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

2.2 Panoramic Images

I used the following panoramic images to see how the algorithm performs on them. I compared the results with 4 and 10 correspondence points.

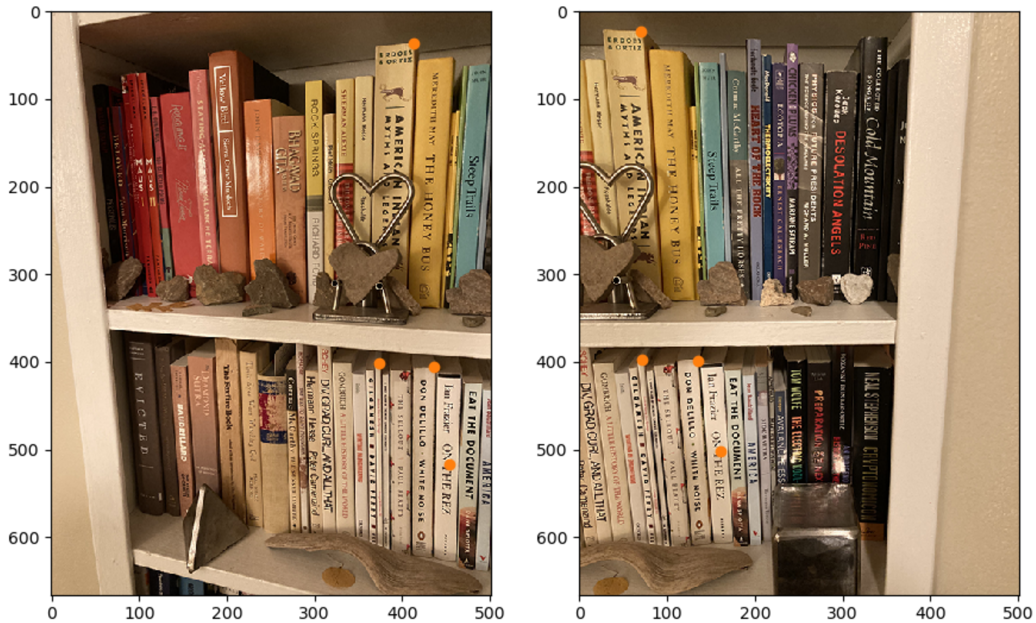

Figure 1: Bookshelf with 4 correspondences. I picked this image because it contains many corners, so accurately picking correspondences was straightforward.

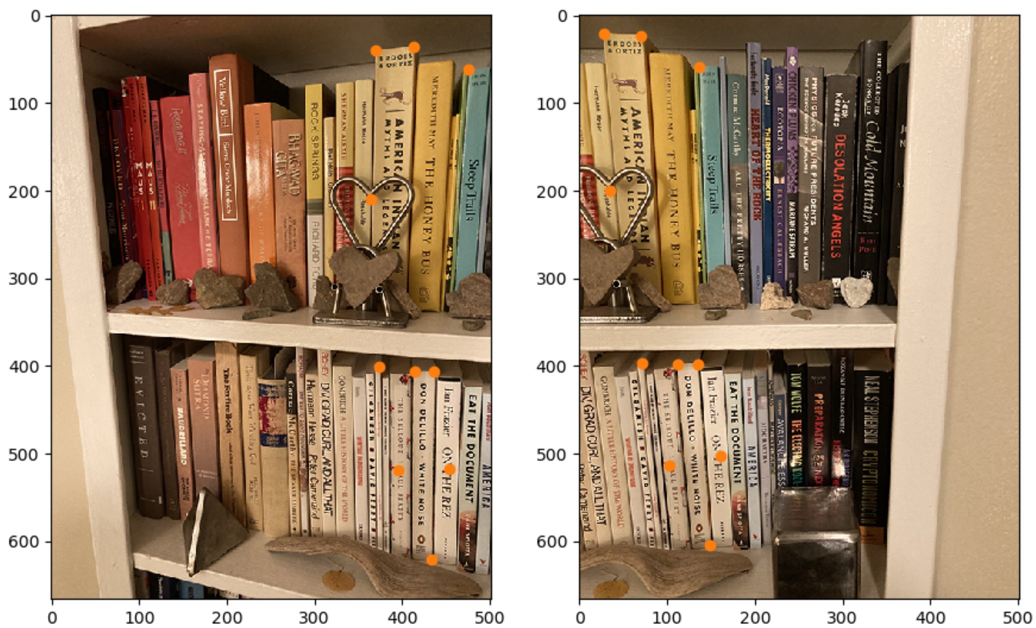

Figure 2: Bookshelf with 10 correspondences. I picked this image because it contains many corners, so accurately picking correspondences was straightforward.

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 4 and 12 correspondence points, shown below.

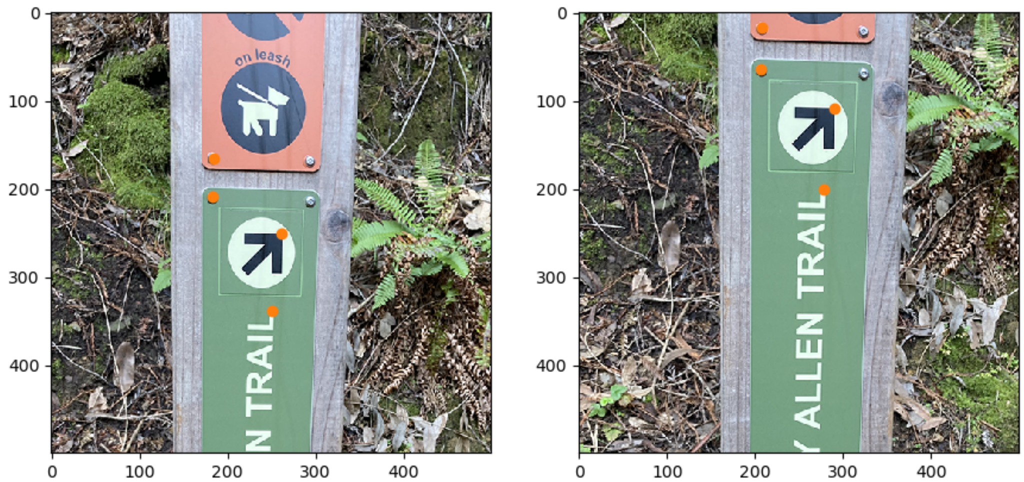

Figure 3: Trail sign with 4 correspondences. I picked this image because it contains many corners, so accurately picking correspondences was straightforward.

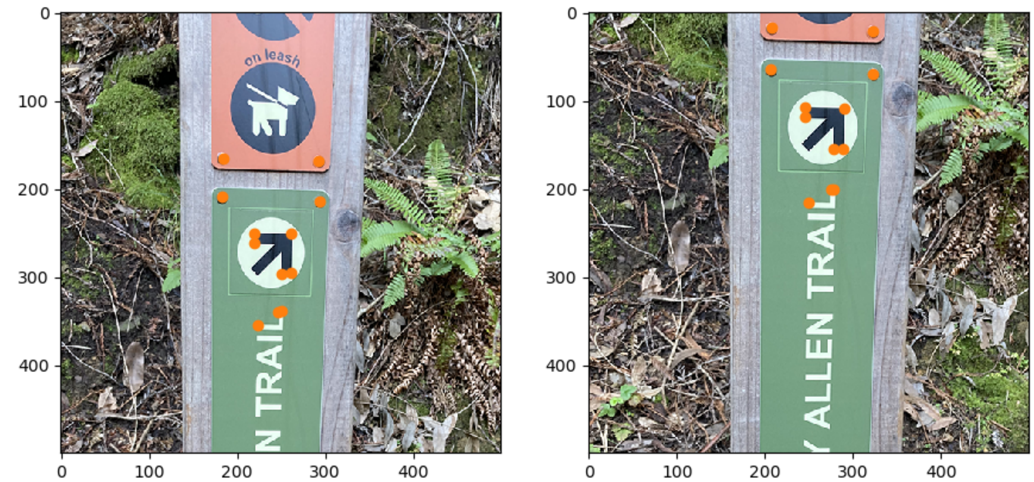

Figure 4: Trail sign with 12 correspondences. I picked this image because it contains many corners, so accurately picking correspondences was straightforward.

2.4 Number of Correspondences

3 Questions

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

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.

$$\begin{pmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1x'_1 & y_1x'_1 \\ -x_2 & -y_2 & -1 & 0 & 0 & 0 & x_2x'_2 & y_2x'_2 \\ -x_3 & -y_3 & -1 & 0 & 0 & 0 & x_3x'_3 & y_3x'_3 \\ -x_4 & -y_4 & -1 & 0 & 0 & 0 & x_4x'_4 & y_4x'_4 \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & x_1y'_1 & y_1y'_1 \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & x_2y'_2 & y_2y'_2 \\ 0 & 0 & 0 & -x_3 & -y_3 & -1 & x_3y'_3 & y_3y'_3 \\ 0 & 0 & 0 & -x_4 & -y_4 & -1 & x_4y'_4 & y_4y'_4 \end{pmatrix} \begin{pmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{21} \\ p_{23} \\ p_{23} \\ p_{31} \\ p_{32} \end{pmatrix} = \begin{pmatrix} -x'_1 \\ -x'_2 \\ -x'_3 \\ -x'_4 \\ -y'_1 \\ -y'_2 \\ -y'_3 \\ -y'_4 \end{pmatrix}$$

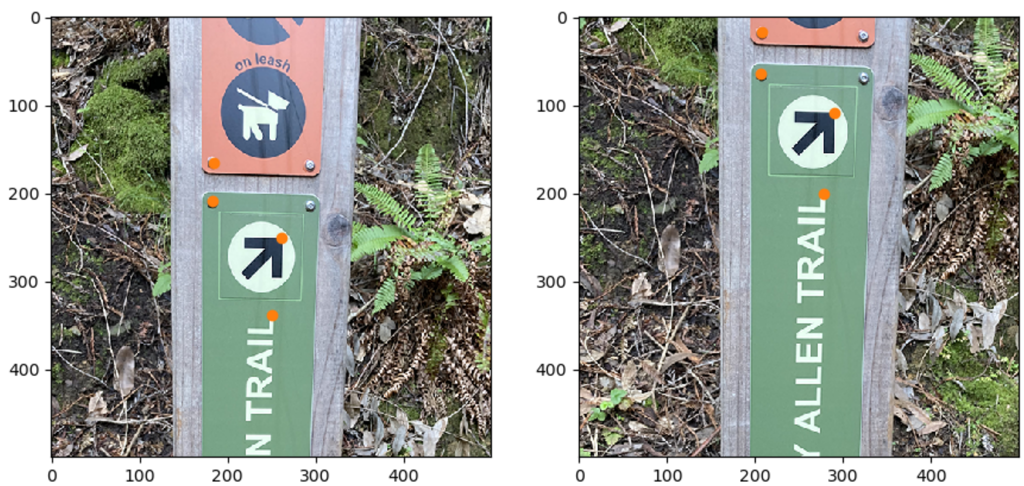

Figure 5: Trail sign with 4 correspondences.

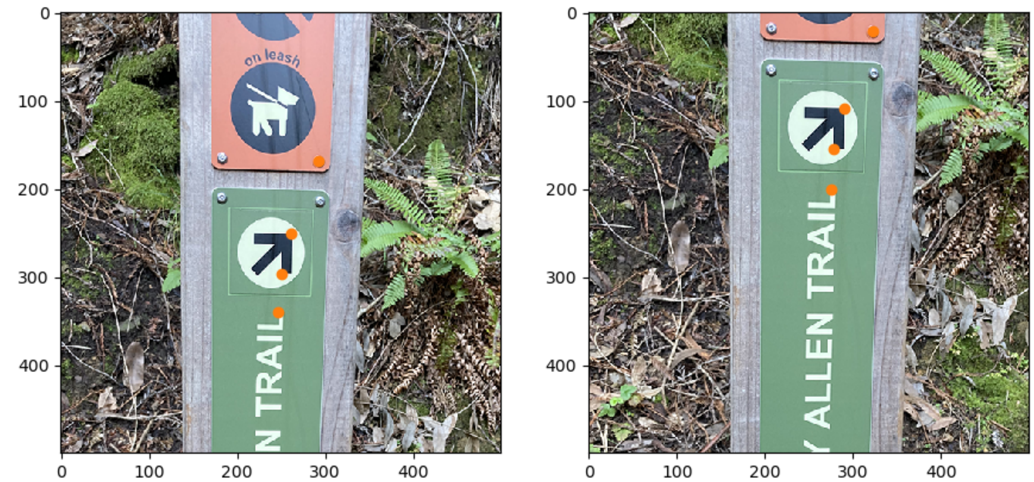

Figure 6: Trail sign with 4 different correspondences.

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

4 Details

4.1 Contrast

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`