

CSCI 4270 and 6270  
Computational Vision,  
Spring Semester 2022

Homework 4

Due: Wednesday, March 16, at 11:59:59 pm EST

This homework, worth **125 points** toward your overall homework grade, is to be done in two parts. The first is a multistage problem of recognizing whether or not images are taken of the same scene and building a montage of images taken not only of the same scene, but from the same viewpoint (or taken of a planar surface). This problem will require an extensive careful write-up describing your design decisions and results. The second part will compare descriptor matching methods and also involve a write-up on your results.

Each part will be submitted separately through Submittity, but neither will involve any Submittity auto-grading.

## Part 1 — 100 Points

### Overview

Here is the basic problem statement, which is elaborated on below: Given a folder of  $N$  images as input, for each pair of images  $I_i$  and  $I_j$  you must

1. Decide if  $I_i$  and  $I_j$  show the same scene.
2. If the decision for 1 is “yes”, then decide if  $I_i$  and  $I_j$  can be aligned accurately enough to form a mosaic.
3. If the decisions for both 1 and 2 are “yes”, then create and output the mosaic that aligns image  $I_i$  and  $I_j$  accurately.

It is possible that more than two images in a set of input images do show the same scene **and** may be combined into a mosaic. This is where the undergraduate and graduate versions of this assignment differ. In order to earn full credit, graduate students **must** produce a multi-image mosaic (in addition to the image pair mosaics). Undergraduates can earn a small amount of extra credit for multi-image mosaics. For graduate students this is the last 10 points on the assignment. For undergraduates, this is 5 points of extra credit. More on this below.

### Details

Each image set you are given includes  $N \geq 2$  images,  $I_1, \dots, I_N$ . Each image should be read in and processed as grayscale! For each pair of images,  $I_i$  and  $I_j$ , with  $1 \leq i < j \leq N$ , your code must do the following:

1. Extract the keypoints and descriptors in each image. We strongly urge you to use SIFT keypoints and descriptors, but you may use anything you wish. **Output** the number of keypoints in each image.
2. Match the keypoints and descriptors between the images. You may use `cv2.BFMatcher` or `cv2.FlannBasedMatcher` to do the matching. The decision about whether or not two keypoints match may be made using the ratio test for descriptors like SIFT, or using the

symmetric matching criteria for descriptors like ORB. At this point there will often be errors in your keypoint matches.

**Output:**

- (a) The number of matches and the fraction of keypoints in each image that have a match (this should be significantly less than 1).
  - (b) A single image showing  $I_i$  and  $I_j$  side-by-side with lines drawn between matched keypoints (see `cv2.drawMatches`). Make each line a different color.
3. If the previous step produced too few matches overall or too small a percentage of matches, then stop attempting to match  $I_i$  and  $I_j$ . (You will need to decide the criteria and the threshold or thresholds.) Otherwise proceed to the next step of matching. **Output** a message giving the decision made at this step.
  4. Using the matches produced by keypoint description matching, use RANSAC to estimate the fundamental matrix  $\mathbf{F}_{j,i}$  that maps pixel locations from  $I_i$  onto lines in  $I_j$ . Please review the significance of the fundamental matrix in your class notes! You may use `cv2.findFundamentalMat`. Do this with the `method` setting `cv2.FM_RANSAC`; you will have to explore the other parameter settings.

After estimating  $\mathbf{F}_{j,i}$ , you must determine which matches are “inliers” — consistent with the fundamental matrix. Specifically, if  $\tilde{\mathbf{u}}_i$  (from image  $I_i$ ) and  $\tilde{\mathbf{u}}_j$  (from image  $I_j$ ) are the homogeneous coordinate locations of a matching keypoint, then  $\mathbf{a}_{j,i} = \mathbf{F}_{j,i}\tilde{\mathbf{u}}_i$  will be the coordinates describing the line in image  $j$  along which  $\tilde{\mathbf{u}}_j$  should lie if it is a correct match. (This is the “epipolar line”.) While in theory  $\tilde{\mathbf{u}}_j$  would be exactly on the line, in practice it may be slightly off. On the other hand, most incorrect matches will typically have  $\tilde{\mathbf{u}}_j$  far from this line. Therefore you can determine which matches are inliers by measuring the distance between  $\tilde{\mathbf{u}}_j$  and the line and counting the number of keypoint matches that are within a small distance of the line. This is easy to do yourself as long as you determine the threshold and are careful to normalize  $\mathbf{a}_{j,i}$  properly so that you can measure distances correctly. **However**, the `mask` array that `cv2.findFundamentalMat` returns does this for you! You are welcome to use it.

**Output** the following from this step:

- (a) The number and percentage of matches that are inliers.
  - (b) An image showing  $I_i$  and  $I_j$  side-by-side with lines drawn between the keypoints that form inlier matches (see `cv2.drawMatches`). Make each line a different color.
  - (c) An image showing the epipolar lines for the inlier matches drawn on image  $I_2$ . Make each line a different color. This one may take a bit of work so we suggest saving it until after everything else is working.
5. If the previous step produced too few matches overall or too small a percentage of matches, then stop attempting to match  $I_i$  and  $I_j$ , and move on to the next image pair. (You will need to decide the criteria and the threshold or thresholds.) Otherwise proceed to the next step of matching. At this point your code will have made the decision that tells us whether or not  $I_i$  and  $I_j$  show the same scene. **Output** a message giving the decision made at this step.
  6. Using the inlier matches from the fundamental matrix estimation step, estimate the parameters of the homography matrix  $\mathbf{H}_{j,i}$  mapping  $I_i$  onto  $I_j$ . You may use `cv2.findHomography`

and RANSAC. Using a criteria for deciding which matches are “inliers”, count the number of inliers for the homography matching between images.

**Output** the following from this step:

- (a) The number and percentage of inlier matches.
  - (b) An image showing  $I_i$  and  $I_j$  side-by-side with lines drawn between the keypoints that form inlier matches (see `cv2.drawMatches`). Make each line a different color.
7. Based on the number of inlier matches from fundamental matrix estimation and from homography estimation, decide whether or not the images can be accurately aligned. The decision should be “yes” if most of the inlier matches from the fundamental matrix estimate are also kept as inliers to the homography estimate. **Output** your decision and the reason for your decision.
  8. If the decision after the previous step is “yes” then build and output the mosaic of the two images. Try to come up with a relatively simple blending method that yields nice results instead of looking like one image is mapped and pasted on top of the other.

## Multi-Image Mosaics

Here is a bit about forming multi-image mosaics, a problem you should leave until everything else is done. First, you need to remember which pairs of images can be aligned using a homography. Think of the images as nodes in a graph and the image pairs as edges. The images that will form the mosaic are the connected components. (If for some reason there is more than one connected component, pick the largest.) Second, you will need to pick an “anchor” image that will remain fixed while the other images are mapped onto it. This should in some sense be the “center” of the set of images in the connected component. Third, you need to compute the transformations that map the images onto this anchor image. This can get tricky quite quickly, so please do something very easy using only the results of matching pairs of images. In particular, if  $I_0$  is the anchor and  $I_i$  is successfully matched with  $I_0$ , then use the transformation homography computed between them. If  $I_i$  does not have a homography with  $I_0$ , but there is another image  $I_j$  that does, then “compose” the transformations:  $I_i$  onto  $I_j$  onto  $I_0$ . This is not as hard as it sounds. In particular, if  $\mathbf{H}_{j,i}$  is the estimated transformation matrix from  $I_i$  onto  $I_j$  and if  $\mathbf{H}_{0,j}$  is the estimated transformation matrix from  $I_j$  onto  $I_0$ , then  $\mathbf{H}_{0,i} = \mathbf{H}_{0,j}\mathbf{H}_{j,i}$  is a good estimate of the transformation from  $I_i$  onto  $I_0$ . In the data I provide there **will not** be any cases where you need to compose more than two transformations if you choose the anchor correctly. Note that commercial software that builds multi-image mosaics uses much more sophisticated methods to estimate  $\mathbf{H}_{0,i}$ .

## Command Line and Output

Your program should run with the following very simple command line:

```
python hw4_align.py in_dir out_dir
```

where `in_dir` is the path to the directory containing input images. We will run some of your submissions to test them. The code should write all images to `out_dir`, which should be a different directory from `in_dir`. This will avoid clutter across multiple runs. Your code will need to output (via `print` statements) a significant amount of text as described above. For each mosaic you create, make the file name be the composition of the names of the input file prefixes, in sorted order. For example, if the images are `bar.jpg`, `cheese.jpg` and `foo.jpg`, then the mosaic of the first two will

be `bar_foo.jpg` and the mosaic of all three will be `bar_cheese_foo.jpg`. Use the extension from the first image (all images will be jpg or JPG). Note that for image pairs that do form mosaics, there will be four output images — the images that result from steps 2, 4, 6 and 8. For pairs that do not form mosaics, there will be fewer output images, depending on which decision (steps 3, 5 or 7) stopped the computation. There will always be an output image from step 2.

## Write Up and Code

Please generate a write-up describing your algorithms, your decision criteria, your blending algorithms, and your overall results. Evaluate **both strengths and weaknesses**, using images — perhaps including some we did not provide — to illustrate. One suggestion is to make a table summarizing the results on all the image pairs, including the matching results, the number and percentage of inliers to F and to H (if they were estimated), and the final decision your algorithm made. This will take some time to generate but it is the type of analysis you will need to learn how to make to evaluate computer vision and machine learning algorithms. You don't have to make this beautiful, just make it clear and easy to follow. The actual text should be no more than a page or so, single-spaced, but the document should be longer because of the results table and the illustrating images.

Finally, make sure your code is clean, reasonably efficient, documented, and well-structured.

## Complete Submission

Your final submission for Part 1 will be a single zip file that includes the following:

1. Your .py file
2. The text output files from running your code on **each** of the image sets provided, plus other image sets you'd like to show. One additional suggestion is to run your algorithm on two images taken from different sets.
3. As many image results as you need to illustrate your successes (and failures), both in forming mosaics and in deciding not to do so!
4. Your final write-up.

**The zip file will be limited to 60MB.** This means it is unlikely that you can include all image results.

## Part 2 — Comparing Descriptor Matching Methods — 25 Points

SIFT keypoint descriptor matching is based on the ratio test. ORB and other matching methods use symmetric matching. This could be used as well for SIFT, but should it? In this problem you will write a Python script to try to analyze this question

First, here is the definition of symmetric matching. Let  $\mathbf{u}_i$ ,  $i \in 1, \dots, N_u$  be the descriptor vectors for the keypoints from image  $I_u$  and let  $\mathbf{v}_j$ ,  $j \in 1, \dots, N_v$  be the descriptor vectors from image  $I_v$ . Then a descriptor  $\mathbf{u}_{i^*}$  from  $I_u$  and a descriptor  $\mathbf{v}_{j^*}$  from  $I_v$  are matched if

$$j^* = \operatorname{argmin}_{j \in 1, \dots, N_v} D(\mathbf{u}_{i^*}, \mathbf{v}_j).$$

and

$$i^* = \underset{i \in 1, \dots, N_u}{\operatorname{argmin}} D(\mathbf{u}_i, \mathbf{v}_{j^*}).$$

where  $D(\cdot, \cdot)$  measures the distance between two descriptors — Euclidean distance for SIFT and Hamming distance for ORB. More simply put,  $\mathbf{u}_{i^*}$  and  $\mathbf{v}_{j^*}$  are matched if each is the other's closest descriptor.

Your job is to implement symmetric matching for SIFT descriptors and compare to ratio test matching, also for SIFT descriptors. You should analyze both (1) image pairs that show the same scene and therefor should match, and (2) image pairs that do not show the same scene and therefore should not match. Note that in the latter case, there are no truly correct matches. Also note that the information about whether or not the two images should be matched is provided to your code in the command-line (based on what you learn from the results of Part 1).

For each pair of images,  $I_1$  and  $I_2$ , let their keypoints be the sets  $\mathcal{K}_1$  and  $\mathcal{K}_2$ . Compute the matches between the keypoint sets as in Part 1 using the ratio test and then using the symmetric matching test. Call the resulting sets of matches  $\mathcal{M}_R$  and  $\mathcal{M}_S$ , respectively.

Your first set of outputs should be the number and percentage of keypoints that matched using the ratio test and using the symmetric matching test. To be specific these are

$$|\mathcal{M}_R| \quad \text{and} \quad |\mathcal{M}_S|$$

for the counts, and

$$\frac{|\mathcal{M}_R|}{\min(|\mathcal{K}_1|, |\mathcal{K}_2|)} \quad \text{and} \quad \frac{|\mathcal{M}_S|}{\min(|\mathcal{K}_1|, |\mathcal{K}_2|)}$$

for the percentages.

The second set of outputs should only be generated if the two images should match. In this case, use the match set  $\mathcal{M}_R$  to estimate the fundamental matrix  $\mathbf{F}$ , as in Part 1, Step 4. Then identify the inlier matches from  $\mathcal{M}_R$  and  $\mathcal{M}_S$ , calling these sets  $\mathcal{M}'_R$  and  $\mathcal{M}'_S$ . The same  $\mathbf{F}$  should be used in both cases, so you'll need to implement the method to count inliers discussed at the end of Part 1, Step 4, at least for  $\mathcal{M}_S$ . The output should be the size of the inlier sets

$$|\mathcal{M}'_R| \quad \text{and} \quad |\mathcal{M}'_S|$$

and the percentage of matches that are inliers

$$\frac{|\mathcal{M}'_R|}{|\mathcal{M}_R|} \quad \text{and} \quad \frac{|\mathcal{M}'_S|}{|\mathcal{M}_S|}.$$

Based on results from several pairs of images, make a recommendation about whether the ratio test or symmetric matching is better and why.

## Command-Line and Output

Here is a suggested command-line

```
python compare.py img1 img2 should_match
```

where `img1` and `img` are the image file names, and `should_match` is a boolean flag (0 or 1) indicating whether or not the images should match.

Use images we provided for Part 1 and any other pictures you'd like to try.

## **What to Submit**

Submit just two files zipped together, `compare.py` and a pdf file writeup summarizing your results and recommendations. The writeup should be less than a page of text plus any results or pictures you'd like to include to illustrate. Try to convince us that your recommendation is correct.