Overview

Andrew Showers

The first step for the program is to find the images in the directory specified by a command line argument. These filenames are sorted then read in by OpenCV. Each image has it's key points and descriptors detected using the OpenCV implementation of SIFT (xfeatures2d.SIFT_create and detectAndCompute). Now that the key points have been detected on all images, the program runs through an algorithm to determine the anchor. Each image is selected as a possible candidate for the anchor and is compared against all other images using the following steps (determined via empirical testing, discussion with others, and my own biases):

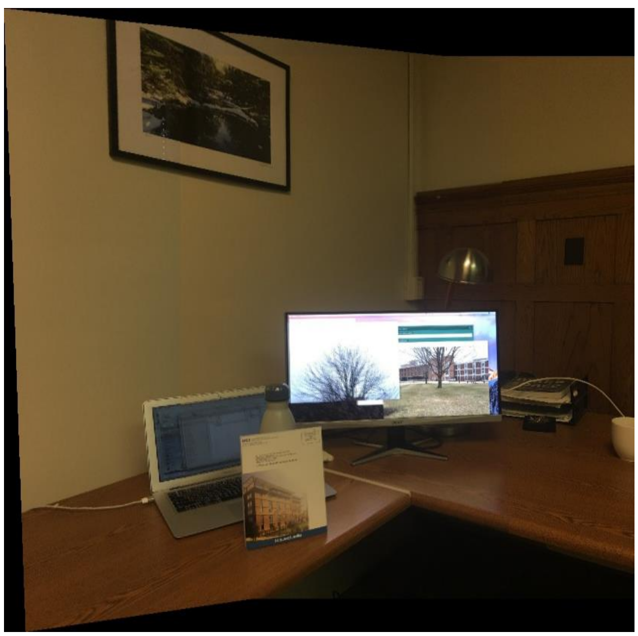
1. Get the keypoint matches between the two images (implemented using cv2.FlannBasedMatcher(): faster than brute force due to KNN tree searching)
2. For all matches, filter the matches based on the ratio test. My threshold for this test was set to 0.8, a commonly used number for this threshold.
3. For all remaining matches, filter the matches based on the F matrix estimation. This was done using findFundamentalMat() using the RANSAC algorithm.
4. For all remaining matches from the previous step, filter the matches based on the H matrix estimation. This was done using findHomography () using the RANSAC algorithm. A reprojection threshold was set to 4.0 after extensive testing. The official documentation recommends a value anywhere from 1-10, with the default = 3.
5. A ratio of the number of pixels survived from step 3 and 4 is used to determine the overall quality of the matches, and a threshold was set to 0.7, meaning that no more than 30% of the matches can be lost going from steps 3 to 4.
6. As a final threshold, the total number of matches must be greater than 30.
7. If the algorithm managed to reach this step without failing to pass any of the previous thresholds, the image has a valid matching pair and is an anchor candidate.
8. Anchor candidates are groups into two sections, those who have 1 matching image, and those who have 2 matching images. If there are any anchors with two matches, these have priority over single match anchors.
9. Anchor candidates in a single group are sorted by the total number of matches they have with other images. The one with the most matches is chosen as the best anchor.

Now that the anchor has been found, the images can be stitched together via the homography matrices calculated during the anchor selection process. The anchor is placed in the new coordinate system of the panorama (determined via perspectiveTransform()) and the other images are warped by their corresponding homography matrices via warpPerspective(). However, due to an issue I faced determining how to apply two homography matrices together for the second image overlay, I had to resort to recalculating key points and the homography matrix of the intermediate panorama step. This is highly inefficient and should be addressed in future versions. For blending, I added an alpha layer to the images as a way to determine any overlaps and on those locations I averaged the pixel values.

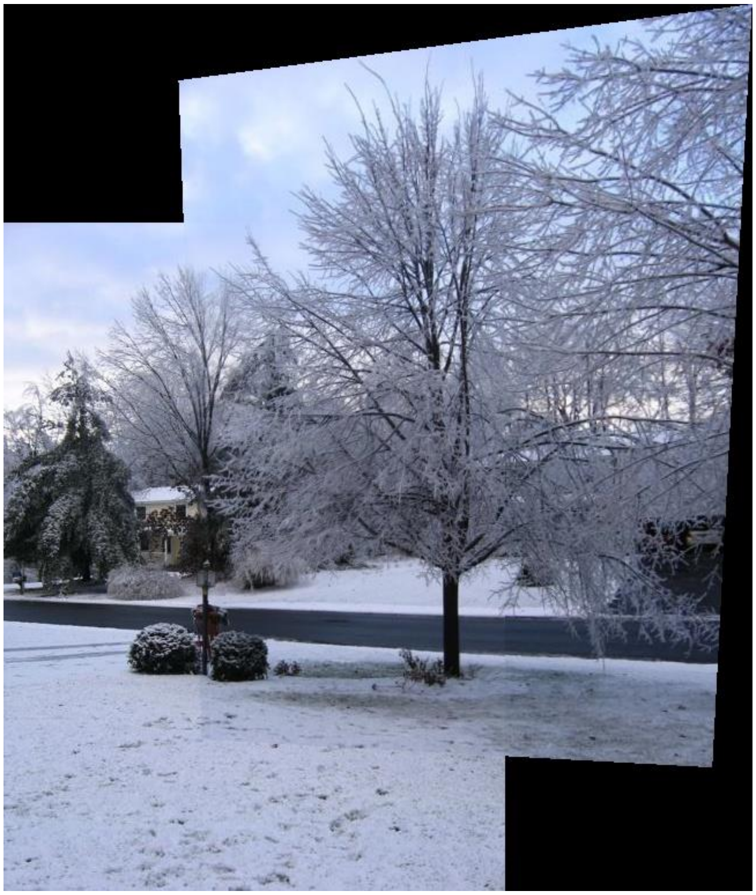
EXPERIMENTAL RESULTS

Most images from the samples provided were able to stitch with the exception of drink machine, which failed to pass step 5 in the list above. The best results were from the panoramas with two images due to the minimal amount of blending required. We can see examples below:

Frear Park



Office 2



Ice storm

VCC entrance

My Room

The largest issue with these examples is the blending. Since it is the average of the two image color intensities, there is going to be a clear change from image 1 - overlap - image 2. These borders could be improved by implementing Gaussian smoothing along the edges, as well as blending based on distance from the image centers. This improved blending would also help with differences in lighting as seen in the VCC entrance example.

As for triple overlay images, results were fairly good except for blending and the poor decision choice to blend all three images for office.

Church

Tree MRC

Office