CS 6640 Project 4

Travis Allen, u1056595

December 6, 2021

1 Preliminaries

I chose to do project 4a: Image Mosaicing. The code for this can be found in the `proj_4.py` and `functions.py` files. `proj_4.py` contains the actual algorithm developed to make a mosaic of greyscale images whose correspondences are documented in a `.json` file. `functions.py` contains some useful functions that would have otherwise crowded the `proj_4.py` file. My solution to this project relies heavily on`numba`, so you must have that installed to run my code. I use it because it *dramatically* speeds up the run time.

2 Experiments

2.1 Given Dataset

I took the given images and formed the following mosaic. I have matched the contrast of the images with histogram matching (see section **??** for more detail).

Figure 1: 4 perspectively equivalent images made into a mosaic.

2.2 Panoramic Images

I used the following panoramic images to see how the algorithm performs on them. This is using 10 correspondence points.

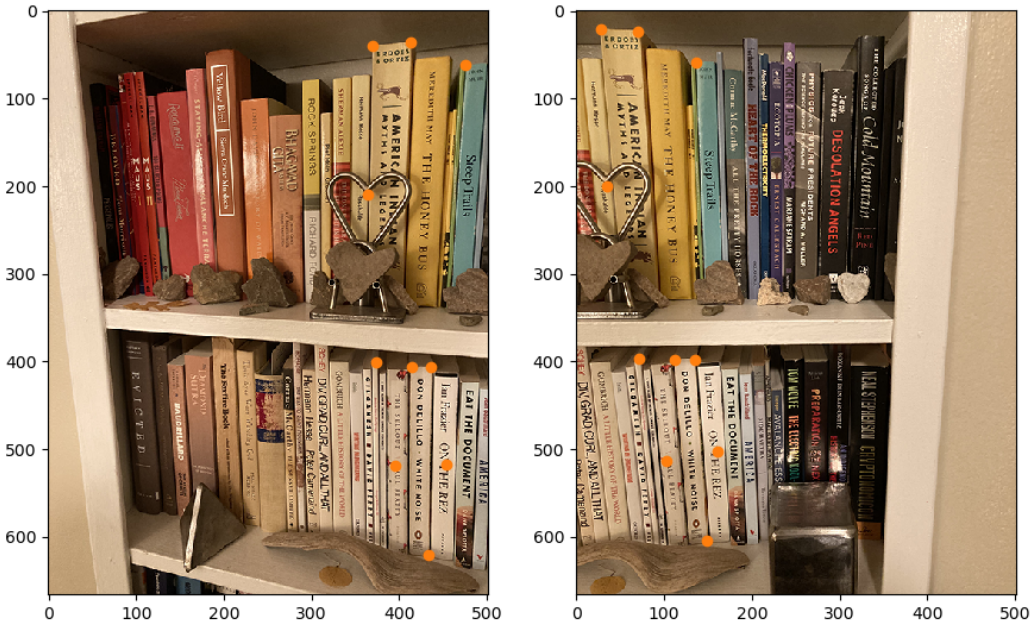

Figure 2: Bookshelf with 10 correspondences. I picked this image because it contains many corners, so accurately picking correspondences was straightforward.

Figure 3: Mosaiced bookshelf with 10 correspondences.

2.3 Planar Images

I used the following planar images to see how the algorithm performs on them. To construct them, I simply took a single image and cropped it twice to keep different, but overlapping parts of the image. Here I used 12 correspondence points, shown below.

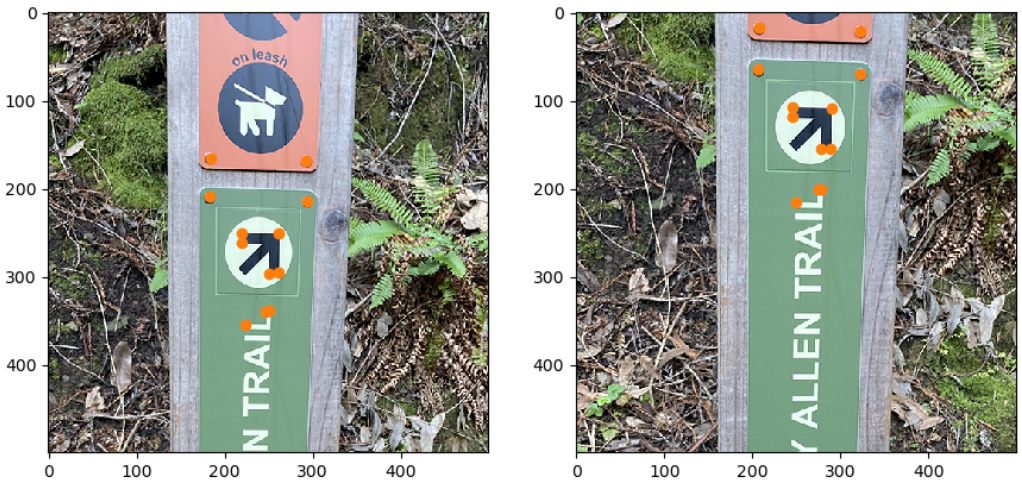

Figure 4: Trail sign with 12 correspondences. I picked this image because it contains many corners, so accurately picking correspondences was straightforward.

Figure 5: Mosaiced trail sign with 12 correspondences.

2.4 Number of Correspondences

I experimented with the number of correspondence points on both the planar and panoramic images.

2.4.1 Planar Images

I tried to use 4 correspondence points on the planar image shown above. I chose the number 4 because it is the theoretical minimum number of correspondence points needed to solve for the transformation. This worked with varying degrees of success. Some combinations of 4 of my total 12 correspondence points produced a mostly valid transformation, shown below. However some combinations of correspondence points did not work as well and would have produced a very large and ugly image. I attribute this to the correspondence points being chosen by hand, so they were not extremely accurate.

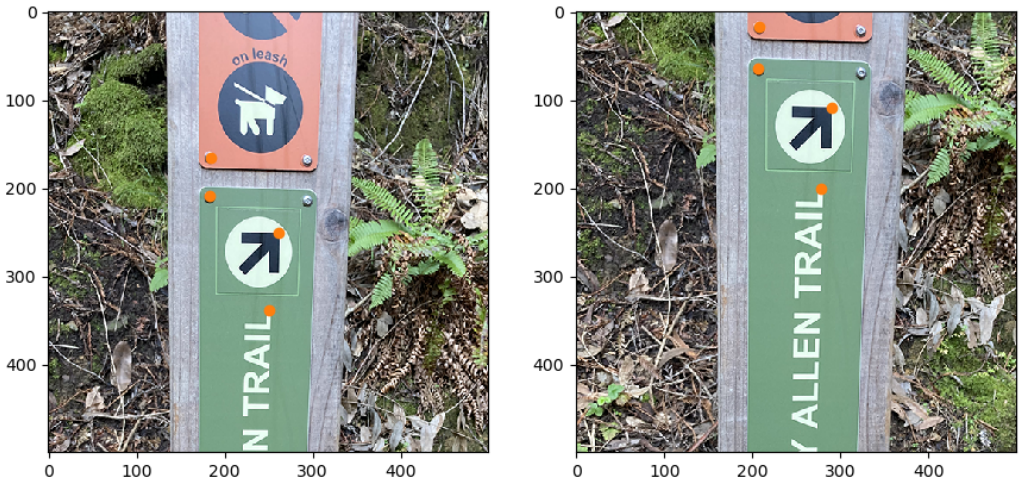

Figure 6: Trail sign with 4 correspondences.

Figure 7: Mosaiced trail sign with 4 correspondences. Things don't line up very well with so few correspondences.

Shown below again for clarity's sake is the same pair of images but constructed with a transformation informed by 12 correspondence points. This seemed to act as a filter on my imprecise correspondence point picking and generally make the image look better.

Figure 8: Mosaiced trail sign with 12 correspondences. This lines up much better.

2.4.2 Panoramic Images

I attempted to use 4 correspondence points on the panoramic images shown above, but I was unsuccessful. I encountered the same issue I described in the section on planar images where the transformation that was produced made too large of an image (orders of magnitude larger than the original images), so I added one correspondence point and successfully made a mosaic with 5 points. This is shown below.

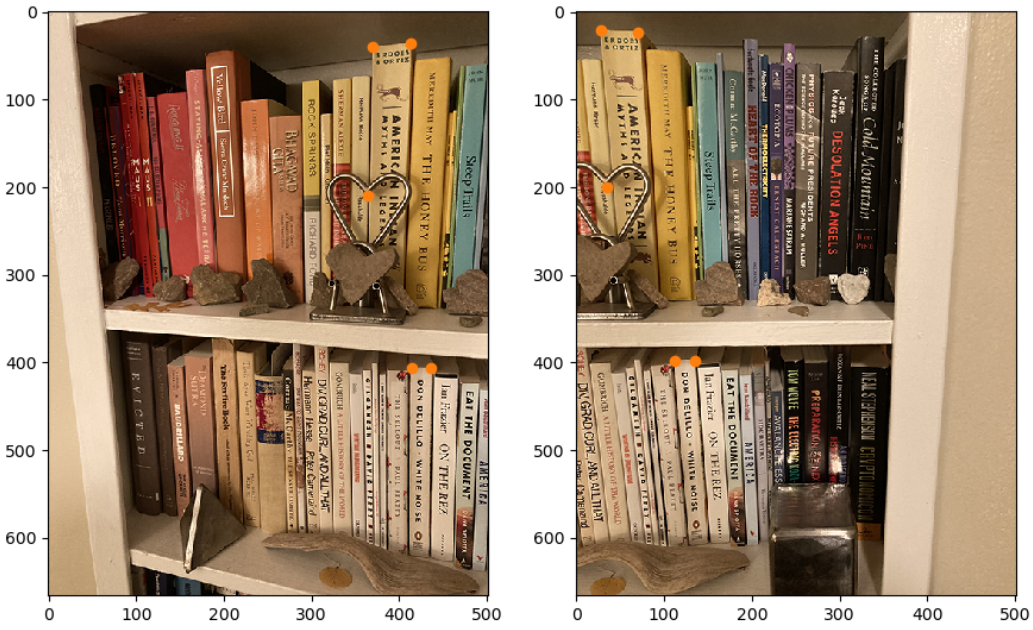

Figure 9: Bookshelf with 5 correspondences. I picked this image because it contains many corners, so accurately picking correspondences was straightforward.

Figure 10: Mosaiced bookshelf with 5 correspondences.

Shown again for clarity are the results from above with 10 correspondence points. In general, the image lines up slightly better, especially near the bottom.

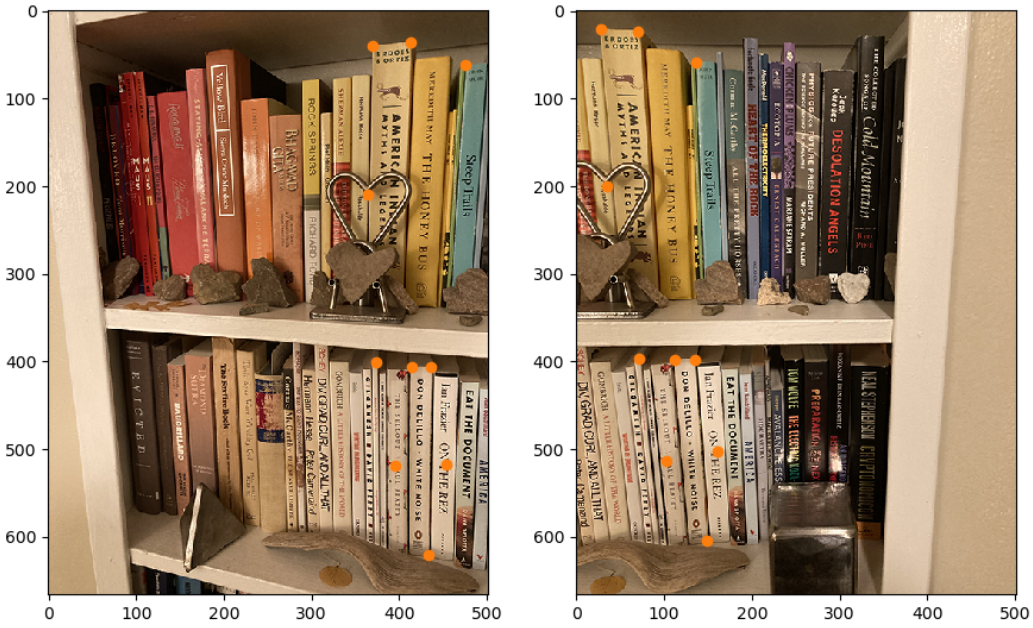

Figure 11: Bookshelf with 10 correspondences.

Figure 12: Mosaiced bookshelf with 10 correspondences.

3 Questions

3.1 How many control points does it take to get a 'good' transformation between images?

My results in the preceding sections indicate that having more correspondence points results in a better image mosaic. Solving a system with SVD provides the least-squares solution, so this comports with our intuition regarding least-squares solutions and the central limit theorem, wherein the "quality" or "truth" of a prediction improves with the number of data points that are used to make it. From my own initial experimentation it seems that 8 or greater correspondence points capture enough of the truth of the perspective equivalence to construct a convincing image.

3.2 How does the algorithm behave at the theoretical minimum of the number of control points?

The theoretical minimum number of correspondence points is 4 because that is the minimum possible number of points that allows the matrix below to have full rank. If this matrix has at least rank 8 and the number of rows is equal to the rank, then the system has an analytical solution. If the system has at least rank 8 and the number

of rows is greater than the rank, then the system has a least-squares solution.

$$\begin{pmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1x_1' & y_1x_1' \\ -x_2 & -y_2 & -1 & 0 & 0 & 0 & x_2x_2' & y_2x_2' \\ -x_3 & -y_3 & -1 & 0 & 0 & 0 & x_3x_3' & y_3x_3' \\ -x_4 & -y_4 & -1 & 0 & 0 & 0 & x_4x_4' & y_4x_4' \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & x_1y_1' & y_1y_1' \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & x_2y_2' & y_2y_2' \\ 0 & 0 & 0 & -x_3 & -y_3 & -1 & x_3y_3' & y_3y_3' \\ 0 & 0 & 0 & -x_4 & -y_4 & -1 & x_4y_4' & y_4y_4' \end{pmatrix} \begin{pmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{21} \\ p_{23} \\ p_{23} \\ p_{31} \\ p_{32} \end{pmatrix} = \begin{pmatrix} -x_1' \\ -x_2' \\ -x_3' \\ -x_4' \\ -y_1' \\ -y_2' \\ -y_3' \\ -y_4' \end{pmatrix}$$

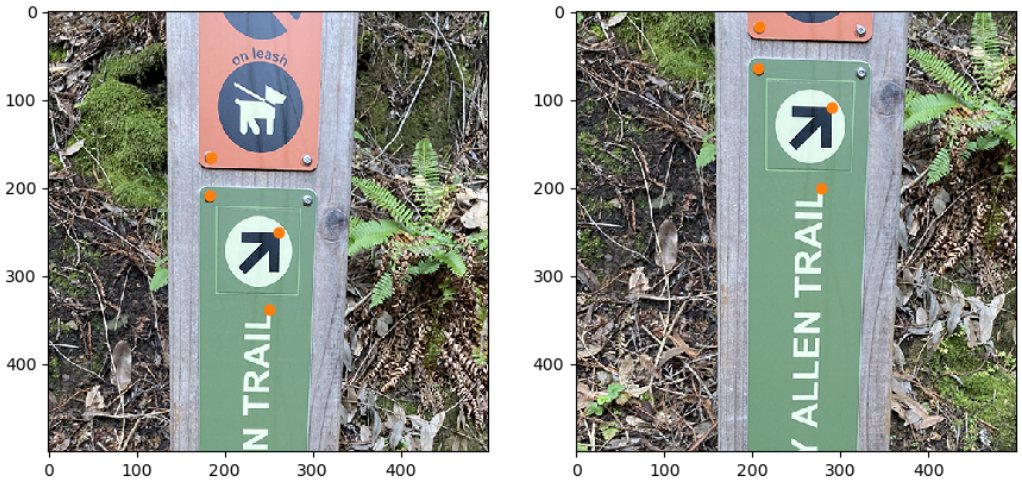

Figure 13: Trail sign with 4 correspondences.

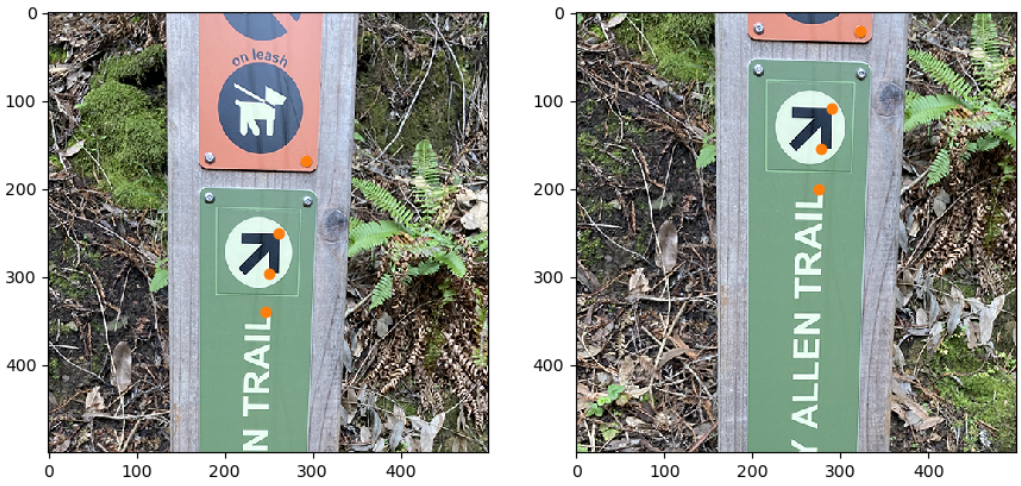

Figure 14: Trail sign with 4 different correspondences.

3.3 From your experiments, how does the accuracy of the control points affect the results?

To test this, I swapped two control points in one of the two images being mosaiced. I did this for 4 control points and then for 10 and 12.

4 Details

4.1 Contrast

?? A good algorithm should automatically adjust for major intensity differences.

To handle this I used the `skimage.exposure.match_histograms` function to match all of the histograms to the first image that I read in to the program. Below are some results with and without histogram matching

Figure 15: Mosaic with no histogram matching. Differences in intensity are very obvious and distracting.

Figure 16: Mosaic with histogram matching. Differences in intensity are much less severe and are in general less distracting.

4.2 Feathering

4.3 Image Size

Initially, for n number of images, I make a canvas that is $n + 1$ times the size of the largest image so that I have enough room to work with when placing images in the mosaic. However, this often results in a canvas which is much larger than it needs to be. To return the canvas to a more reasonable size for viewing once the mosaic is complete, I execute the following procedure. I search through the large canvas to find the first and last rows and columns which contain only zero elements. I do this by using the `numpy.sum()` function on each row and column and checkign to see if it is equal to 0. The canvas consists of all zeros before I place images on it, so this method works by assuming that a row of all zeros contains no image information. I perform it this way because I figure that `numpy`'s vectorization is faster than my own implementation of computing the sum or individually inspecting every element in the image.

- Place all of the images in a folder with a known path to the directory that contains `problem_4.py`
- Place all of the names of all of the images in a `.txt` file in the folder, with each name separated by a new line
- Write the names of the folder and the file in lines `27` and `28` of `problem_4.py`
- Write the maximum size of the images in lines `21` and `22` of `problem_4.py`