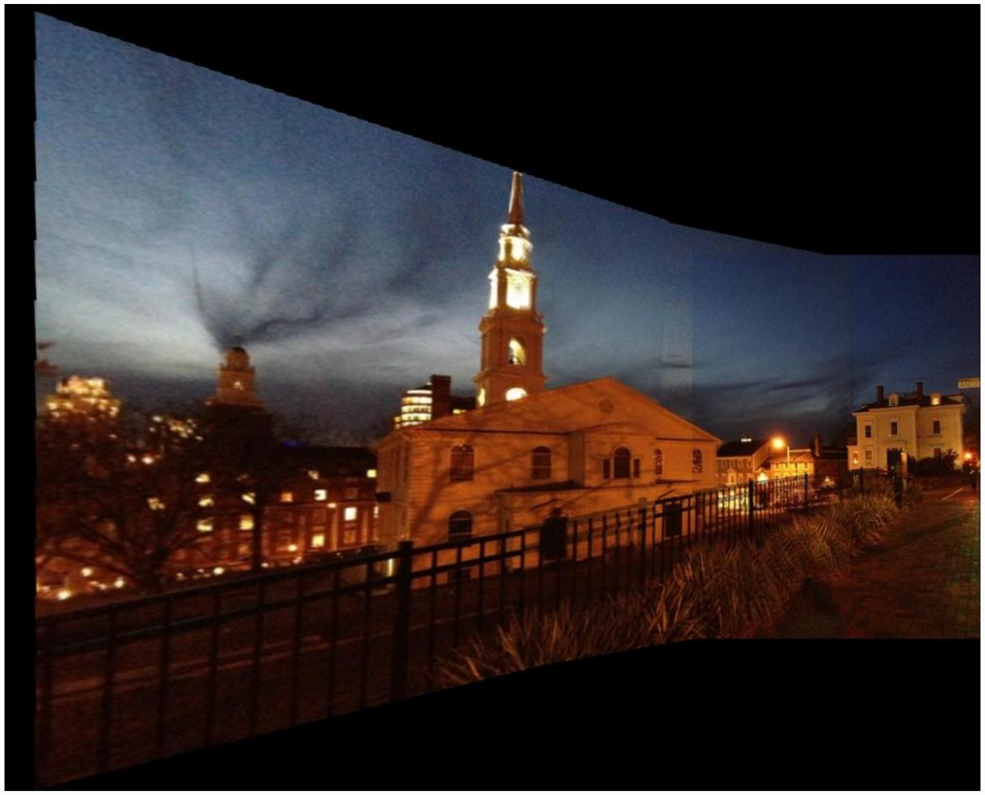
ian strickman (istrickm)

CS129 / Project 6 / Automated Panorama Stitching

This project involves stitching together images to create panoramic shots of scenes. To stitch two images, you must first identify a homography from one to the other, and then composite the two images based on that homography. For multiple images, you stitch two and then stitch the next one with the result from the first stitching, and so on. For this assignment, the composition has been done for us, so we must identify the homography. The way we do this is via feature detection and matching followed by RANSAC.

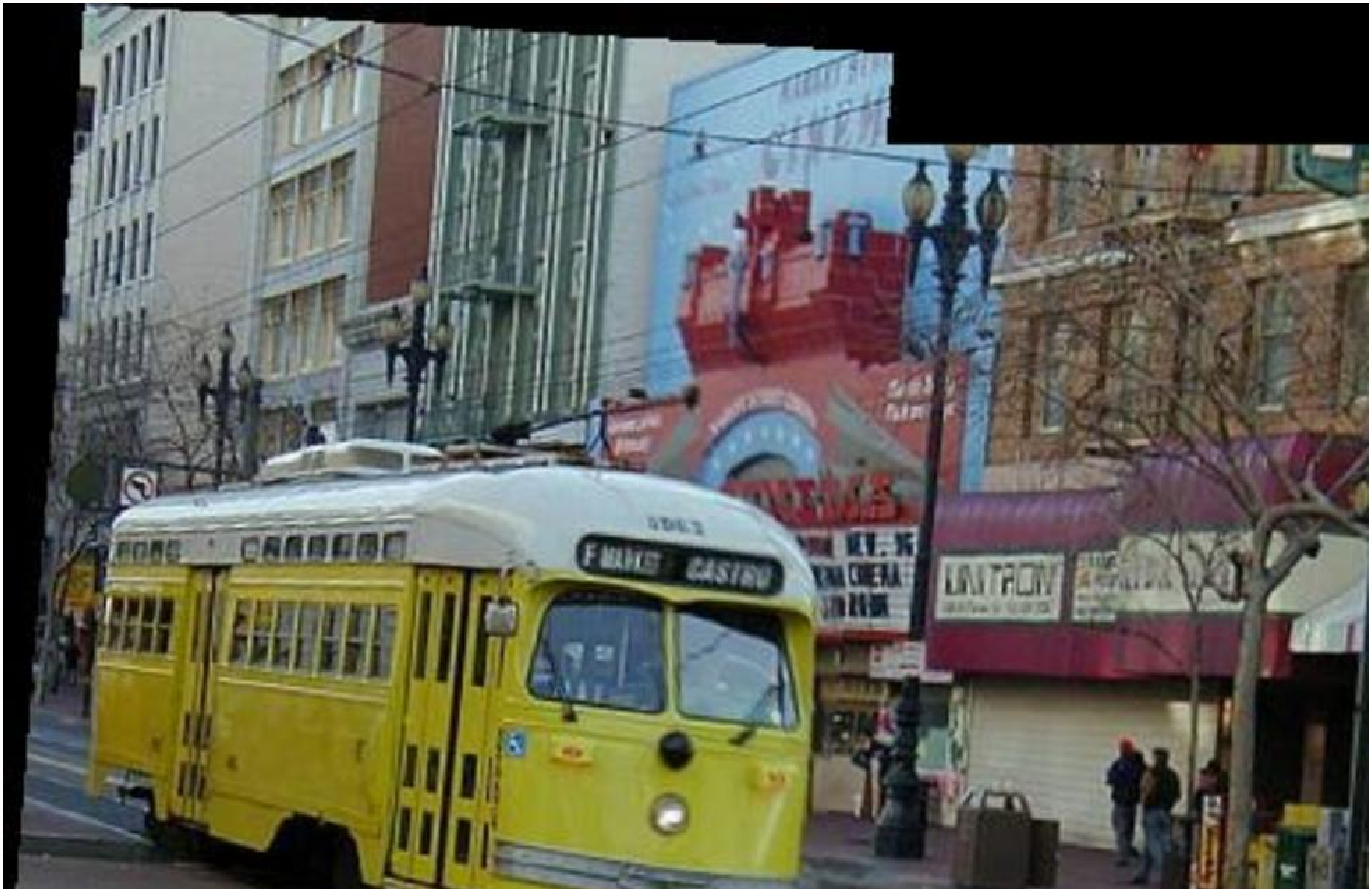

Church near RISD in the evening

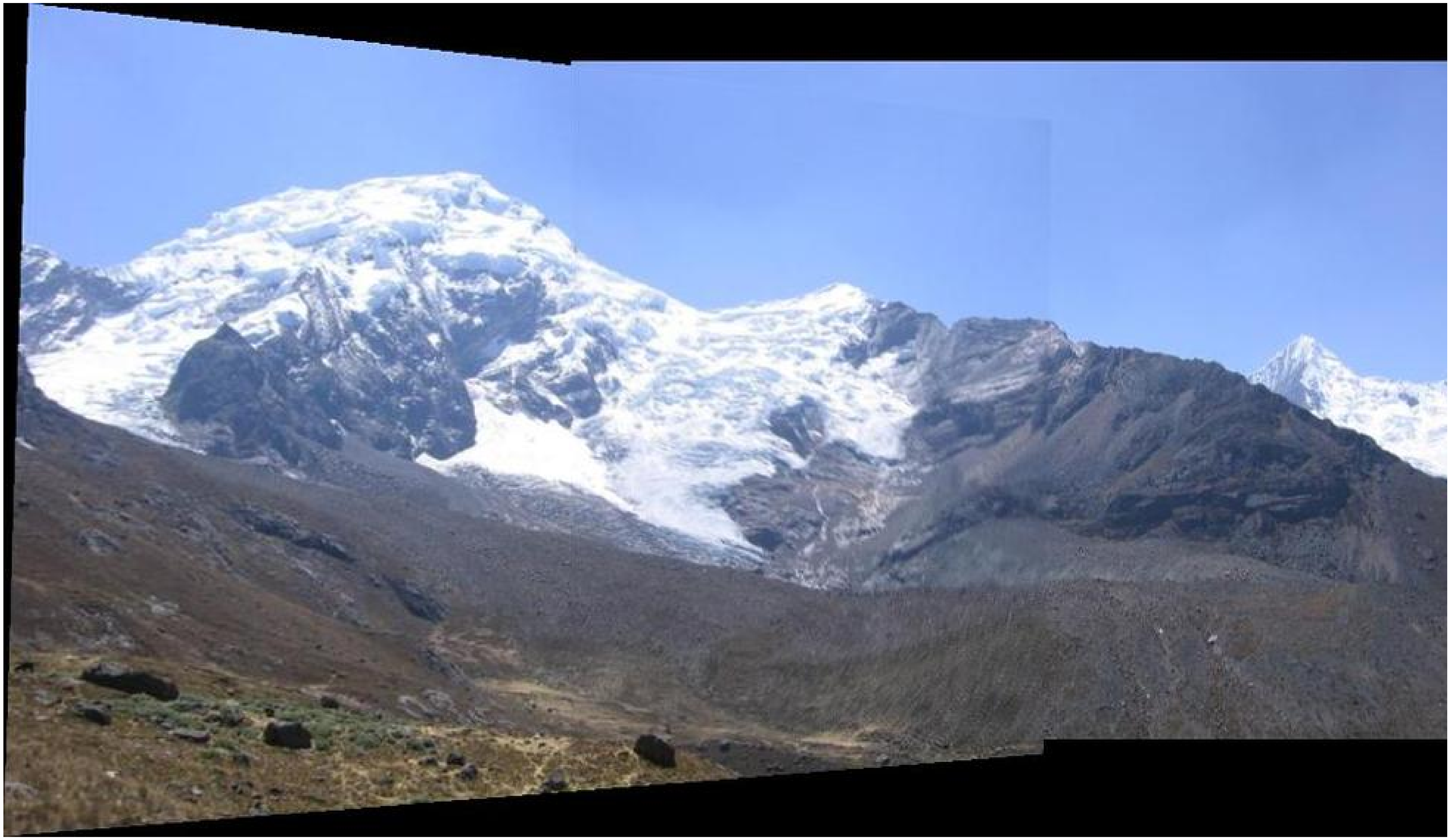
Algorithm

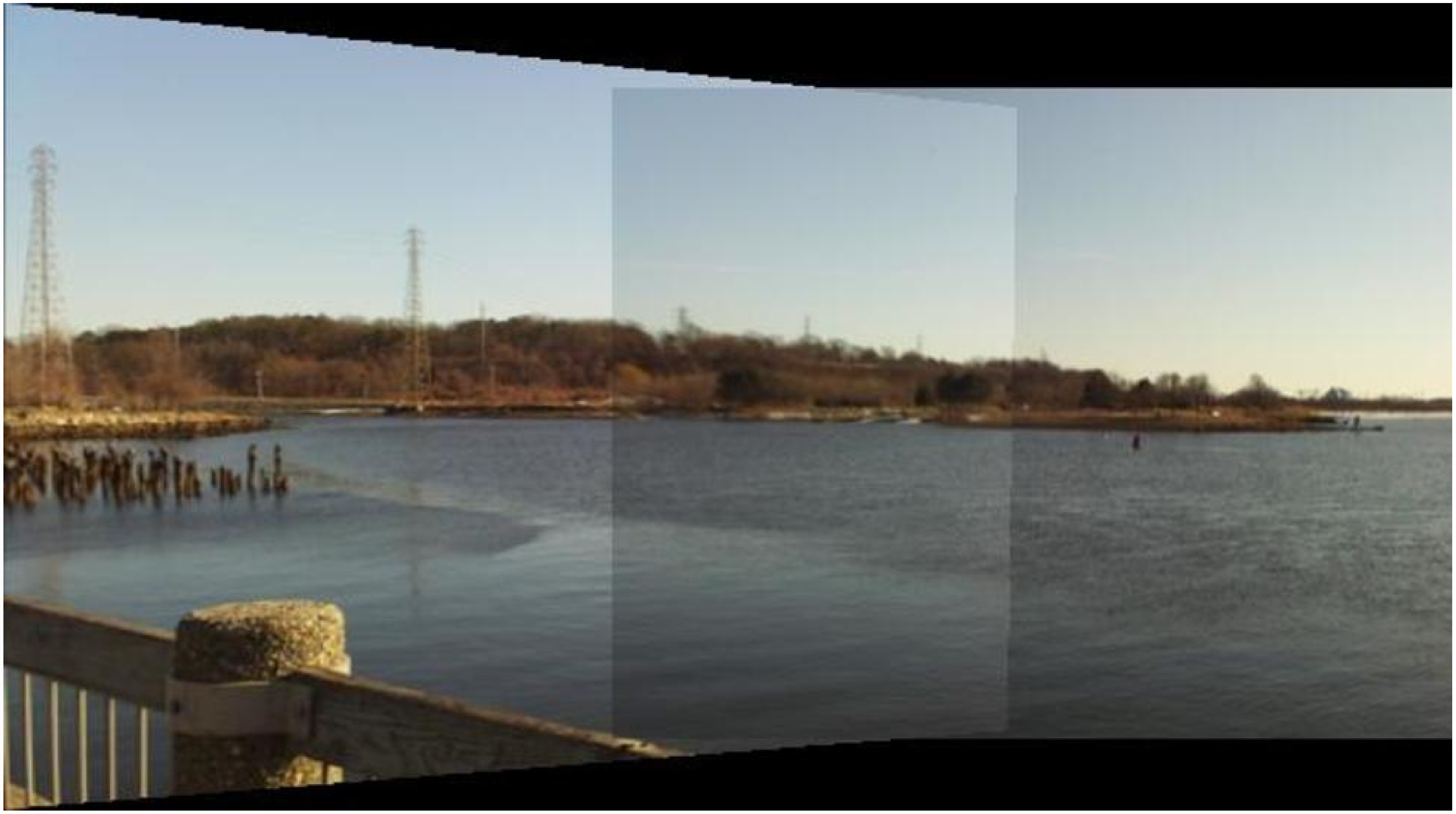
For the first part of this algorithm, we're following a modified version of that in a paper by Brown et al.. The idea is to first detect interest points in the two photos to stitch using the harris corners feature detection algorithm. This algorithm detects where there are corners in the image, that is where there are points such that the difference between a patch centered on that point varies widely with patches centered on nearby points. The useful aspect of this algorithm is that it will detect a lot of correct features, so we will be able to find correspondences reasonably well. Unfortunately, the algorithm also comes up with a lot of false positives, and by the nature of the problem some of the features in one image won't exist in the second. We take these issues into account when we do our matching algorithm. The first step of this is to get rid of feature points that are too close to one another, picking the higher gradient points over the lower ones. Once we have pruned in this way, we go through and extract a feature description from the patch surrounding each point. If we are using 40 pixel wide patches, then the feature description will consist of an 8x8 grid of grayscale pixel values that correspond to a resized version of the 40x40 patch in the original image. The idea with this patch is that it is spatially invariant, so it will be the same (or very similar) in other images. To account for variations in the lighting conditions of the images, we normalize the feature descriptors so that they will not vary based on brightness or dynamic range of the image. Once we have these feature descriptors, we simply go through for each feature point in the first image and see if it matches any feature points in the second image. Our metric for matching is that the ratio between the best match and the second best match is less than some threshold. This means that if a feature from image A matches a feature from image B very well and the next best match isn't great, then we are reasonably confident that we have a match. On the other hand, if the matches in image B are all of similar value, then we aren't confident and we don't take that point.

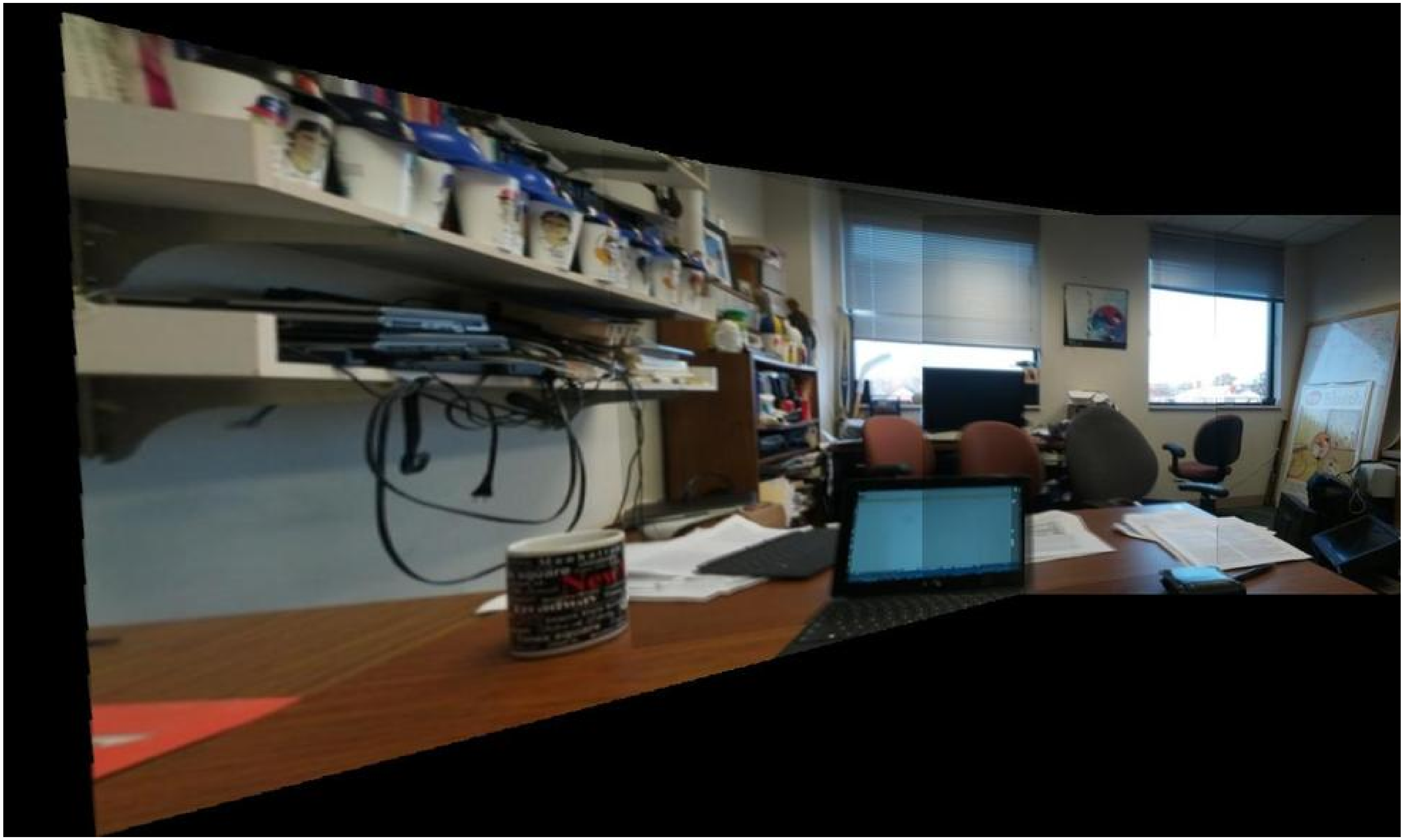
At this point we have a lot of matching points, most of which are probably reasonably correct. We now use RANSAC to determine a homography between the two images. Basically we pick four random pairs of points and generate a homography based on those points. Next we take that homography and apply it to every other pair of points and see how close that homography is to working for each pair of points (that is, if applied to the point in image A, how close is it to its pair in image B). We repeat this process many times and keep track of the homography where the most points matched up. After we finish repeating that process, we calculate a new homography using an overconstrained system made up of all the points we found in the best agreement. The idea here is that by randomly picking four points, we will stumble upon a reasonably close to true homography, and that when we do the inliers among our points will all agree, while some outliers will not. This works as long as outliers are outliers in different ways.

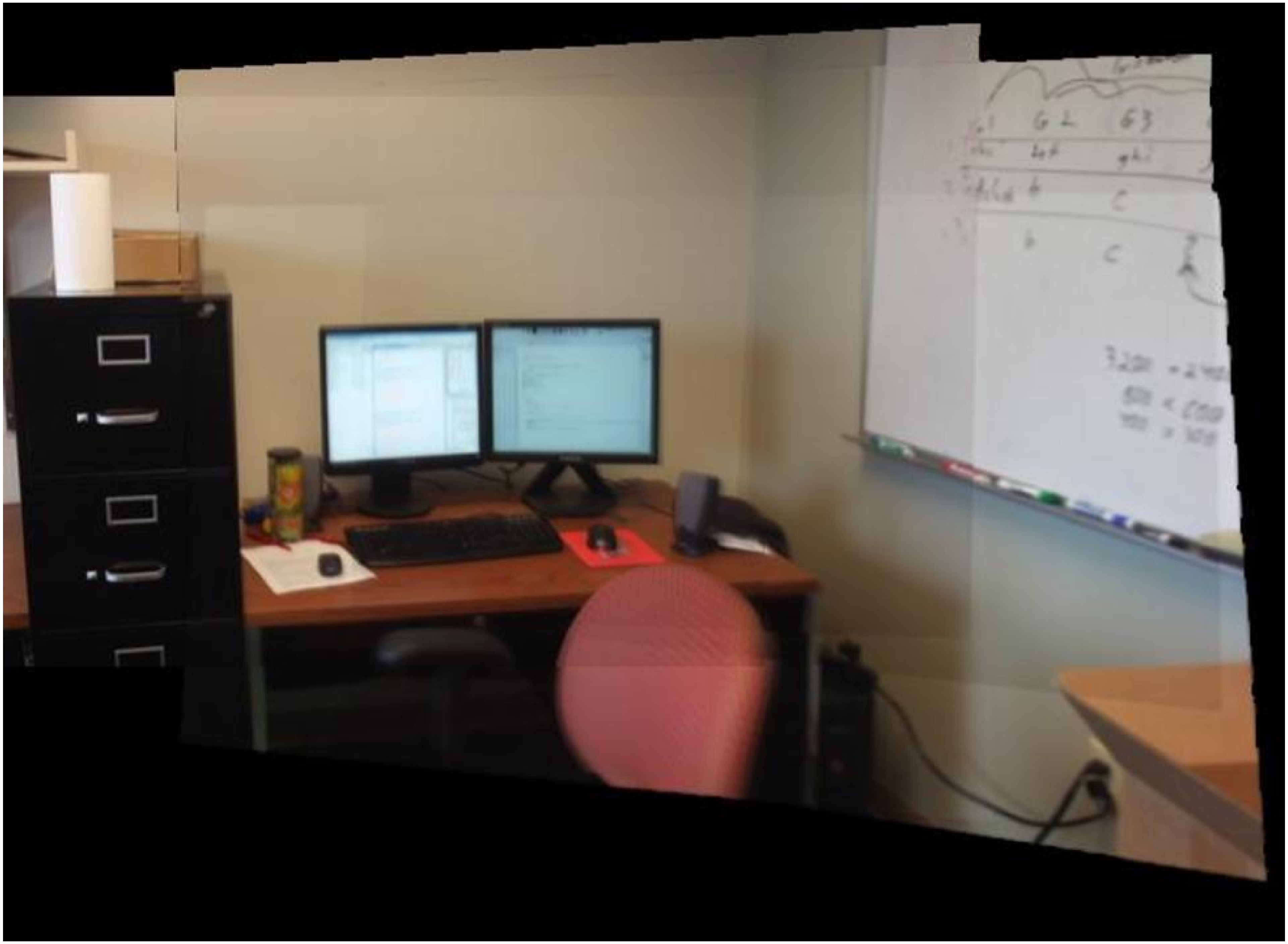
Results

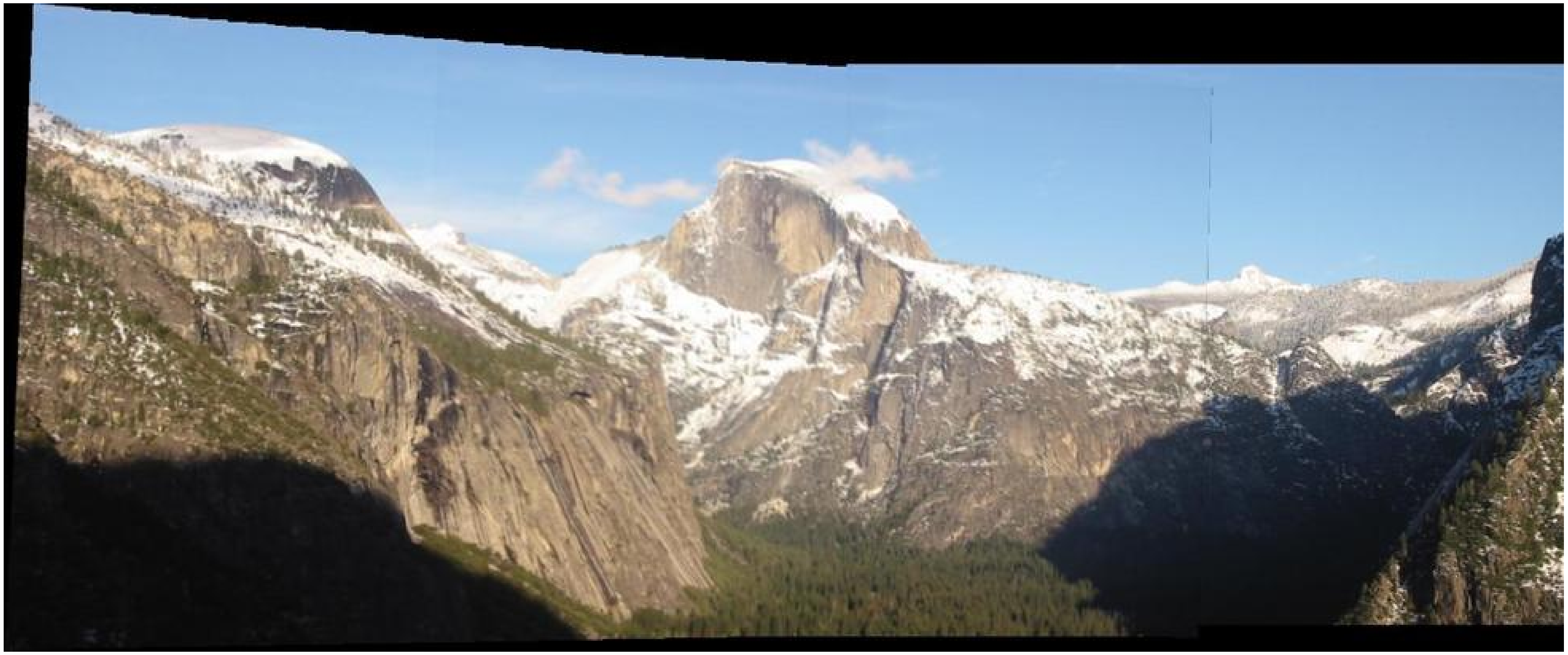


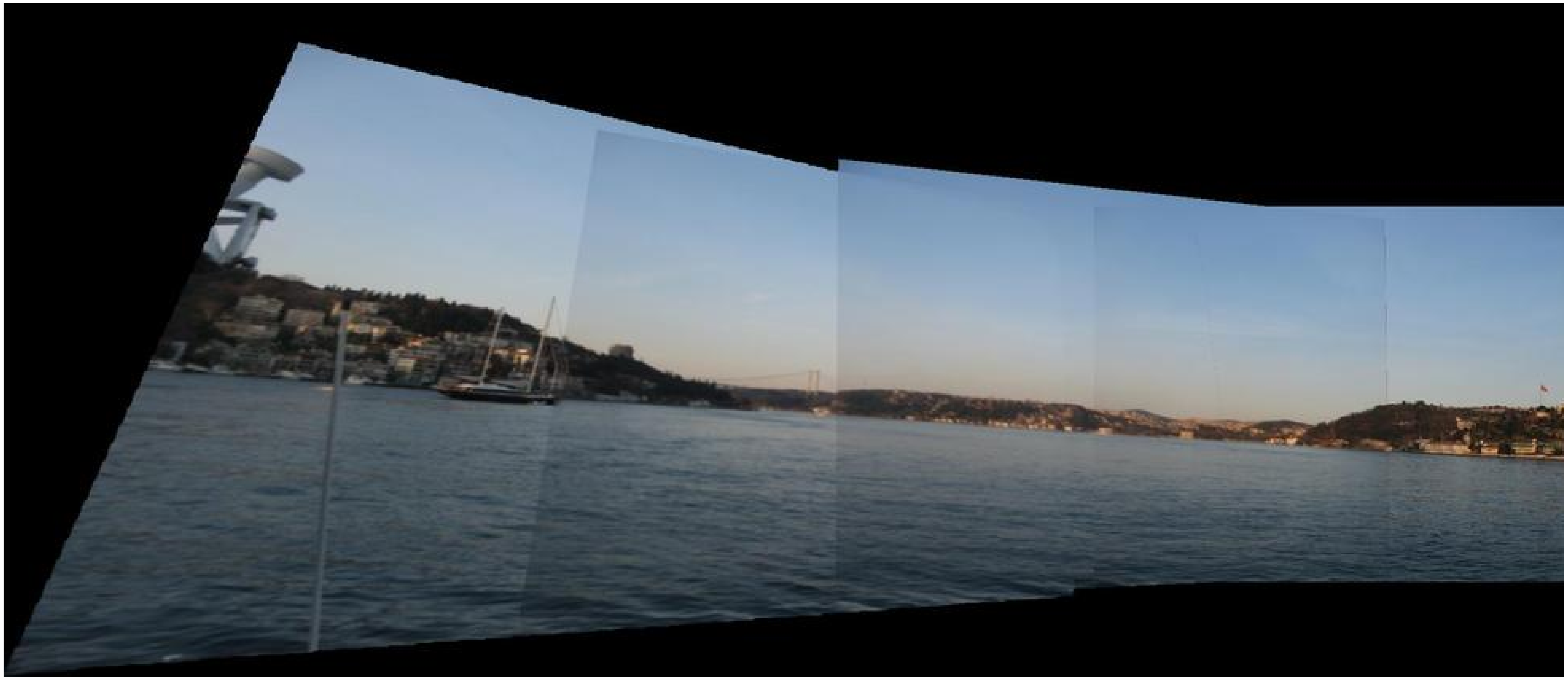


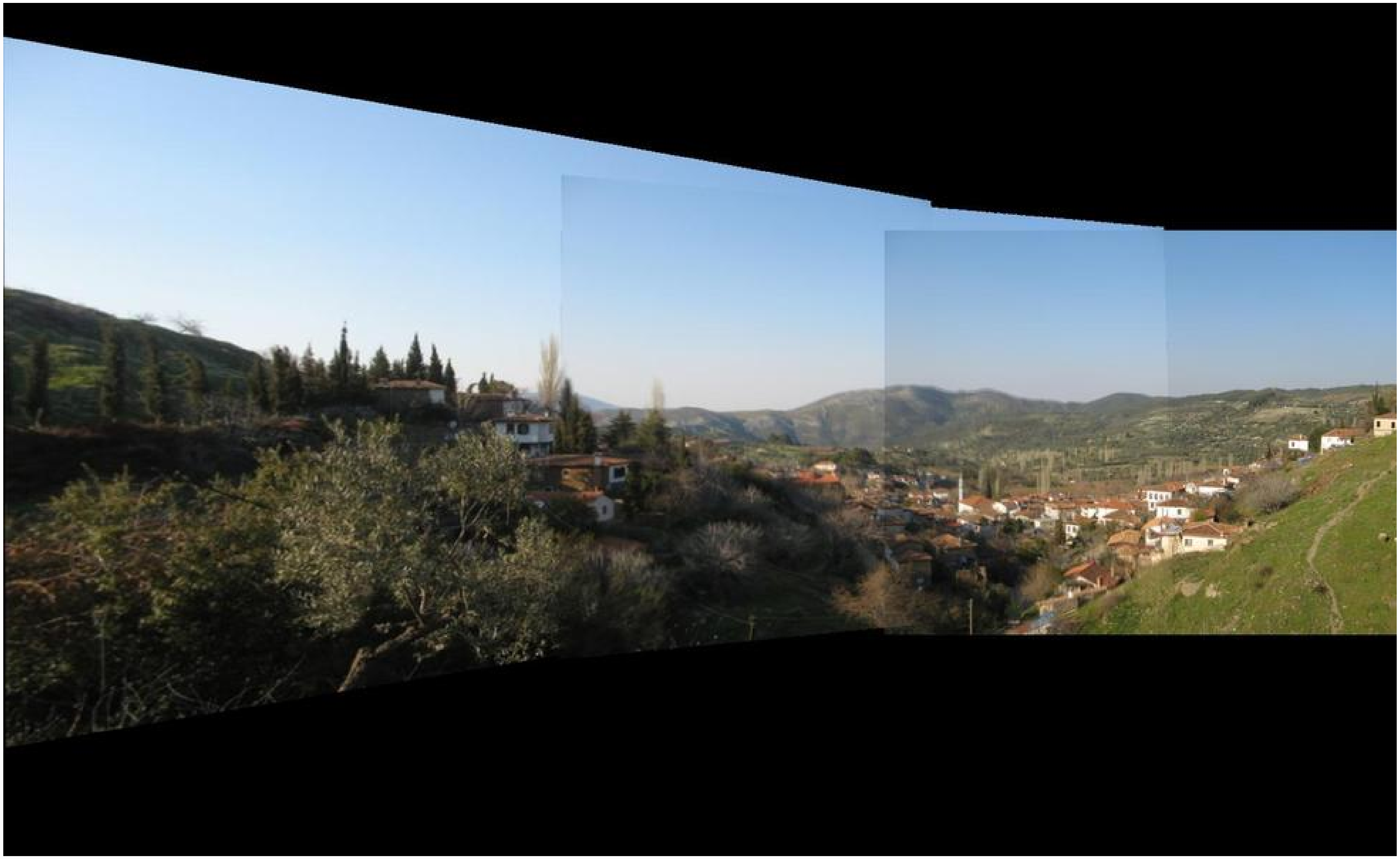


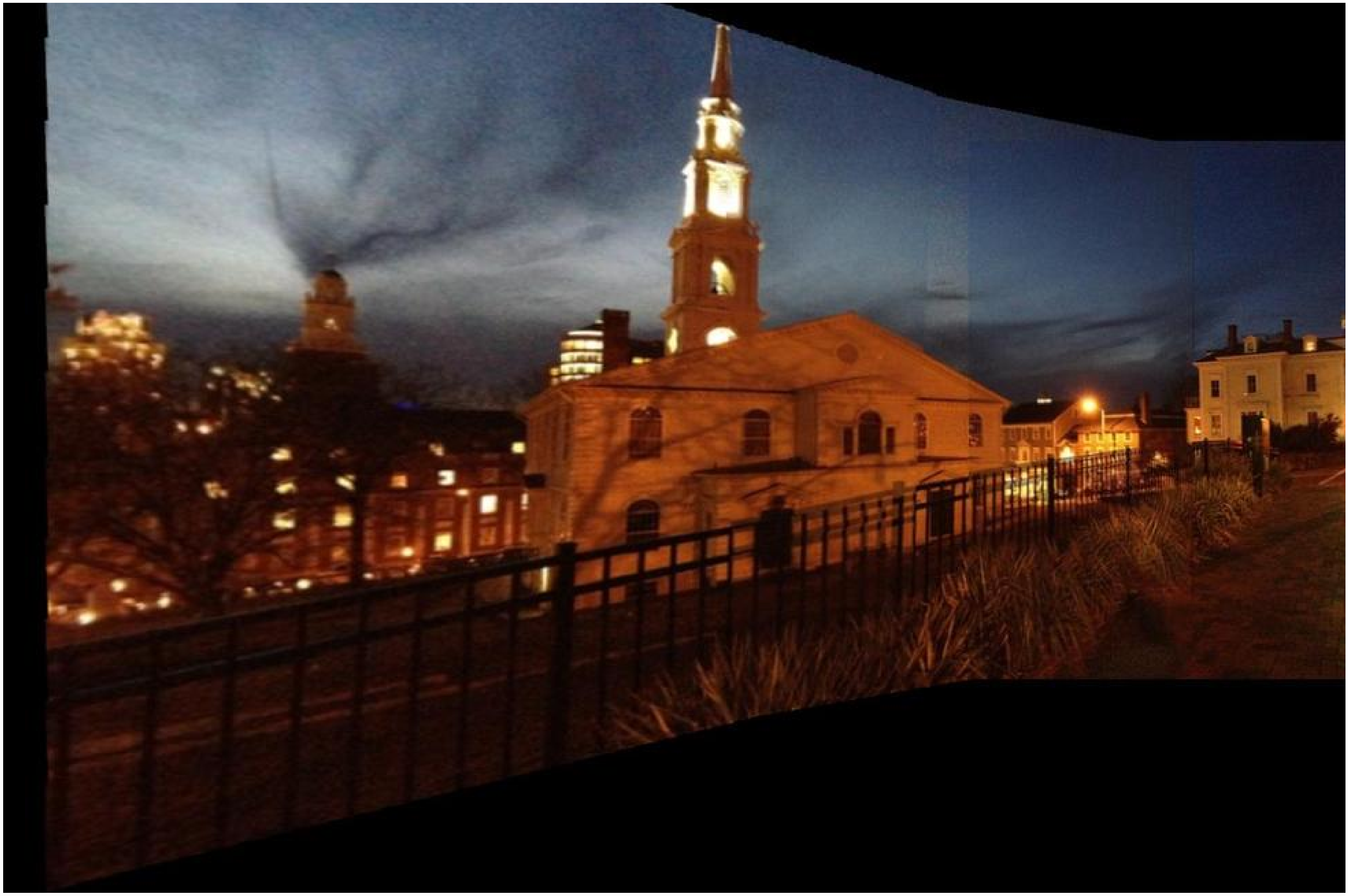


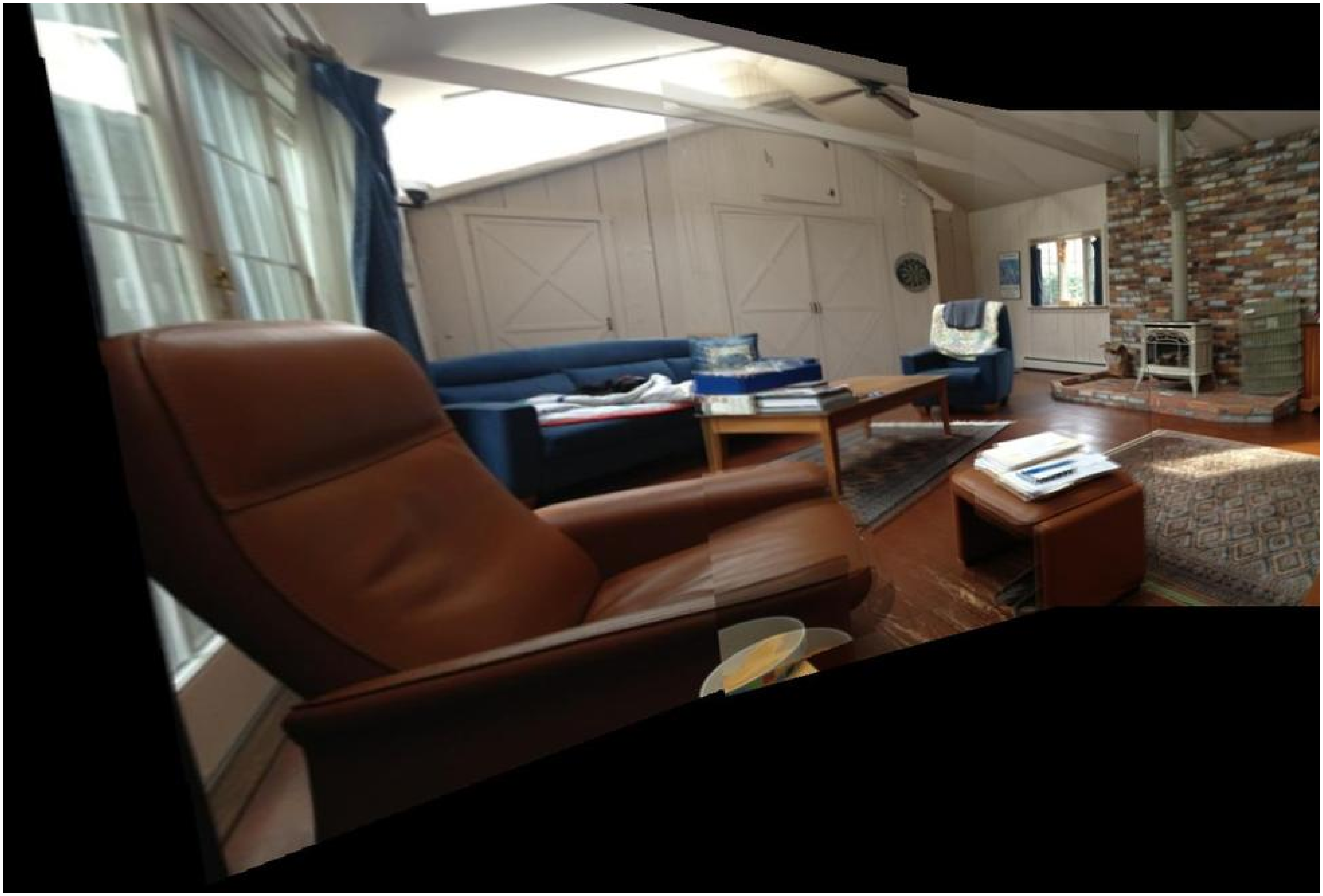












Discussion

This algorithm seems to work reasonably well, although it clearly worked much better for the given test images than for my own images. The weakness here is when the images are taken from slightly differing spatial viewpoints, as can happen if you don't have a tripod or you're taking pictures with your phone. The last two panoramas were created using images from my phone, and although they worked alright, you can see issues with, for instance, the bowl in the last image. Overall, though, the algorithm works quite well.