AI3604 – Computer Vision
Fall 2020
Homework #2
Due: 11:59 PM, Tuesday, December 7, 2021

In this assignment, you will write Python code to detect discriminating features in an image and find the best matching features in other images. Then you will estimate a global transformation between the image pair using your matched features and then warp one image towards the other to stitch them together as a panorama. You'll evaluate their performance on a suite of benchmark images.

Submit your code and write-up (see explanation below) on OC.

You will get no credit for using a "magic" function to answer any questions where you should be answering them. If you are unsure of an allowable function, please ask on OC or ask Teaching Assistants before assuming.

Along with this handout you will find Python files containing the functions you need to complete. The docstring in each function specifies the type and format of each input/output argument. Please follow these specifications and do not modify the function signatures. Feel free to implement your own helper functions if necessary.

You also need a "driver" program to load and pre-process the images, call the functions you implemented, and display or save your results. Your driver program should be implemented inside main function.

The "data" subfolder contains the following images for you to test with:

- bikes1.png, bikes2.png, bikes3.png
- graf1.png, graf2.png, graf3.png
- leuven1.png, leuven2.png, leuven3.png
- wall1.png, wall2.png, wall3.png

Written Assignment

You need to include a write-up accompanying your code. In the write-up you need to do the following:

(1) For each required function, briefly explain how you implement it and list all the hyperparameters you need to choose/tune and report the values you use. Examples include (**but are not limited to**):

- If a filter or a window is used, what is the size of it?
- When thresholding a score map, what is the threshold value?

(2) You will need to compare the following image pairs and report the results (more details below on how to compare):

- bikes1 ↔ bikes2, bikes1 ↔ bikes3
- graf1 ↔ graf2, graf1 ↔ graf3
- leuven1 ↔ leuven2, leuven1 ↔ leuven3
- wall1 ↔ wall2, wall1 ↔ wall3

Programming Assignment

Blob Detection (15 points)

As we have seen in the lectures, Harris corners are not invariant to scale change. Implement the Laplacian blob detection algorithm (see the lecture notes for details) .

Complete function **detect_blobs**. In addition to the feature locations, this function should also return the corresponding blob scales and orientations.

Feature Description (15 points)

Now, let's see if we can do a better descriptor for matching features than cross-correlation or SSD. Try implementing the descriptors described in section 6.1 of the SIFT paper. Be careful of features by the image boundary. Ignore those that are too close to the boundary such that the window you are using cannot fully fit around it.

Complete function **compute_descriptors**. For each corner location, we now should have a 128-element feature descriptor which we will use in the feature matching step, but with a different matching strategy.

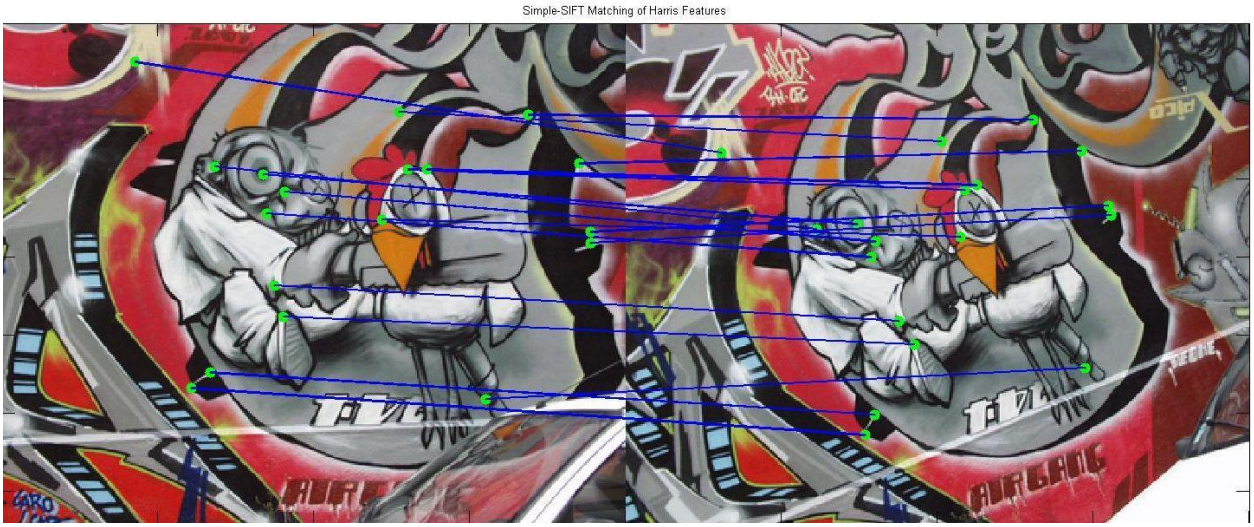
Descriptor Matching (15 points)

Instead of using the "Mutual Marriage" approach (described above), use the "Ratio Test" approach. In short, we take the top match and the second best match (according to the smallest L2-distances of the descriptors) and only accept this as a match if the ratio of the distance of the best/second-best is <= some threshold (typically 0.6-0.7 is a good choice here).Complete function **match_descriptors**.

Displaying Matches (15 points)

To display your matches in Python, given images I_1 and I_2, we construct a side-by-side image. Then, we go through all of our matches, and draw a line from the feature in the left image (I_1) to its location in the right image (I_2). Remember that when drawing this line, the feature location in the right image (I_2) should

be offset by the width of the left image (I_1) in the side-by-side view. Below is an example:



Complete function **draw_matches**. Ignore the "outlier_labels" argument for now. Draw the feature points as small filled green circles and the matches with blue lines using cv2.circle() and cv2.line().

Run your program on an image pair of your choice and display the results. Do not include the output images in your report yet. In the next question, you will use RANSAC to determine which matches are outliers. You will need to draw the matches again but use different colors to show inliers and outliers.

Notice that the function is only expected to **draw** a side-by-side image. **Any operation that displays images or writes images to disk should be done outside this function.**

Image Alignment (20 points)

Next, given a set of feature matches from two images, compute the **affine** transformation that best aligns the points together using the RANSAC algorithm to remove outliers.

Complete function **compute_affine_xform**. The output should be a 3-by-3 matrix that maps pixel coordinates in image 1 to coordinates in image 2. In other words, if A is the transformation matrix and x is the (homogeneous) coordinates of a location in image 1, then Ax should give you the corresponding location in image 2.

In addition to the transformation matrix, the function should also output a list of Boolean values indicating whether each input match is an outlier or not. Modify function **draw_matches** so it plots the inliers and outliers in different colors when "outlier_labels" is given. Specifically, draw inlier matches with blue lines and outlier matches with red lines. In this way we can see if the RANSAC algorithm worked well.

Go through all of the image pairs listed above (8 of them), run the feature detection/matching pipeline and display your side-by-side results. In your write-up, report the transformation matrices estimated by your function, and include all visualization results.

Image Stitching (20 points)

Now that you have estimated the transformation between the image pair, you can stitch them by warping one image to the other and see how well the transformation was computed.

Complete function **stitch_images**. Using cv2.warpAffine() and/or cv2.warpPerspective() is allowed. Before you implement this function, think carefully about which image should be warped to which, given the transformation matrix you estimated. For better visualization, average the pixel intensity in the overlap region so that you can see patterns from both images.

Show this for all 8 pairs of images and comment on how well the alignment worked for each in your write-up. If the alignment looks poor, you should state why you believe this happened. (*Hint: look at the geometry of the scenes)*.