bestH2to1 = H2to1_sample
            best_inliers = inliers

        # Print inliers information
        # if i % 100 == 0:  # Print every 100 iterations
        #     print(f"Iteration {i}: {inlier_count} inliers")

    return bestH2to1, best_inliers
```

Q2.2.4: Automated Homography Estimation and Warping
Include your code and result in your write-up.

Below is the composited image results running default RANSAC parameters:



The composited image output using default parameters.

Here is the code snippet for compositeH:

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

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

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

    # Invert homography
    H2to1_invert = np.linalg.inv(H2to1)


    # TODO: Create mask of same size as template
    mask = np.ones((template.shape[0], template.shape[1]), dtype=np.uint8)

    # TODO: Warp mask by appropriate homography
    warped_mask = cv2.warpPerspective(mask, H2to1_invert, (img.shape[1], img.shape
        [0]))


    # TODO: Warp template by appropriate homography
    warped_template = cv2.warpPerspective(template, H2to1_invert, (img.shape[1], img
        .shape[0]))


    # TODO: Use mask to combine the warped template and the image
    'handle both grayscale and color images'
    # If grayscale
    if len(img.shape) == 2:
        composite_img = img.copy()
        composite_img[warped_mask > 0] = warped_template[warped_mask > 0]
    # If color
    else:
        composite_img = img.copy()
        for i in range(3):
            composite_img[:, :, i] = composite_img[:, :, i] * (1 - warped_mask) +
                warped_template[:, :, i] * warped_mask

    return composite_img
```

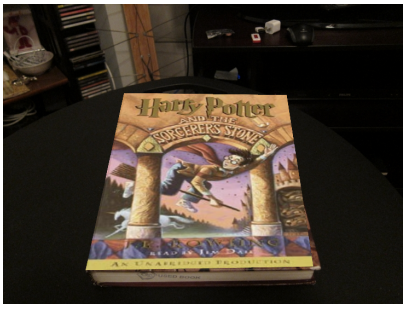
Here is the code snippet for warpImage():

```
# Import necessary functions
from matchPics import matchPics
from planarH import computeH_ransac, compositeH
from displayMatch import displayMatched


# Q2.2.4

def warpImage(opts):
    # Read images
    cv_cover = cv2.imread('../data/cv_cover.jpg')
    cv_desk = cv2.imread('../data/cv_desk.png')
    hp_cover = cv2.imread('../data/hp_cover.jpg')

    # Check if images are loaded properly
    if cv_cover is None or cv_desk is None or hp_cover is None:
        print("Error: Image loading failed.")
        return

    # Resize hp_cover to cv_cover size
    hp_cover_resized = cv2.resize(hp_cover, (cv_cover.shape[1], cv_cover.shape[0]))

    # Compute homography
    matches, locs1, locs2 = matchPics(cv_cover, cv_desk, opts)

    if matches is None or locs1 is None or locs2 is None:
        print("Error: Feature matching failed.")
        return

    # Implement RANSAC
    bestH2to1, best_inliers = computeH_ransac(locs1[matches[:,0]][:, [1, 0]], locs2[
        matches[:,1]][:, [1, 0]], opts)

    print(f"Best Homography Matrix:\n{bestH2to1}")
    print(f"Number of inliers: {np.sum(best_inliers)}")

    # Warp images
    composite_img = compositeH(bestH2to1, hp_cover_resized, cv_desk)
    cv2.imwrite('../results/Q2.2.4_result.png', composite_img)

    # display matched features for debugging
    # displayMatched(opts, cv_cover, cv_desk)

    # display images
    cv2.imshow('Composited Image', composite_img)
    cv2.waitKey(0)

    pass


if __name__ == "__main__":

    opts = get_opts()
    warpImage(opts)
```

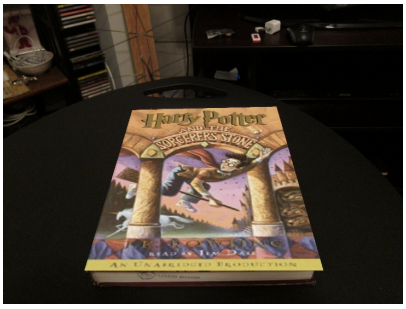
Q2.2.5: RANSAC Parameter Tuning
Include the result images in your write-up, and explain the effect of these two parameters respectively.

A small ablatioin study was conducted and the results are the shown below.

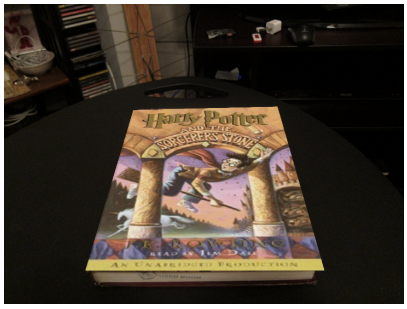
Varying number of iterations with constant inlier tolerance:



iteration 25, tolerance 2.0

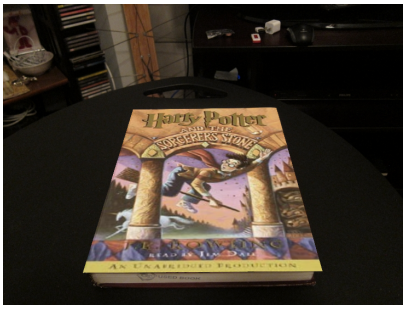


iteration 500, tolerance 2.0

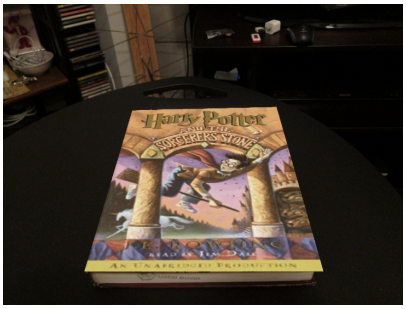


iteration 1000, tolerance 2.0

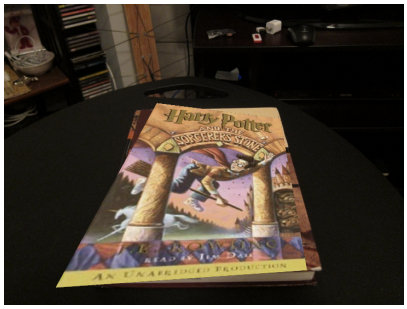
Varying number of inlier tolerance with constant iterations:



iteration 500, tolerance 0.5



iteration 500, tolerance 2.0

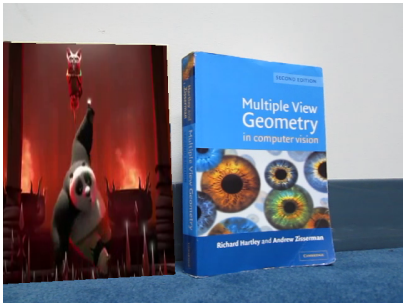


iteration 500, tolerance 10

From the above study, we can observe that higher number of iterations and lower tolerances generates better results compared to lower number of iterations and higher tolerances. Number of iterations allows the algorithm to sample more data, which increases the robustness but requires more time for computation. On the other hand, the inlier tolerance value dictates the inlier selection strictness. High tolerance leads to more matching features but lower accuracy (outliers).

Q3.1: Incorporating video
Please write a script ar.py to implement this AR application and save your result video as ar.avi in the result/ directory. You may use the function loadVid() that we provide to load the videos. In your write-up, include three screenshots of your ar.avi at three distinct timestamps (e.g. when the overlay is near the center, left, and right of the video frame). See Figure 4 as an example of where the overlay is in the center of the video frame. Please also include the code snippet in your write-up. If the video is too large, please include a Google Drive link to your video in the write-up instead and ensure the shared link gives the TA's viewing permission.

Below are the three screenshots of my ar.mp4 at three distinct timestamps:



Overlay is near the left.

Overlay is near the middle.

Overlay is near the right.

Google drive link for my ar.mp4: https://drive.google.com/drive/folders/19VZeDIA6CSnlZhU56NFZRpoB2XjLLrjK

Here is the code snippet:

```
import numpy as np
import cv2
import skimage.io
import skimage.color
from opts import get_opts

# Import necessary functions
from matchPics import matchPics
from planarH import computeH_ransac, compositeH
from displayMatch import displayMatched


# Q2.2.4

def warpImage(opts):
    # Read images
    cv_cover = cv2.imread('../data/cv_cover.jpg')
    cv_desk = cv2.imread('../data/cv_desk.png')
    hp_cover = cv2.imread('../data/hp_cover.jpg')

    # Check if images are loaded properly
    if cv_cover is None or cv_desk is None or hp_cover is None:
        print("Error: Image loading failed.")
        return


    # Resize hp_cover to cv_cover size
    hp_cover_resized = cv2.resize(hp_cover, (cv_cover.shape[1], cv_cover.shape[0]))


    # Compute homography
    matches, locs1, locs2 = matchPics(cv_cover, cv_desk, opts)

    if matches is None or locs1 is None or locs2 is None:
        print("Error: Feature matching failed.")
        return

    # Implement RANSAC
    bestH2to1, best_inliers = computeH_ransac(locs1[matches[:,0]][:, [1, 0]], locs2[
        matches[:,1]][:, [1, 0]], opts)

    # print(f"Best Homography Matrix:\n{bestH2to1}")
    # print(f"Number of inliers: {np.sum(best_inliers)}")

    # Warp images
    composite_img = compositeH(bestH2to1, hp_cover_resized, cv_desk)
    cv2.imwrite('../results/Q2.2.5_result.png', composite_img)

    # display matched features for debugging
    # displayMatched(opts, cv_cover, cv_desk)

    # display images
    cv2.imshow('Composited Image', composite_img)
    cv2.waitKey(0)

    pass
```

```
if __name__ == "__main__":

    opts = get_opts()
    warpImage(opts)
```

Q3.2: Make Your AR Real Time (Extra Credit)
As an output of the script, you should process the videos frame by frame and have the combined frames played in real time. Make sure to note the achieved fps in your write-up in addition to all the steps taken to achieve real-time performance. Please also include the code snippet in your write-up.

Did not attempt.

Q4: Create a Simple Panorama

Below are the panorama result images:

Original left image.

Original right image.

Panorama image.

Here is the code snippet:

```
# Import necessary functions
import matplotlib.pyplot as plt
from opts import get_opts
from matchPics import matchPics
from planarH import computeH_ransac, compositeH
from displayMatch import displayMatched

# Q4

def assemble_panorma(img_left_path, img_right_path,opts):

    # Read images
    img_left = cv2.imread(img_left_path)
    img_right = cv2.imread(img_right_path)

    # Check if images are loaded properly
    if img_left is None or img_right is None:
        print("Error: Image loading failed.")
        return

    # Compute homography
    matches, locs1, locs2 = matchPics(img_left, img_right, opts)

    if matches is None or locs1 is None or locs2 is None:
        print("Error: Feature matching failed.")
        return

    # Implement RANSAC
    bestH2to1, best_inliers = computeH_ransac(locs1[matches[:,0]][:, [1, 0]], locs2[
        matches[:,1]][:, [1, 0]], opts)

    # Warp images
    panorama_img = compositeH(bestH2to1, img_left, img_right)

    # display matched features For debugging
    # displayMatched(opts, img_left, img_right)


    # save and display parnorama image
    cv2.imwrite('../results/Q4_result.png', panorama_img)
    cv2.imshow('Panorma Image', panorama_img)
    cv2.waitKey(0)

    pass

if __name__ == "__main__":

        opts = get_opts()
        # my images
        img_left_path = '../data/mypano_left.jpg'
        img_right_path = '../data/mypano_right.jpg'
        assemble_panorma(img_left_path, img_right_path, opts)
```

References

[1] D. Tyagi, "Introduction to BRIEF (Binary Robust Independent Elementary Features)," *Medium*, 7-Apr-2020. [Online]. Available: https://medium.com/@deepanshut041/introduction-to-brief-binary-robust-independent-elementary-features-436f4a31a0e6.

[2] "What is Hamming Distance?," *TutorialsPoint*. [Online]. Available: https://www.tutorialspoint.com/what-is-hamming-distance.

[3] Wikimedia Foundation, "Harris corner detector," *Wikipedia*, 6-Jan-2024. [Online]. Available: https://en.wikipedia.org/wiki/Harris_corner_detector.

[4] Wikimedia Foundation, "Features from Accelerated Segment Test," *Wikipedia*, 25-Jun-2024. [Online]. Available: https://en.wikipedia.org/wiki/Features_from_accelerated_segment_test.