

# CS585 Problem Set 3 (Total points: 30 + 5 bonus)

Assignment adapted from Svetlana Lazebnik

## Instructions

1. Assignment is due at **5 PM on Tuesday Feb 22 2022**.

2. Submission instructions:

A. A single `.pdf` report that contains your work. Your response for all questions should be electronic (no handwritten responses allowed). You should respond to the questions individually and include images as necessary. Your response in the PDF report should be self-contained. It should include all the output you want us to look at. You will not receive credit for any results you have obtained, but failed to include directly in the PDF report file.

PDF file should be submitted to [Gradescope](#) under `PS3`. Please tag the responses in your PDF with the Gradescope questions outline as described in [Submitting an Assignment](#).

B. You also need to submit code in the form of a single `.zip` file that includes all your code, all in the same directory. You can submit Python code in `.ipynb` format. Code should also be submitted to [Gradescope](#) under `PS3-Code`. *Not submitting your code will lead to a loss of 100% of the points.*

C. We reserve the right to take off points for not following submission instructions. In particular, please tag the responses in your PDF with the Gradescope questions outline as described in [Submitting an Assignment](#).

## Problems

**Stitching pairs of images.** We will estimate homography transforms to register and stitch image pairs.

**Test Images:** We are providing a image pairs that you should stitch together. We have also provided sample output, though, keep in mind that your output may look different from the reference output depending on implementation details.

**Getting Started** Detect feature points in both images and extract descriptor of every keypoint in both images. We will use SIFT descriptors from OpenCV library. You can refer to this [tutorial](#) for more details about using SIFT in OpenCV. Please use opencv version 4.5 or later.

In [169...]

```
# to run in google colab
import sys
if 'google.colab' in sys.modules:
    import subprocess
    subprocess.call("pip install -U opencv-python".split())
```

In [229...]

```
import matplotlib.pyplot as plt
import numpy as np
import cv2, skimage
from scipy.spatial import distance
import scipy

# some helper functions

def imread(fname):
    """
    read image into np array from file
    """
    return cv2.imread(fname)

def imread_bw(fname):
    """
    read image as gray scale format
    """
    return cv2.cvtColor(imread(fname), cv2.COLOR_BGR2GRAY)

def imshow(img):
    """
    show image
    """
    plt.imshow(img)
```

```

skimage.io.imshow(img)

def get_sift_data(img):
    """
    detect the keypoints and compute their SIFT descriptors with opencv library
    """
    sift = cv2.SIFT_create()
    kp, des = sift.detectAndCompute(img, None)
    kp = np.array([k.pt for k in kp])
    return kp, des

def plot_inlier_matches(ax, img1, img2, inliers):
    """
    plot the match between two image according to the matched keypoints
    :param ax: plot handle
    :param img1: left image
    :param img2: right image
    :param inliers: x,y in the first image and x,y in the second image (Nx4)
    """
    res = np.hstack([img1, img2])
    ax.set_aspect('equal')
    ax.imshow(res, cmap='gray')

    ax.plot(inliers[:,2], inliers[:,3], '+r')
    ax.plot(inliers[:,0] + img1.shape[1], inliers[:,1], '+r')
    ax.plot([inliers[:,2], inliers[:,0] + img1.shape[1]],
            [inliers[:,3], inliers[:,1]], 'r', linewidth=0.4)
    ax.axis('off')

```

**1. Putative Matches [5 pts].** Select putative matches based on the matrix of pairwise descriptor distances. First, Compute distances between every descriptor in one image and every descriptor in the other image. We will use `scipy.spatial.distance.cdist(X, Y, 'squared')`. Then, you can select all pairs whose descriptor distances are below a specified threshold, or select the top few hundred descriptor pairs with the smallest pairwise distances. In your report, display the putative matches overlaid on the image pairs.

In [230...]

```

def get_best_matches(img1, img2, num_matches):
    kp1, des1 = get_sift_data(img1)
    kp2, des2 = get_sift_data(img2)

    # Find distance between descriptors in images
    dist = scipy.spatial.distance.cdist(des1, des2, 'squared')

    # Write your code to get the matches according to dist
    # <YOUR CODE>
    idx = np.argpartition(dist, 100, axis = None)
    kp1_pos = idx[:num_matches]//kp2.shape[0]
    kp2_pos = idx[:num_matches]%kp2.shape[0]
    return np.hstack([kp2[kp2_pos], kp1[kp1_pos]])

img1 = imread('./stitch/left.jpg')
img2 = imread('./stitch/right.jpg')

data = get_best_matches(img1, img2, 300)
fig, ax = plt.subplots(figsize=(20,10))
plot_inlier_matches(ax, img1, img2, data)
#fig.savefig('sift_match.pdf', bbox_inches='tight')

```



**1. Homography Estimation and RANSAC [20 pts].** Implement RANSAC to estimate a homography mapping one image onto the other.

Describe the various implementation details, and report all the hyperparameters, the number of inliers, and the average residual for the inliers (mean squared distance between the point coordinates in one image and the transformed coordinates of the matching point in the other image). Also, display the locations of inlier matches in both images.

**Hints:** For RANSAC, a very simple implementation is sufficient. Use four matches to initialize the homography in each iteration. You should output a single transformation that gets the most inliers in the course of all the iterations. For the various RANSAC parameters (number of iterations, inlier threshold), play around with a few reasonable values and pick the ones that work best. Homography fitting calls for homogeneous least squares. The solution to the homogeneous least squares system  $AX = 0$  is obtained from the SVD of  $A$  by the singular vector corresponding to the smallest singular value. In Python, `U, S, V = numpy.linalg.svd(A)` performs the singular value decomposition (SVD). This function decomposes  $A$  into  $U, S, V$  such that  $A = USV$  (and not  $USV^T$ ) and `V[-1, :]` gives the right singular vector corresponding to the smallest singular value. *Your implementation should not use any opencv functions.*

In [519]...

```
def ransac(data, max_iters=10, min_inliers=10):
    """
    write your ransac code to find the best model, inliers, and residuals
    """

    DESCRIBE IMPLEMENTATIONS:
    1) Compute Homography:
    Construct A to solve  $AX = 0$  using SVD, extract  $v[-1, :]$  and reshape to get
    transformation matrix
    2) ransac:
    randomly choose 4 points to compute homography on.
    Use the result Homography H to fit ransac algorithm, then return H with the
    most inliers.
    Parameters choice explain:
    * inlier threshold is small because we want the transformed img to fit well
    However not too small, or it will not be able to find a good transformation

    # <YOUR CODE>
    #print(data)
    inlier_threshold = 4
    homo_list = [] #list of homography
    numinliers = [] #number of inliers for each homography
    inliers_list = [] #list of inliers for each homography
    residual = []
    for i in range(max_iters):
        count=0 #counter to count inliers
        inliers_list_of_this_line = []
        randomPoint = np.array([0,0,0,0])
        for n in range(4):
            randomPoint[n] = np.random.randint(0, data.shape[0])
        matches = np.vstack([data[randomP] for randomP in randomPoint])
        H = compute_homography(matches)
        img1xy = np.vstack([data[:,2], data[:,3], np.ones((data.shape[0],))])
        img2xy = np.vstack([data[:,0], data[:,1], np.ones((data.shape[0],))])
        transform = H@img2xy
        error = np.sum((img1xy - transform)**2, axis = 0)
        if error < inlier_threshold:
            count+=1
            inliers_list.append(transform)
            numinliers.append(count)
    return H, inliers_list, numinliers
```

```

sum_error = 0
for i, e in enumerate(error):
    if e < inlier_threshold:
        count +=1
        inliers_list_of_this_line.append(data[i])
        sum_error += e

homo_list.append(H)
numinliers.append(count)
inliers_list.append(inliers_list_of_this_line)
if count!=0:
    residual.append(sum_error/count)
else:
    residual.append(999999)

index = np.argmax(np.array(numinliers))
print('Residual:', residual[index])
# display inliers
print('----Inliers Display----')
fig, ax = plt.subplots(figsize=(20,10))
res = np.hstack([img1, img2])
ax.set_aspect('equal')
ax.imshow(res, cmap='gray')
ax.plot(np.array(inliers_list[index])[:,2], np.array(inliers_list[index])[:,3], 'o', color = 'r')
ax.plot(np.array(inliers_list[index])[:,0] + img1.shape[1], np.array(inliers_list[index])[:,1], 'o', color = 'w')
ax.axis('off')

return homo_list[index], numinliers[index]

def compute_homography(matches):
    """
    write your code to compute homography according to the matches
    """
    # <YOUR CODE>
    x = []
    x_prime = []
    zero = np.zeros((3,))
    for i in range(4):
        x.append(np.hstack([matches[i,0], matches[i,1], 1]))
        x_prime.append(np.hstack([matches[i,2], matches[i,3], 1]))
    x = np.array(x)
    x_prime = np.array(x_prime)
    homography = []
    for i in range(4):
        homography.append(np.array(np.hstack([zero.T, x[i], -x_prime[i][1]*(x[i])])))
        homography.append(np.array(np.hstack([x[i], zero.T, -x_prime[i][0]*(x[i])])))
    homography = np.array(homography)
    u,s,v = np.linalg.svd(np.array(homography))
    M = np.reshape(v[-1,:],(3,3))
    M = M/M[2,2]
    return M

# Report the number of inliers in your matching
H, max_inliers = ransac(data)
print("Inliers:", max_inliers)

```

Residual: 1.731694529071912  
----Inliers Display---  
Inliers: 5



1. **Image Warping [5 pts].** Warp one image onto the other using the estimated transformation. You can use opencv functions for this purpose.

In [521...]

```
from PIL import Image
from IPython.display import display

def warp_images(img1, img2, H):
    """
    write your code to stitch images together according to the homography
    """
    # <YOUR CODE>
    dst = cv2.warpPerspective(img2, H, (img1.shape[1]+img2.shape[1], img1.shape[0]))
    return np.hstack([img1, dst[:, (dst.shape[1]-img1.shape[1]):]])

# display the stitching results
img_warped = warp_images(img1, img2, H)
display(Image.fromarray(img_warped))
```



1. **Image Warping [5 bonus pts].** Warp one image onto the other using the estimated transformation *without opencv functions*. Create a new image big enough to hold the panorama and composite the two images into it. You can composite by averaging the pixel values where the two images overlap, or by using the pixel values from one of the images. Your result should look similar to the sample output. You should create **color panoramas** by applying the same compositing step to each of the color channels separately (for estimating the transformation, it is sufficient to use grayscale images). You may find `ProjectiveTransform` and `warp` functions in `skimage.transform` useful.

In [668...]

```
import skimage.transform as skt
from scipy.interpolate import NearestNDInterpolator

def warp_images_noopencv(img1, img2, H):
    """
    write your code to stitch images together according to the homography
```

```

"""
# <YOUR CODE>
panorama = np.empty((img1.shape[0],img1.shape[1]+img2.shape[1], 3))
for x in range(0, img2.shape[1]):
    for y in range(0, img2.shape[0]):
        newxy = H@np.hstack([x,y,1])
        newxy = newxy/newxy[2]
        y_coor = int(newxy[1])
        x_coor = int(newxy[0])
        if (x_coor < panorama.shape[1]) and ( 0 <= y_coor < panorama.shape[0]):
            panorama[y_coor, x_coor] = img2[y,x].copy()

#fill the pixels with missing colors with their nearest neighbor values
#using Interpolator
data = panorama
mask = np.where(~(data==0))
interp = NearestNDInterpolator(np.transpose(mask), data[mask])
filled_data = interp(*np.indices(data.shape))
panorama = filled_data.copy()
panorama[:, :img1.shape[1]] = img1.copy()
return panorama.astype(np.uint8)

# display and report the stitching results
img_warped_nocv = warp_images_noopencv(img1, img2, H)
display(Image.fromarray(img_warped_nocv))

```



In [ ]: