Zachary (Zack) Crenshaw
Prof. Maire
Introduction to Computer Vision
February 9, 2020

Homework 2: Interest Points and Feature Descriptors

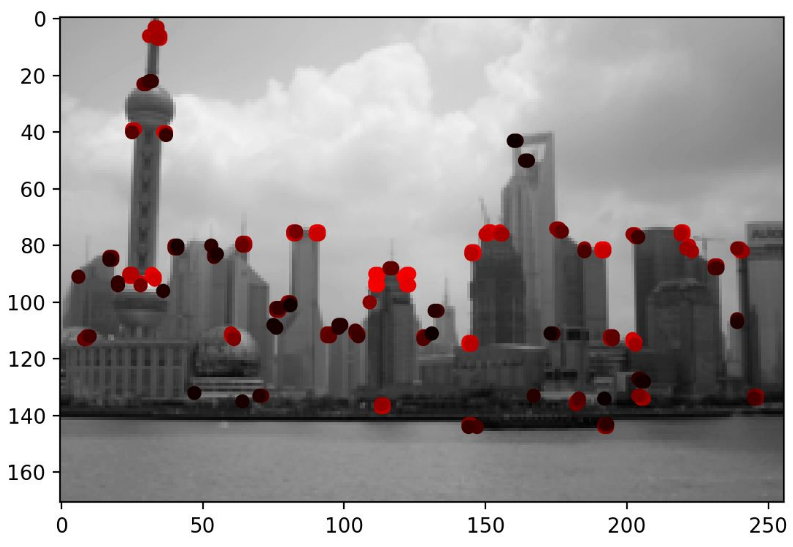
Feature choices:

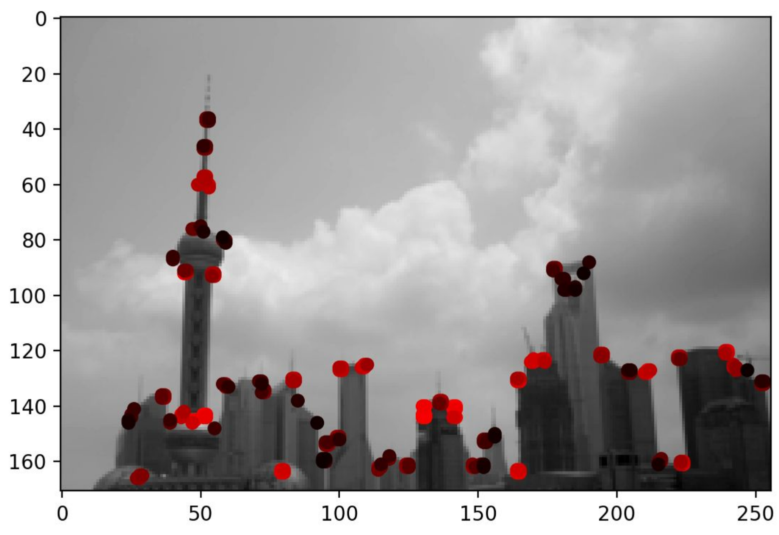
Finding interest points

I used a Harris Corner Detector (using Sobel Gradients as the gradients) to find the interest points. I did not use any windowing/change of scale, as this did not appear to improve performance by a significant margin. I tried to use a Gaussian smoothing on the image before finding the interest points, as once referenced the source material, but this only appear to make things worse. I used a k value of 0.04, and a threshold of 0.001 to find the points of highest interest. All points over 0 should be corners, so this small threshold just prevents strange edge cases from seeming to be of interest, without knocking out important corners. Perhaps if the max number of points is increased this might decrease performance, but with a max threshold of 200 points, these parameters gave the best performance. For the non maximum suppression, I looked in a 5x5 window around the interest point, and took the interest point with the highest score.

In the end, there is some slight clustering of points near corners, but these did not appear to affect performance as all the major corners are still covered in most frames of reference (with respect to the different views for each panorama). Increasing the window of the non max suppression deleted some points that were deemed by human inspection to be important from the top 200 list, and a smaller window made the clustering worse.

Example of output for shanghai-23 and shanghai-24:





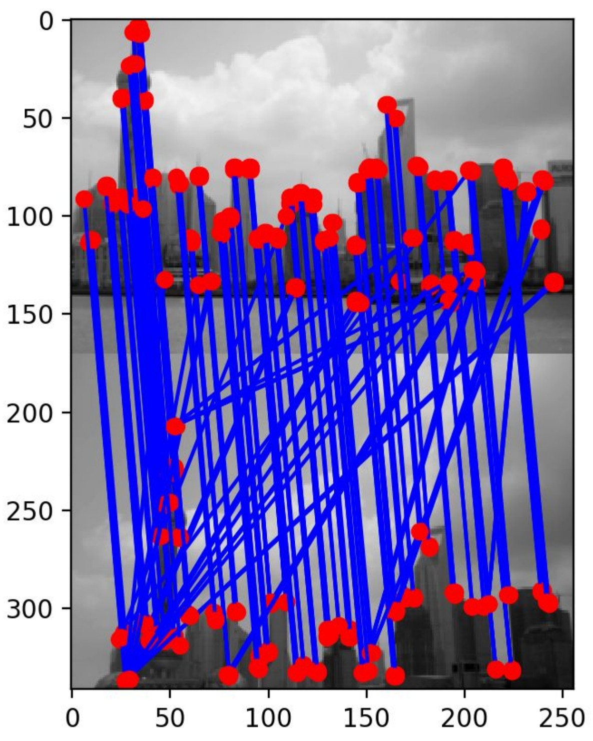
Extracting features

For the feature descriptor, I did as suggested, with a 3*3*8 feature vector for each interest point. I took the orientation and magnitude of the points around the feature points found in the same manner as in the canny edge detector (mimicking the “mag” and “theta” arrays), binned from 0 to 2π, in intervals of π/4. The bin width was empirically the best at a width of 5. I tried smoothing before feature extracting, thresholding the magnitudes, normalizing the magnitudes, and both thresholding and normalizing. All of these seemed to decrease accuracy.

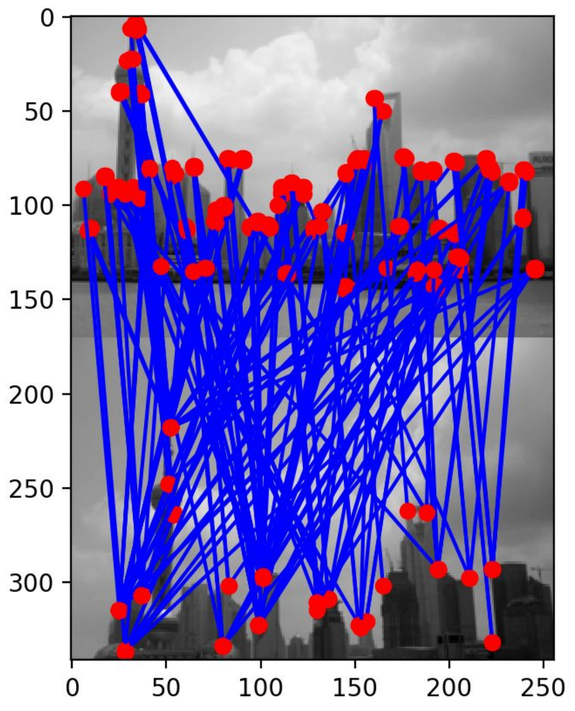
Matching features

To match features, I implemented the native and kdtree methods. I matched based on least squared distance in the 72 dimensional space. For the similarity score I used the ratio of the nearest neighbor distance to the second nearest neighbor (NN distance squared / SNN distance squared). The naive matching takes around 20 seconds, with some time performance improvements made by shortcuting out of the distance function if we are already over the distance to the second nearest neighbor. The kdtree operates in about 0.5 seconds, which is considerably faster, with only some decreased accuracy. The matching does not appear to be as good, but the Hough transform ends up being only slightly worse than the naive method, and much, much, faster. When building the kdtree NN search function, I found that for the inequality in line 471 that decides the order of the search, that acurract was significantly increased if "diff <= 0", not "diff < 0", meaning that the left bin is searched first if the difference between the points on that axis is 0.

Naive matching:



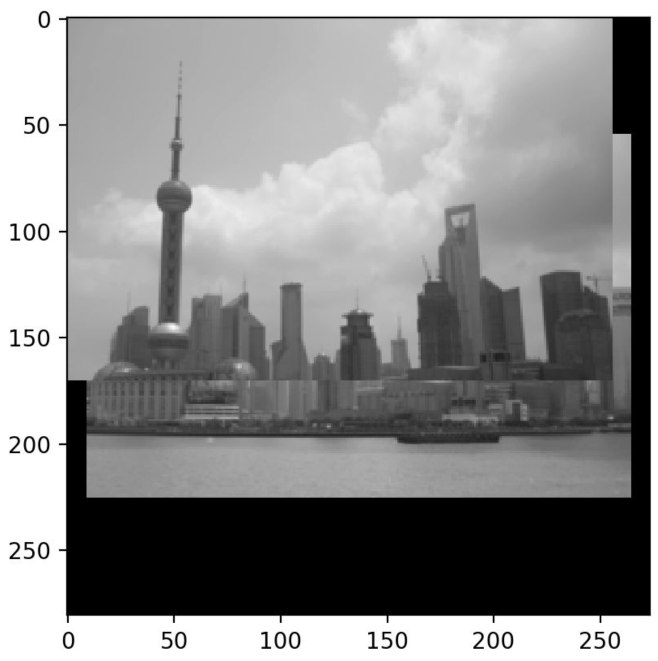
Kdtree matching:



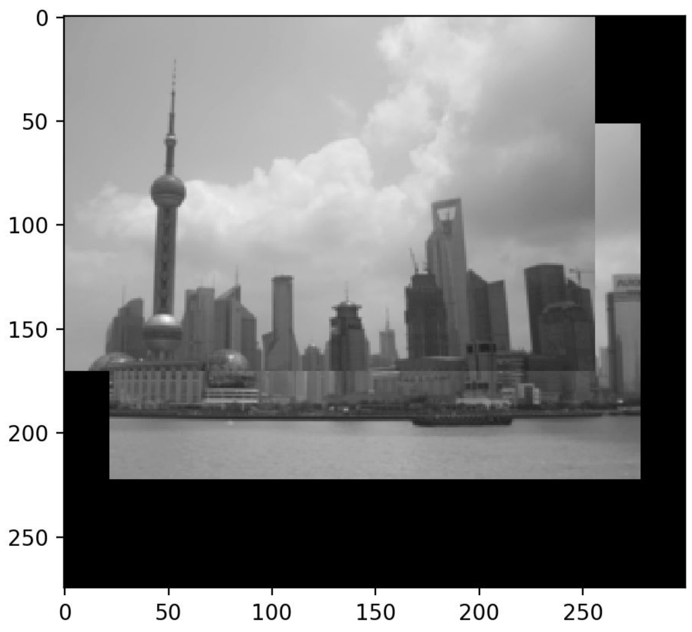
Hough transform

To discretize the translation parameter, I computed the translation between all the matches, and found the greatest and least values on both x and y axes, and created what is more or less a histogram for each axis. Empirically I found that 60 bins gave the best performance without slowing things down considerably. Each match gets a vote equal to its score (NN distance squared / SNN distance squared).

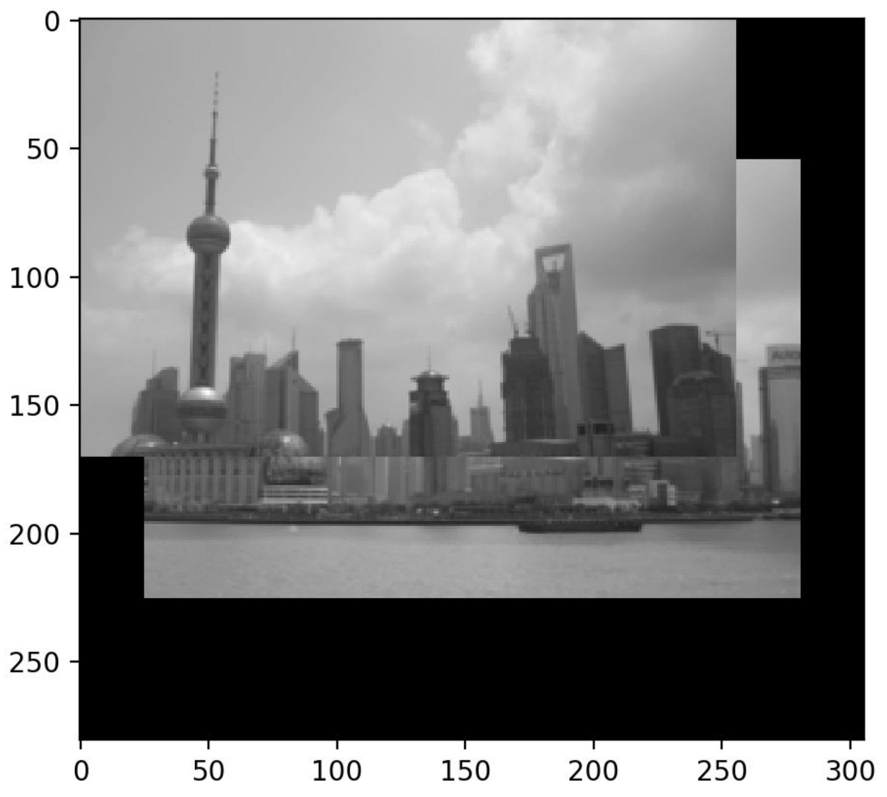
Simple Hough transform (on naive, 10 bins):



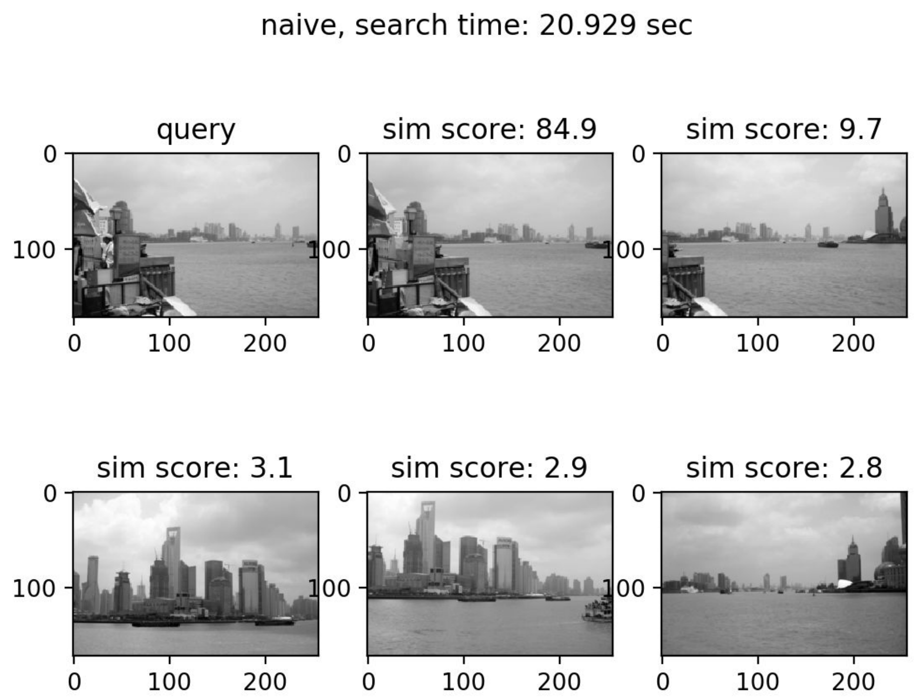
Hough transform using scoring and 60 bins (on naive)

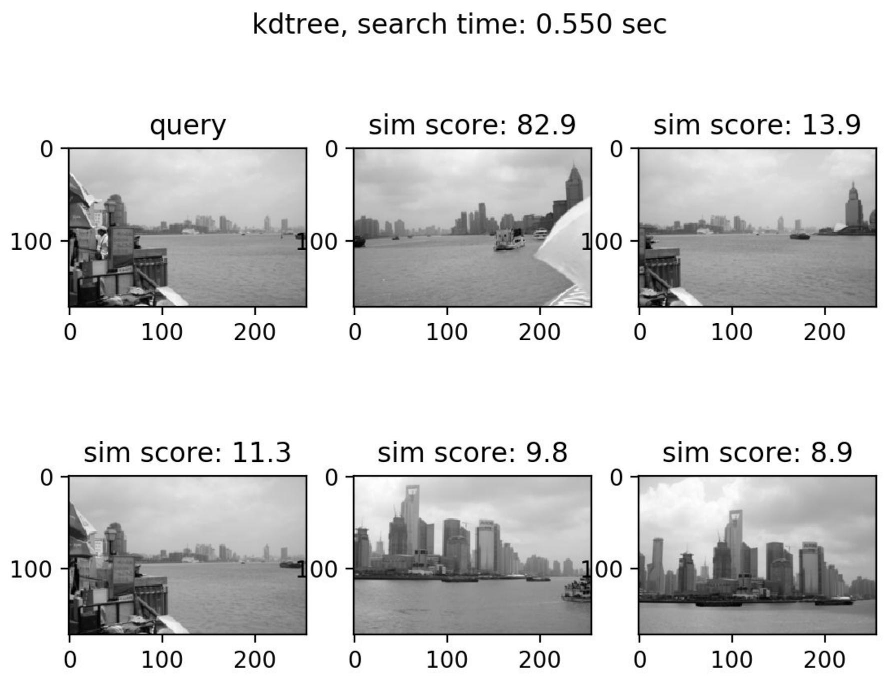


Hough transform using scoring and 60 bins (on kdtree)



Performance comparison:


naive, search time: 20.929 sec
query
sim score: 84.9
sim score: 9.7
sim score: 3.1
sim score: 2.9
sim score: 2.8


kdtree, search time: 0.550 sec
query
sim score: 82.9
sim score: 13.9
sim score: 11.3
sim score: 9.8
sim score: 8.9