University of Michigan
EECS 442: Foundations of Computer Vision
Fall 2021. Instructor: Andrew Owens.

Problem Set 9: Panoramic Stitching

Posted: Wednesday, Nov. 17, 2021 **Due:** Wednesday, Dec. 1, 2021

Please convert your Colab notebook to a PDF file and submit the PDF file to Gradescope, *making sure to label your answers*. We have included the PDF conversion script at the end of the notebook. Nothing needs to be submitted to Canvas. **Credit:** This problem was originally developed by the course staff for MIT 6.819/6.869.

The starter code can be found at:
https://colab.research.google.com/drive/1yWUXwOnzKaC4MAQxCZ8wj8zLyhuwZXsW?usp=sharing

We recommend editing and running your code in Google Colab, although you are welcome to use your local machine instead.

Problem 9.1 *Panoramic Stitching*

In this problem we will develop an algorithm for stitching a panorama from overlapping photos (Figure 1), which amounts to estimating a transformation that aligns one image to another. To do this, we will compute ORB features[1] in both images and match them to obtain correspondences. We will then estimate a homography from these correspondences, and we'll use it to stitch the two images together in a common coordinate system.

In order to get an accurate transformation, we will need many accurate feature matches. Unfortunately, feature matching is a noisy process: even if two image patches (and their ORB descriptors) look alike, they may not be an actual match.

To make our algorithm robust to matching errors, we will use RANSAC, a method for estimating a parametric model from noisy observations. We will detect keypoints and represent descriptors using ORB. We will then match features, using heuristics to remove bad matches. We have provided you with two images (Figure 1) that you'll use to create a panorama.

(a) You will start by computing features and image correspondences.

- **(2 points)** Implement `get_orb_features(img)` to compute orb features for both of the given image.

[1] ORB is a hand-crafted feature similar to SIFT. Until recently, SIFT was unfortunately patented, and was therefore difficult to use in OpenCV.

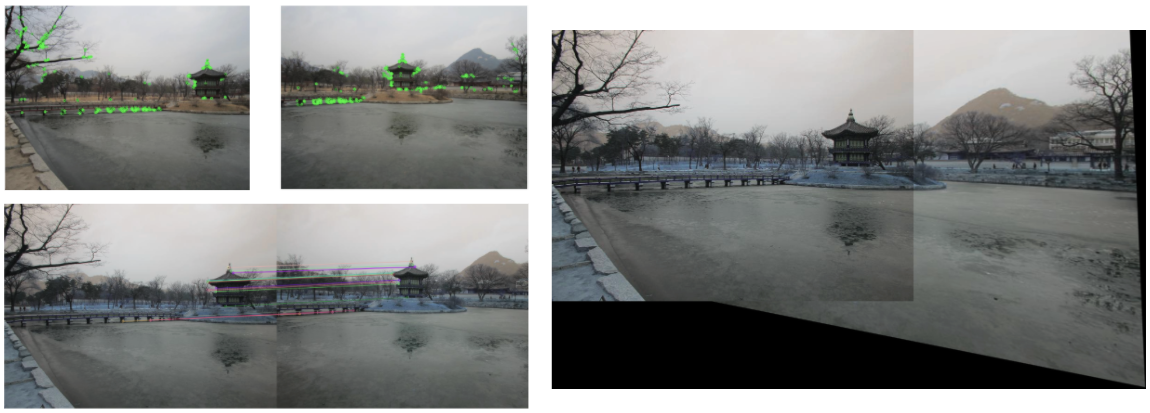

Figure 1: Panorama produced using our implementation. The image pair shown on the left represents the keypoints in the two source images and below them are the predicted feature correspondences. On the right is the stitched panorama.

- **(2 points)** Implement `match_keypoints(desc1, desc2)` to compute keypoint correspondences between the two source images using the ratio test. Run the plotting code to visualize the detected features and resulting correspondences.

(b) **(2 points)** Write a function `find_homography(pts1, pts2)` that takes in two $N \times 2$ matrices with the x and y coordinates of matching 2D points in the two images and computes the 3×3 homography H that maps `pts1` to `pts2`. You can implement this function using nonlinear least squares (or, alternatively, the direct linear transform).

Hint: For nonlinear least squares, we reccomend using `scipy` library's built-in nonlinear least squares. To use that function, you need to:

- Define a cost function, $f(h; \texttt{pts1}, \texttt{pts2})$, that calculates the projection error (a vector of length $2N$) between `pts1` and projected `pts2` using homography H as $\texttt{pts1} - \texttt{cart}(H * \texttt{homog}(\texttt{pts2}))$. Here `homog`(x) converts x into homogeneous coordinates, `cart`(x) converts x to cartesian coordinates and h is the flattened version of the homography H to be estimated.[2]
- Provide an initial guess for the homography h: a length 9 vector filled with '1's should be good enough.

(c) **(2 points)** Your homography-fitting function from (b) will only work well if there are no mismatched features. To make it more robust, implement a function `transform_ransac(pts1, pts2)` that fits a homography using RANSAC. You can call `find_homography(pts1, pts2)` inside the inner loop of RANSAC. You will also be responsible for figuring out the set of parameters to use to produce the best results.

(d) **(2 points)** Write a function `panoramic_stitching(img1, img2)` that produces a panorama from a pair of overlapping images using your functions from the previous parts. Run the algorithm on the two images provided. You result should be similar to that of Figure 1.

[2] Alternatively, you can try minimizing this loss using PyTorch and gradient descent. We didn't do this in our implementation, but it should have a very similar result — nonlinear least squares can sometimes provide a big improvement, but probably not for a simple problem like this one!