Panorama Stitching

William Schulmeister
Sarah Bang
Brian Chin
Yashashri Pendse

General and Error Handling

The general scheme of our implementation was going, in order from the second image to last, calculating the geometric transformation necessary to match features from the current-1 image to the current one. The stitch at the end goes through all these transformations all at once.

We also established 20 as the lower limit for matched features to allow for matching. Any lower would be considered a “random” image that doesn’t fit into the panorama and an error would be thrown. This decision worked in the general case (like test_images - TestSet4),

but also threw an error in test_images - TestSet2, where the images were simply out of order (See Appendix for these outputs). This behavior is expected given our method, but a more user-friendly algorithm would retry out-of-order images later to try and attach in a different order.

For this document, we will show progressive output for train_images - Set1. Custom sets are displayed at the bottom.

AMNS

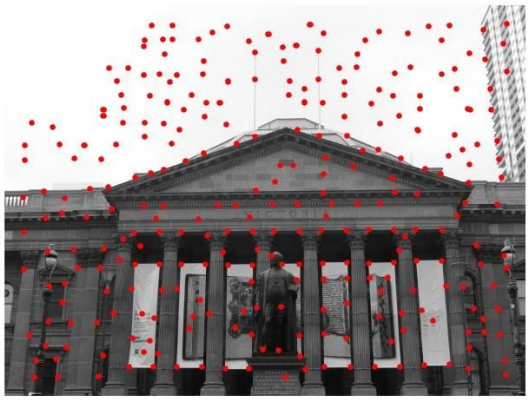
Description

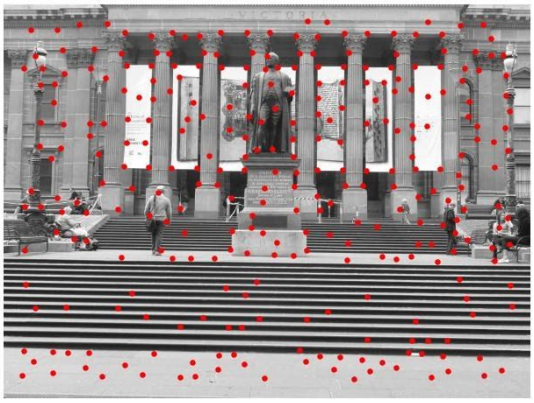
We then used the cornermetric function to get all the corners of each grayscale image. After we used imregionalmax to get just the strong corners in the image, we used the ANMS algorithm on all but the outermost 20 pixels of each image. We do this to account for the 40x40 patches that will need to be taken later. We selected N-Best to be 300 through trial and error.

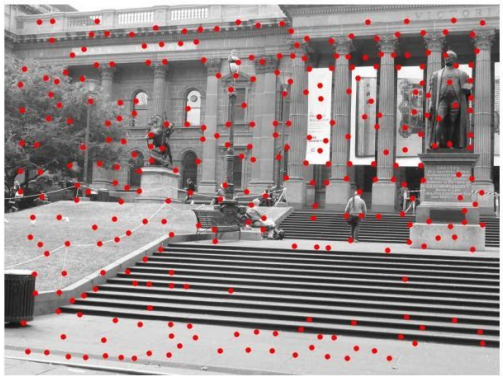
Shortcomings

Specifying N-Best manually includes a level of variability that is hard to account for in unknown sets. Although it performed well enough in our estimation, this rigid variable may not be as accurate elsewhere.

ANMS takes just the amount of points that are more equally distributed, however this may lead to removing some of the better points in a dense location which may contain better information than some of the points we are attempting to keep in less dense areas. This could lead to a decrease in accuracy in the matching and computation of the features later.







Feature Description

Description

We used the standard gaussian filter in fspecial for simplicity across all inputs and the fact that different parameters did not have a noticeable effect on the final product. For imfilter-ing the patches, we chose to use the 'replicate' option, which makes array values outside the border equal to the nearest border, since the borders were often basic colors or shapes.

At every N-Best point, getFeatureDescriptors() would take a patch around the point of the form: I(row-19:row+20, column-19:column+20), blur, reshape and standardize as described. The value is returned as a 64 x N-Best matrix.

Shortcomings

Although standardizing does help to an extent, blindly taking a 40x40 patch is not infallible. This technique is not resistant to scale or rotation, among other things, and only seems to be accurate when these facts are roughly negligible. It also could fall short on sets like TestSet1, where different patches may be almost identical because of patterns in the object.

Feature Matching

Description

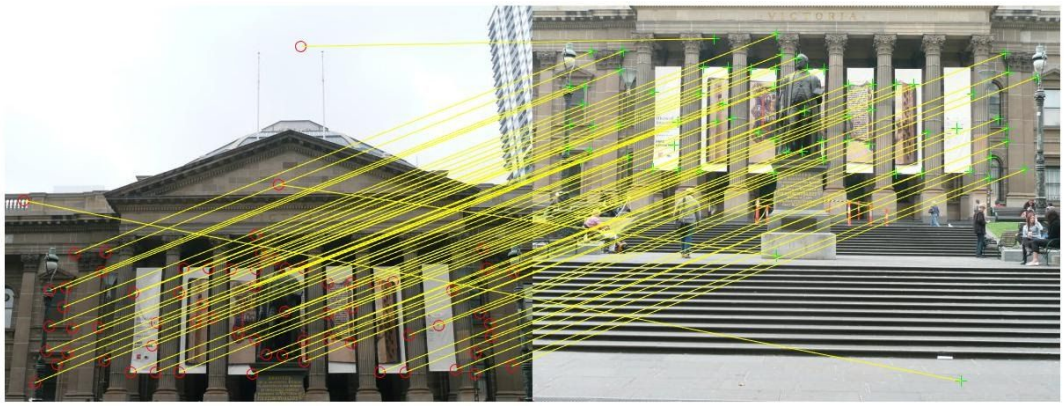
When implementing feature matching, we used Images 1 and 2 from train_images Set 1 to test and visualize the function. By comparing every feature descriptor from both images, we found the two lowest correspondences/vectors (computing SSD) and compared the ratio between these two best matches with our threshold (0.5). If ratio < threshold, then we kept the matched pair, ultimately resulting in the "confident" feature correspondences. We chose a threshold of 0.5 as a sweet spot where we let in enough matches that we didn't miss the good ones, but also kept out enough extraneous matches to not bog down RANSAC.

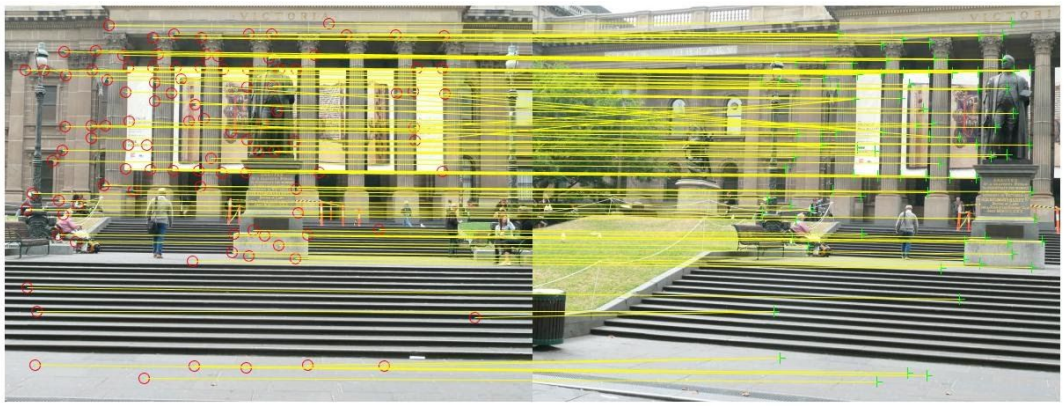
Shortcomings

A problem we ran into was the function being dependent on the correctness of earlier parts of the project, as the output gives no visual clues. Having seen that Feature Matching was not working properly the first couple tries, we found ourselves having to pinpoint exactly what was going wrong in our overall code in order to debug. Fortunately, we found the issue in

Feature Description and the rest of the functions flowed perfectly after our minor fix with the algorithm.

Additionally, in the case of test_images - TestSet1, the matching found enough pairs between the floors (or the ceiling in some cases of test_images - TestSet4) that the entire panorama was swayed. A more robust algorithm would potentially look for the subject first before matching.





RANSAC

Description

RANSAC takes 4 random matched features from the output of feature matching and calculates the homography for these features. The homography between the 4 pairs is then applied to every pair. If the pair's SSD is less than our threshold, we call this pair an inlier and

add it to a currInliers array. Once we have applied the homography to every pair we have a completed currInliers array. We only keep the largest set of inliers by having a variable bestInliers and resetting it to currInliers if currInliers is larger in size. currInliers is then reset. We keep repeating this process until our best inliers array consists of at least 90% of the matched features or we have done the process maxIters times.

We chose 1000 for maxIters as a reasonably large N to get the best result possible. As it turned out, the algorithm ran to 1000 most times, but accuracy did not suffer.

We chose 1 as our threshold for SSD through positive results in trial and error.

We chose 90% as the cutoff number to ensure this would only happen when the bestInliers array was sufficiently full.

Shortcomings

Our implementation was largely successful, but potential problems could arise, again, due to the arbitrary nature through which we defined constants. While we don't believe errors in our test output were due to RANSAC problems, there is no guarantee that in the future, our threshold or maxIters wouldn't be insufficient.

Stitching and Blending

Description

To stitch our panorama, we first estimated the transformation between I(n) and I(n-1) by using a built-in MATLAB function (fitgeotrans), where "I" represents an image in the set and "n" represents which image # we are on, and then computing T(n) * T(n-1) * … * T(1) where "T" represents the transformation matrix. This transformation is used to warp via imwarp(). We then computed the output limit and average X limits for each transform, which we then formed a center image.

To begin stitching, we first initialized our panorama by finding the minimum and maximum output limits, and using that to round an initial width and height (dimensions) of our panorama. We defined it further by using imref2d, which created a 2-D spatial reference object

defining the size of the actual panorama. For large inputs like train_images - Set3, we attempted to mitigate memory errors by scaling the panorama canvas by .1. While not a perfect solution (see shortcomings below), it was satisfactory for the scope of this project.

With all the warped images appropriately placed on the panorama canvas, we could do the final stitching and blending. For every point in the panorama, an average would be taken of all the layers. To preserve brightness, the negative spaces of each warp were not taken into account. With the blending and stitching process done, we use imshow to output our (mostly) lovely panorama.

Shortcomings

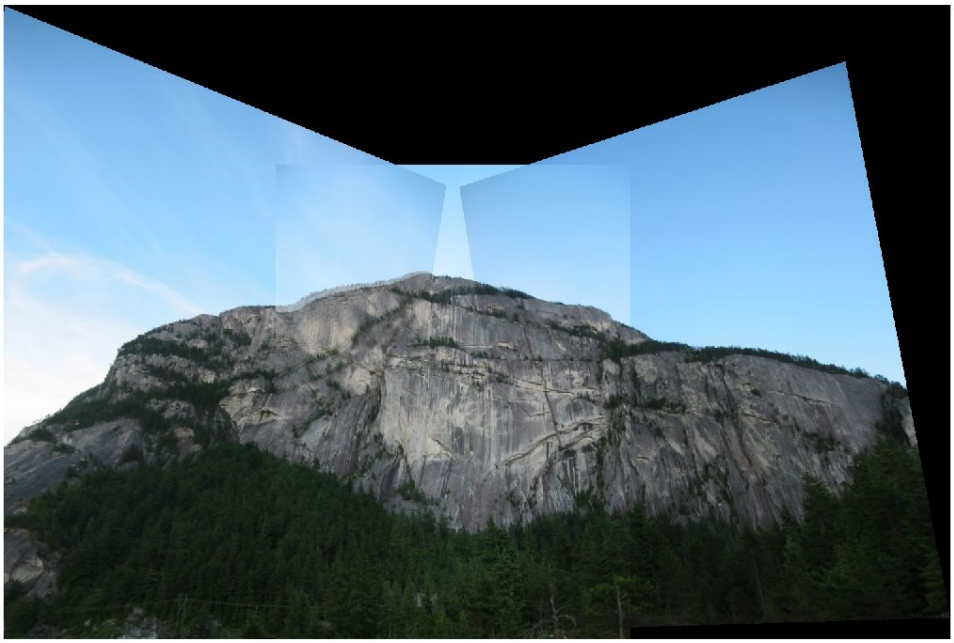
The biggest shortcoming here is the fact that we did not do Cylindrical Projection. Because of this, stitching was sometimes *slightly* off. In the case of train_images - Set3, it could also have required a huge amount of space because of the degree to which warping was necessary. Simply reducing the size by a factor of 10 to accommodate memory limitations ended up costing us significant accuracy and, even then, barely ran on our machines. It seems that for wide/tall panoramas, cylindrical projection is necessary.

Our method of blur was also a shortcoming. It was generally insufficient in making the images seem natural and was the worst of both worlds when it came to unique features in overlapping portions like people in the image below.

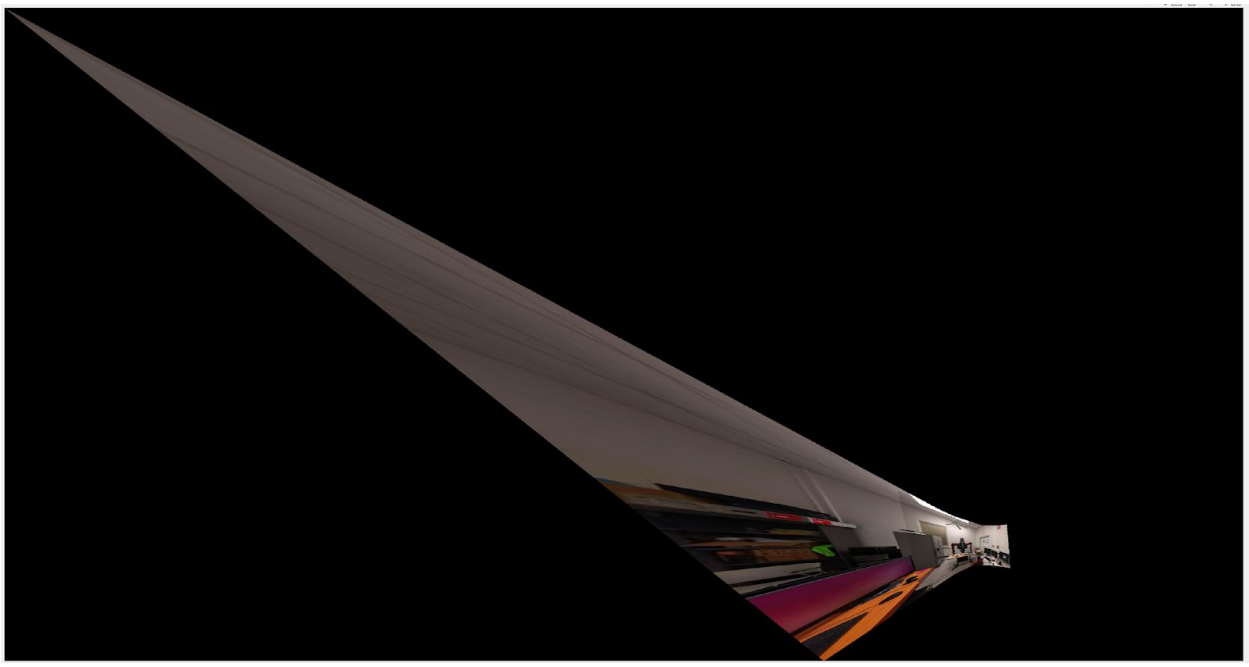
Panoramas

Train Set 1

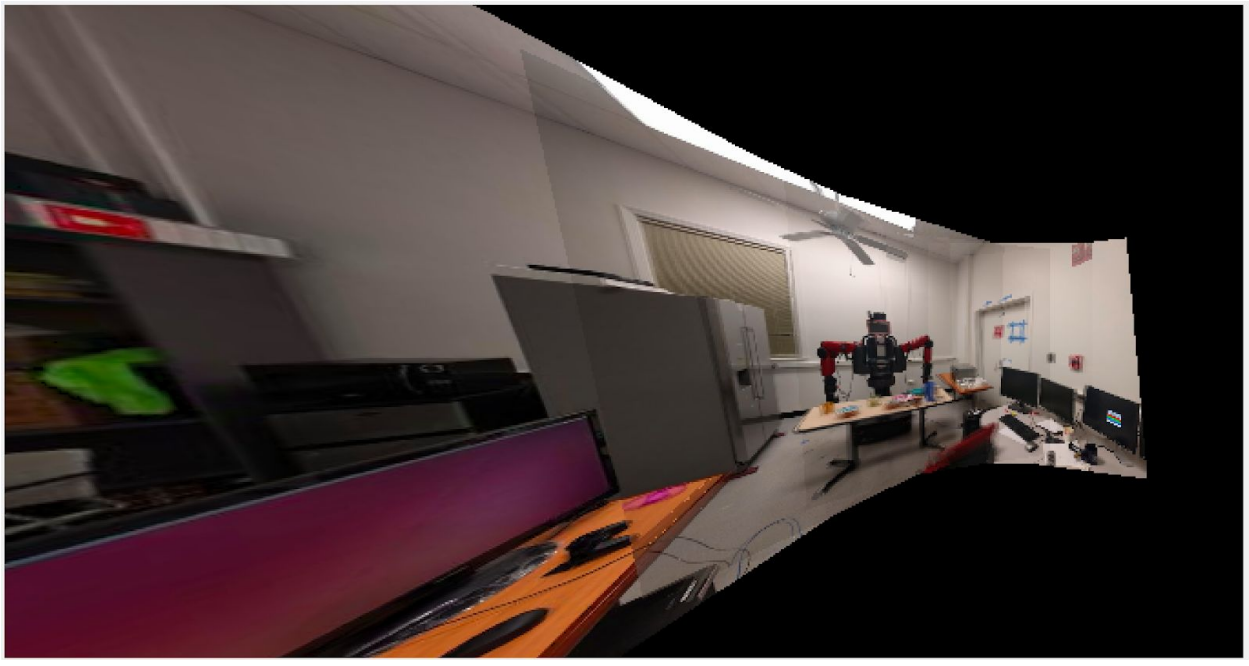
Train Set 2



Train Set 3



Zoomed in vvv

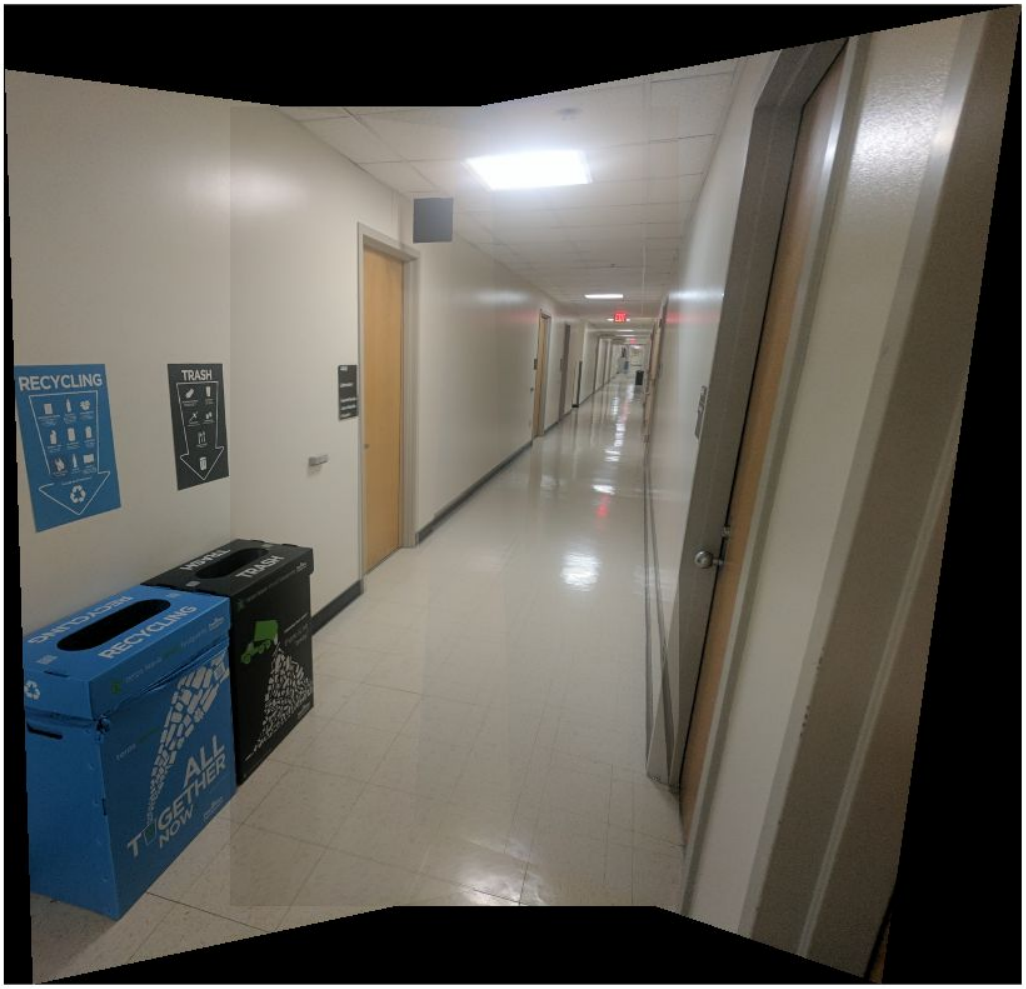


Test Set 1

Test Set 2

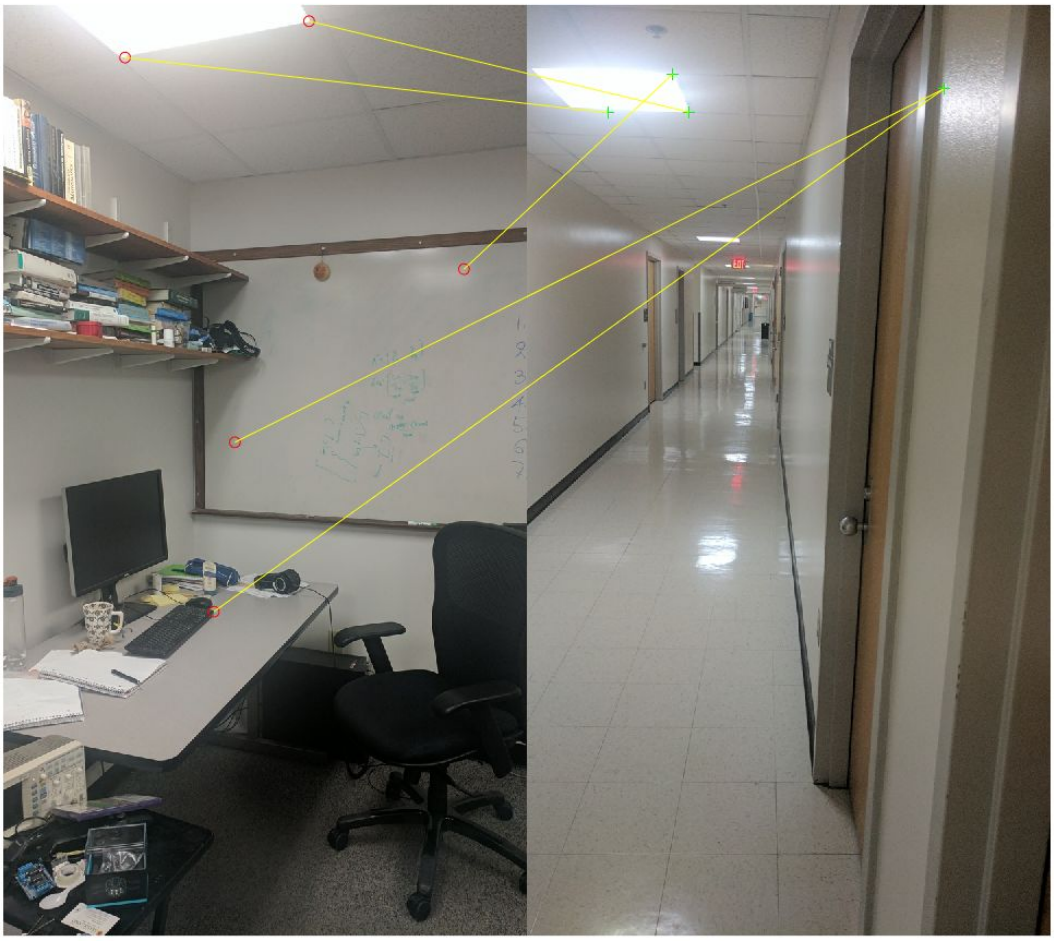
The program correctly throws an error on image 4 as it is not part of the canonical set.

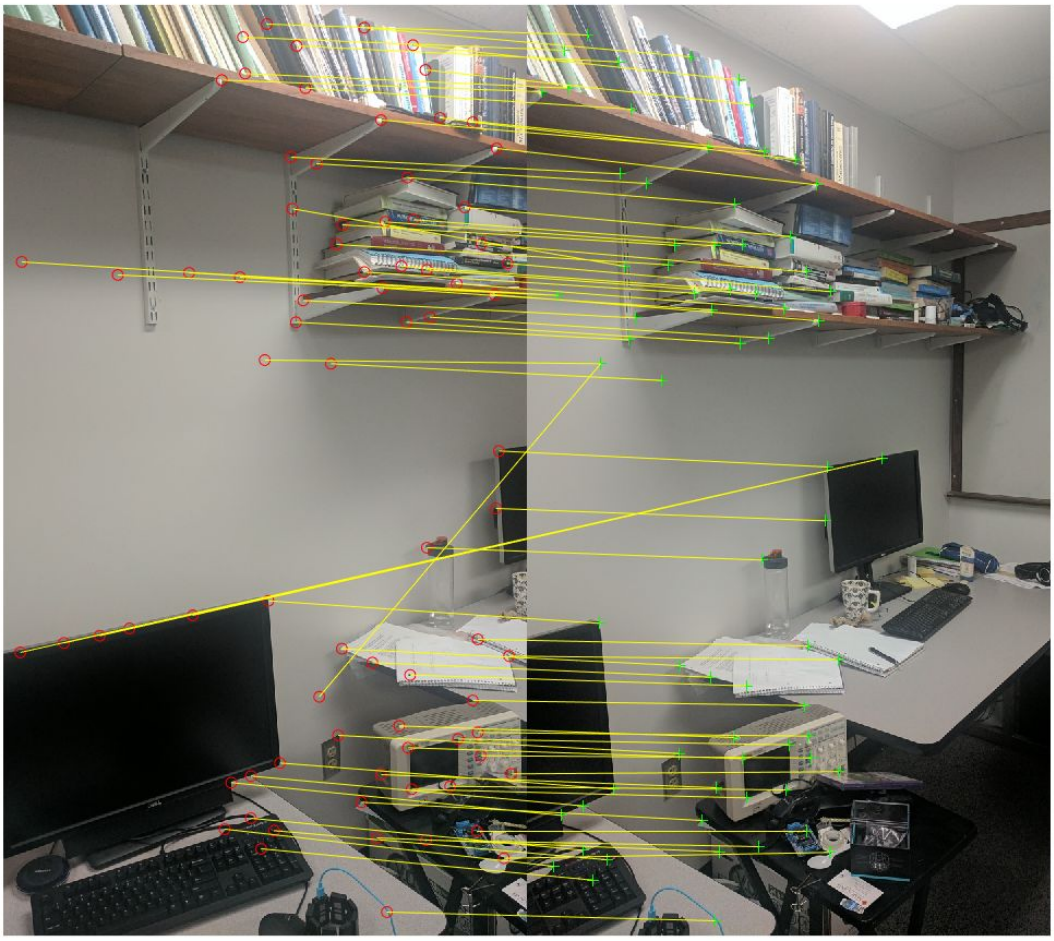
Test Set 3



Test Set 4

Test Set 4 returned an error when trying to create a panorama because the number of matched points (set to 20) was insufficient for Image 5. Displayed below are the RANSAC matches between the 4 Images (1-2, 2-3, 3-4, respectively).





Custom Image Set 1

Image 1

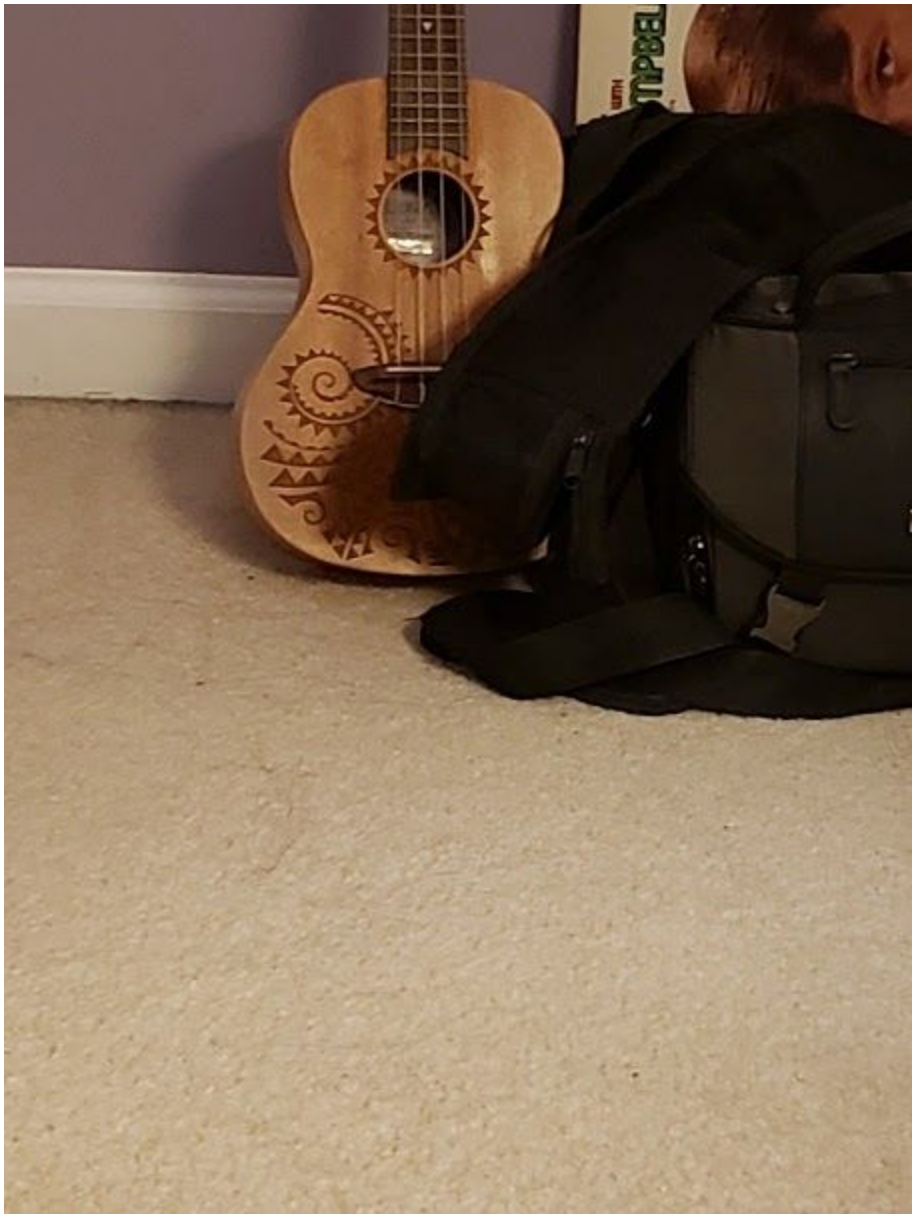
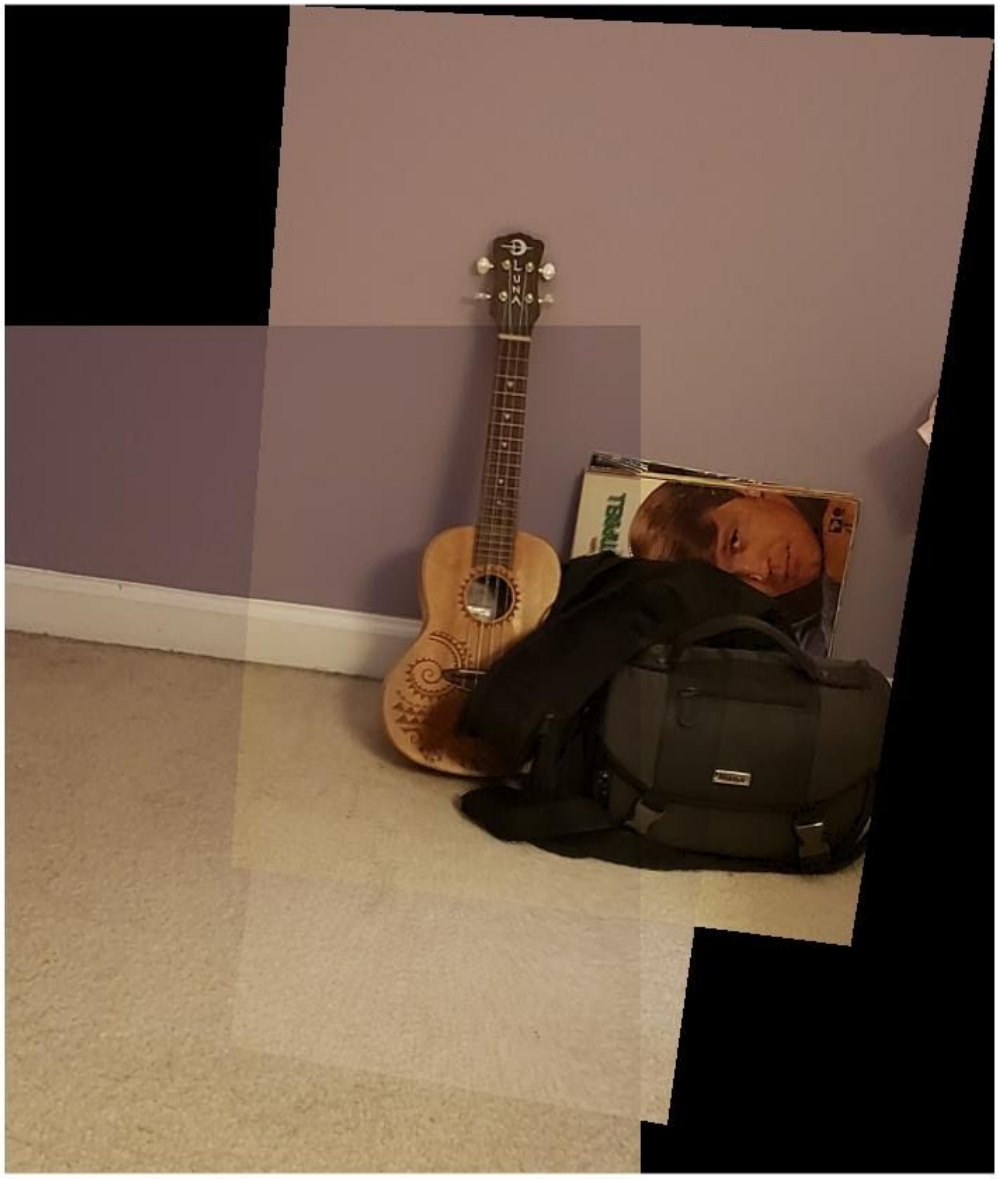


Image 2

Image 3

Panorama



Custom Image Set 2

Image 1

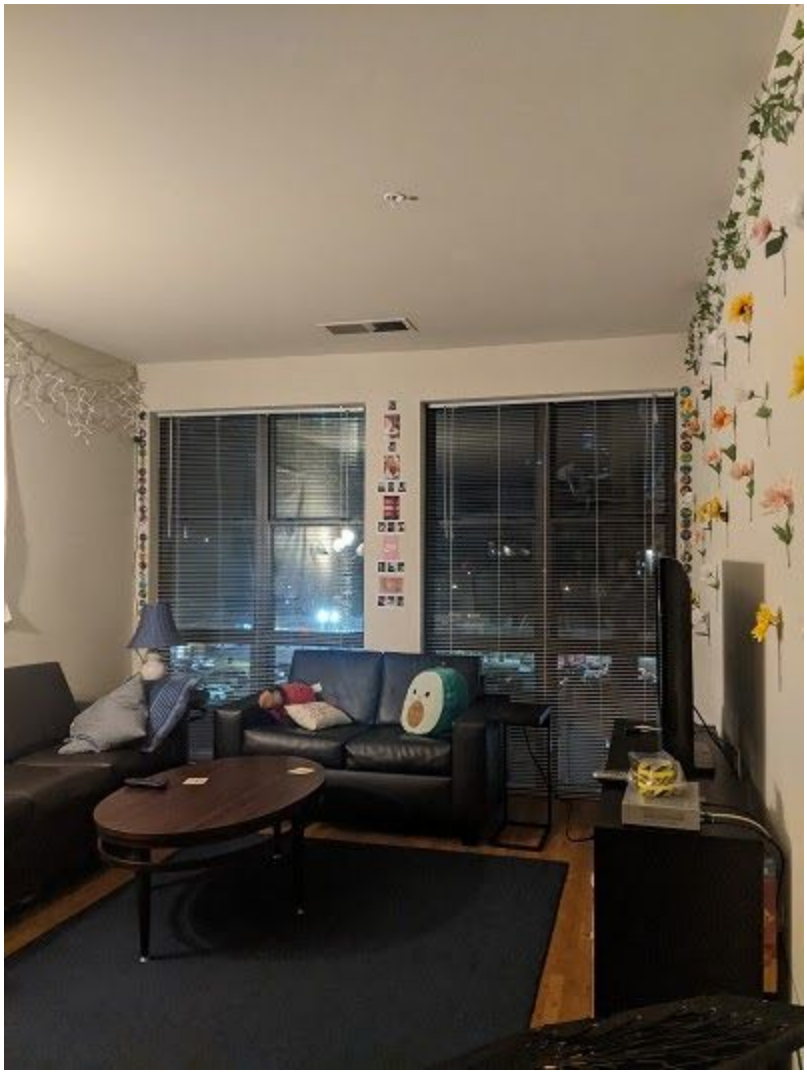


Image 2

Image 3

Panorama