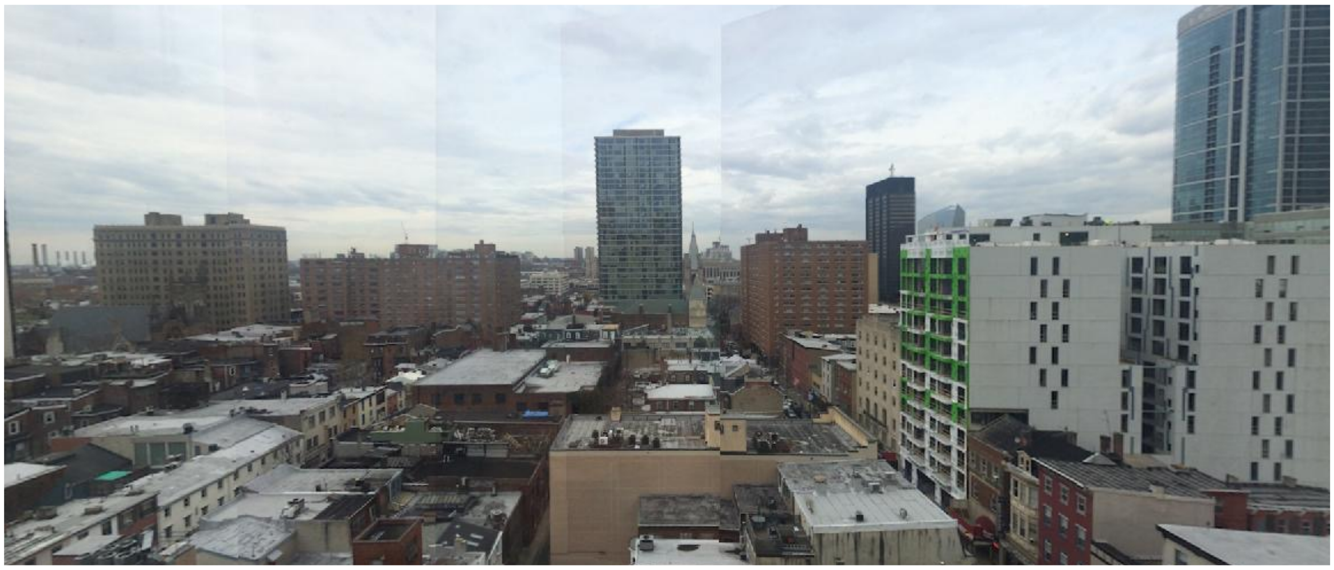
Automatic 2D Mosaic

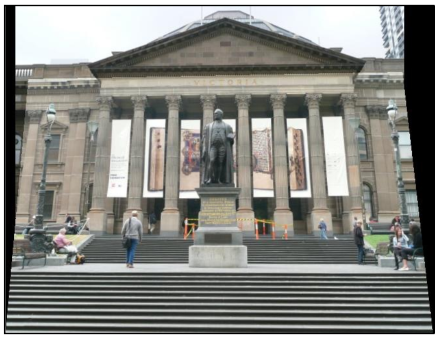
CIS 581 Computer Vision and Computational Photography

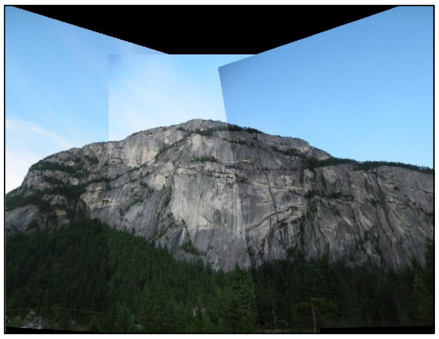
Sebastian Schloesser - November 21, 2014

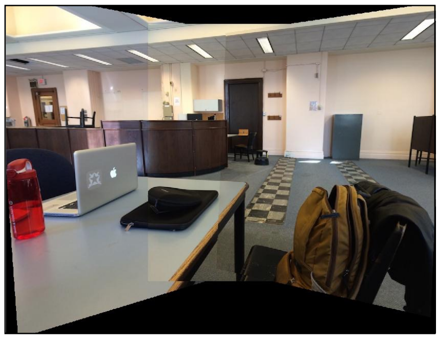


Introduction

This project was divided into several functions, each of which computed a critical set of data that the next would use. The project utilized concepts from previous projects we worked on as well. In some cases, I opted to use the Matlab version of the functions, even though I had implemented them in a previous project. An example would be non-maximum suppression. However, in other cases, I found my implementations to work better than those offered by the Matlab toolbox. In the end, the images came out very nicely, and many images can be handled if they are not too large in resolution. I reduced all the images from my iPhone to 560x750 pixels, which gave me no problems even with 6 images being stitched together.







From the intermediate images that I observed, it was clear that the RANSAC algorithm is a critical part of this process and that feature matching along can definitely not provide a robust and consistent enough result. If you lower the threshold on feat_match, the percentage of correct matches increases, but the overall number of matches found decreases significantly. It is more worthwhile to compute many points with feat_match and select the correct ones with RANSAC.

Function Definitions

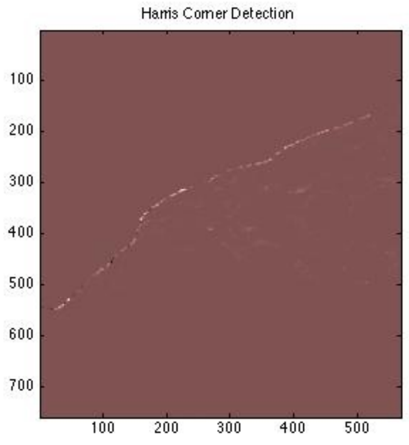
- Adaptive Non-Maximal Suppression
    - [y x rmax] = anms(cimg, max_pts);
- Extract Feature Descriptors
    - [p] = feat_desc(im, y, x);
- Descriptor Matching
    - [m] = feat_match(p1, p2);
- RANSAC

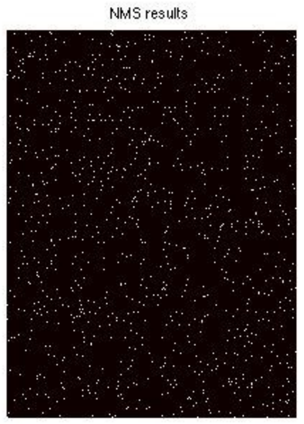
        - [H, inlier_ind] = ransac_est_homography(y1, x1, y2, x2, thresh);
- Stitching
    - img_mosaic = mymosaic(img_input);

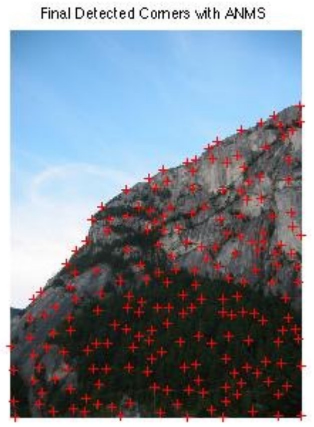
Detailed Function Descriptions and Process

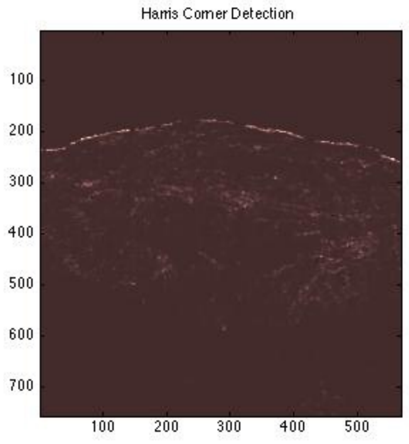
Adaptive Non-Maximal Suppression

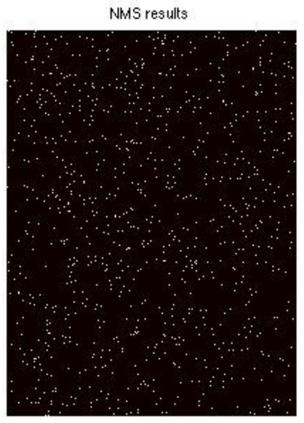
This function's goal was to take the Harris edges calculated by Matlab (using the cornermetric function) and to obtain an even distribution of key points on the image. These were the obtained results. The first column shows the results from the initial corner detection, which shows very clustered and poorly distributed points. After performing initial non-maximum suppression, a fair distribution is achieved, as shown on the second column. However, the points are too dense and don't represent good features. For example, there are too many points in the sky. After data processing and applying ANMS, the points on the third column are achieved. As seen by the original image in the background, they represent features fairly well.

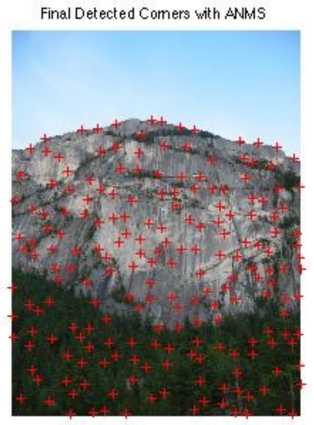












One important thing I did to achieve the good results shown above was simply removing the weakest values of the NMS results before applying ANMS. I tweaked the percentage of points to remove and found that .001% was idea—it removed most of the points in the sky and that were not going to make good features. I used the following code.

```
% Remove the least cornery corners
mx = max(max(cimg));
mn = min(min(cimg));
thresh = (mx-mn)*0.00001;
cimg(cimg < thresh) = 0;
```

Extracting Feature Descriptors

This function was quite straight forward. However, it is worth mentioning that to handle the case where descriptor squares came out of the picture's frame, I simply discarded these features. I used a try catch statement to simplify the code, which basically skips a loop when an out of bounds exception is raised. This results with zeros in the column.

Matching Features

For this function, I spent some time vectorizing the code to move the process faster. As I did with the previous project, I used repmat and kron to repeat each column in p1 and then repeat the entire p2 as many times as there are points. Then I was able to simply subtract the two matrices and perform the squaring operation to all of these values. Then, I summed the columns, which gave a very long row vector. I converted this vector into a matrix so that each row represented a point in p1, and every column, the match value of that point from the row vs the point in p2 corresponding to that column. I performed sorting and divided the first 2 columns to obtain the ratio to be compared with a threshold of 0.7, which gave good results.

RANSAC

As I mentioned before, RANSAC was a key component of this program. The following images will show the matched features from feat_match in yellow lines. The green lines are the inlier matches after performing the RANSAC algorithm. It is evident that the feat match produces a fair number of incorrect matches and RANSAC is very robust at finding the correct homography. With the resolutions I was working with, it wasn't very long to run this algorithm so I chose to run it 500 times for each image to ensure that the best set of inliers was chosen. I also played around with the threshold and found that 1 did a pretty good job. To avoid using excessive amounts of memory during the 500 iterations, I simply stored the indexes of the 4 points used to obtain each homography and the number of inliers they resulted in. At the end, I recomputed the homography using the 4 points that created the most inliers and then, computed the homography once more using the entire set of inliers.

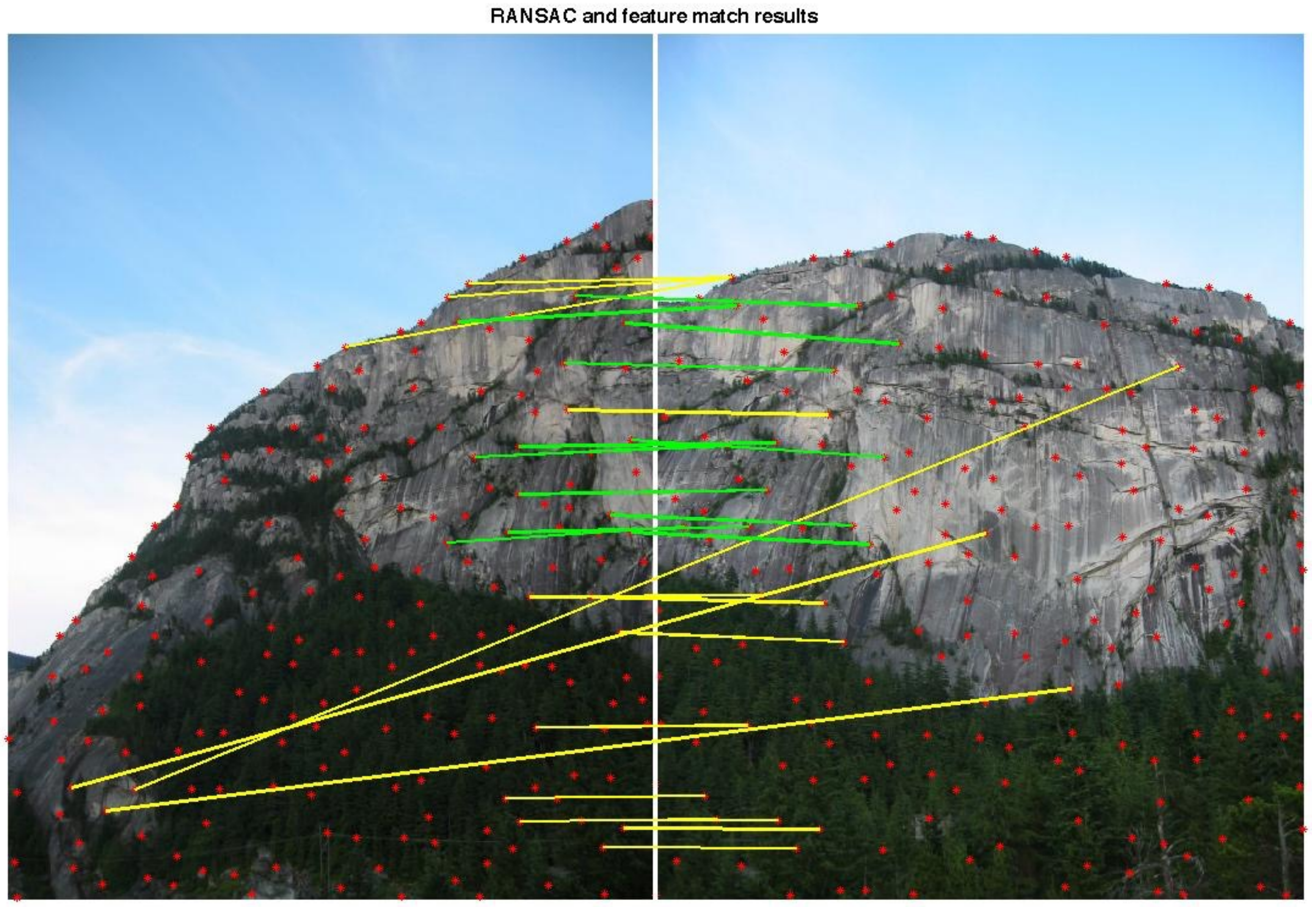


Image Stitching

I heavily based this part on Matlab's image panorama stitching tutorial, which allowed for a very robust stitching of multiple images. Assuming the pictures are in order and form a horizontal panorama, the homography for each picture is created in terms of the first one. Each subsequent homography between pairs is calculated and then multiplied by the previous homography in order to obtain a cell array of H matrices, all transforming their respective images to the first image. Then, the middle image is selected and its H matrix is inverted to be then multiplied by all the other H matrices. This switches the transformations from all the images to the middle image, which results in a much more visually appealing final result. This method made it very convenient to run the previous functions sequentially and then modify the H matrices to work with any number of images. However, because of the distortion that occurs at the edges of the panorama, I found that using more than 6 images did not produce excellent results. Matlab's alphaBlender was very convenient because it performed the masking for me. The imwarp function was also critical for this process. These are the results I obtained from a panorama looking out of my window. The respective RANSAC results are also shown.