CAP6419: 3D Computer Vision

Assignment 3: Image Mosaic Generation Report

By: Shane Davis

11/5/2022

Abstract

An image mosaic is an artificial image generated from a sequence of images that is intended to look like a wide image of a scene. This paper goes over the process I took to generate the mosaic. Afterwards, resulting mosaics are displayed along with the analysis on those results to determine issues within the mosaics. Finally, I conclude with the final analysis and future improvements on this process.

Approach

My approach to generating the mosaic was mainly based on Algorithm 4.6, Automatic estimation of a homography function between two images using RANSAC, from the textbook (Page 123) [1].

Setup Steps

To give a basic outline, the program first loads in all the images from a folder, with all the images following the format of {prefix}_{n}.jpg, with n being positive integers starting at 1 and incrementing throughout the sequence. It then calculates a sufficiently large canvas, which I determined that (4 * width, 2 * height) was a sufficient size for, and places the middle image onto the middle of the canvas. Afterwards, it precomputes the SIFT feature points for the middle image because it will never change throughout the main running loop of the algorithm.

Main Algorithm

The main algorithm loop runs through all the images in a specific order, with the index first starting just below the middle index and going to 1, then restarting back just above the middle index and finishing the sequence. I chose this ordering because I wanted to prioritize images that were closer to the middle image first, which gave me better results than if I were to run a basic loop starting from 1 and ending at the end of the sequence.

For each image at the chosen index, the following steps are ran:

1. Detect and extract the SIFT features from the chosen image.
2. Find the matching features from the chosen image and the middle image.
3. Extract the points from the matched features from both images and correct the middle points to be starting at the center of the canvas.
4. Run Peter Kovesi's RANSAC algorithm on the points to determine the inliers, using a threshold of 0.001 [2].
    a. I determined the distance threshold of 0.001 from just adjusting the value until I got good enough resulting images. This distance threshold represents the normalized distance of a data point from the generated point from the calculated homography, which controls if the data point is an inlier.
5. If the number of inliers is above 10, calculate the homography using the normalized DLT algorithm (Algorithm 4.2, The normalized DLT for 2D homographies, page 109 [1]) with the inlier points.

    a. I chose to add this additional threshold of greater than 10 inliers because I was finding that even with the sufficient amount of 4 or more inliers, the applied homographies gave poor results on some of the images.
6. Place the chosen image onto a temporary canvas, apply the calculated homography to the image, and stitch the new temporary canvas back onto the working canvas, replacing only blank pixels on the working canvas.
    a. Replacing only blank pixels on the working canvas is important since we do not want to replace pixels that have already been calculated, since the ordering of the indexes will prefer closer images to the middle image first, meaning that pixels that are already on the canvas are most likely the best that it is going to generate.

Final Step and Limitations

The final step is to crop the finished canvas to the final image, since the canvas is most likely going to be larger than the actual image. It then finally saves the result as {prefix}_mosaic.jpg in the current folder.

One clear limitation with this approach is that all images must contain a significant portion of the middle image. This is because every time the homography is calculated, it uses the middle image along with a chosen image. Therefore, any images that do not contain a significant portion of the middle image (significant as in the SIFT feature extractor will be able to identify features from the middle image in the chosen image), will give poor results with low inliers (which are filtered).

Results

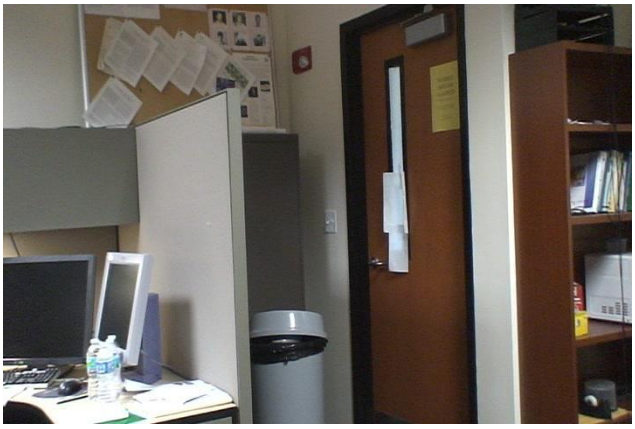
Provided Images

Office

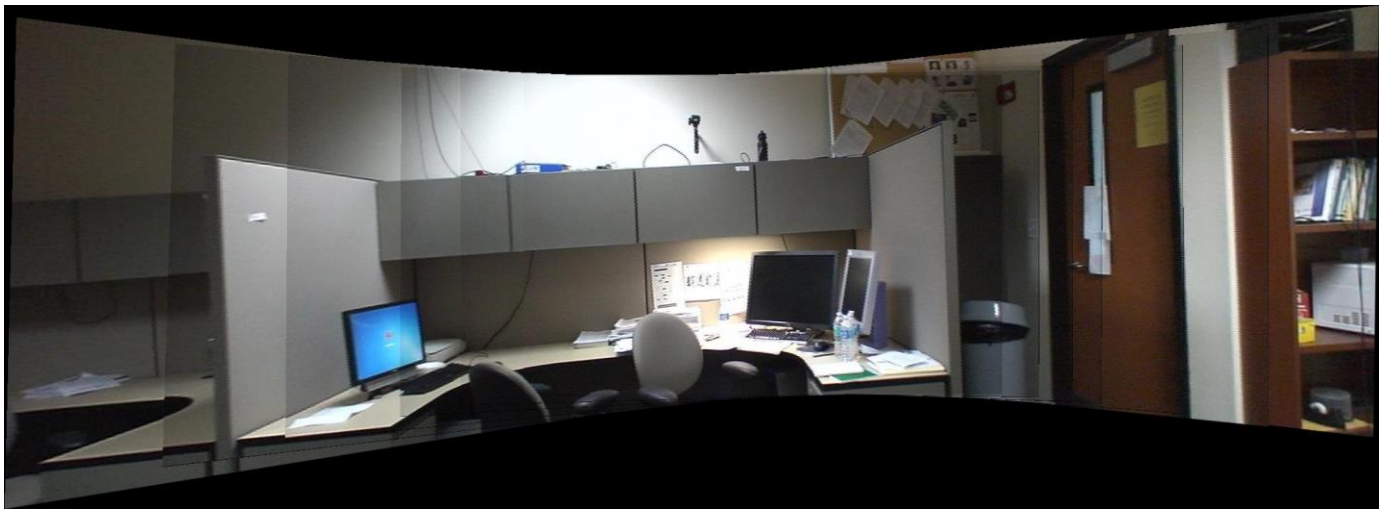
Middle Source Image

Left-most Source Image

Right-most Source Image



Resulting Mosaic



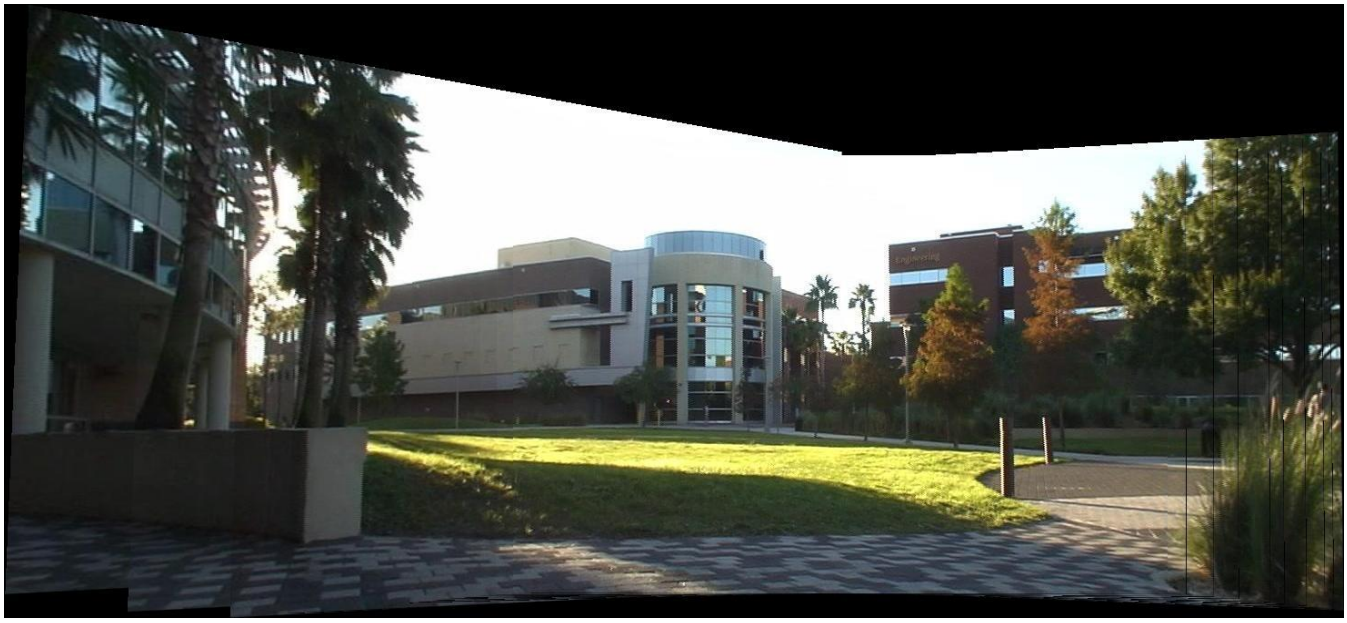
HEC

Middle Source Image

Left-most Source Image

Right-most Source Image

Resulting Mosaic

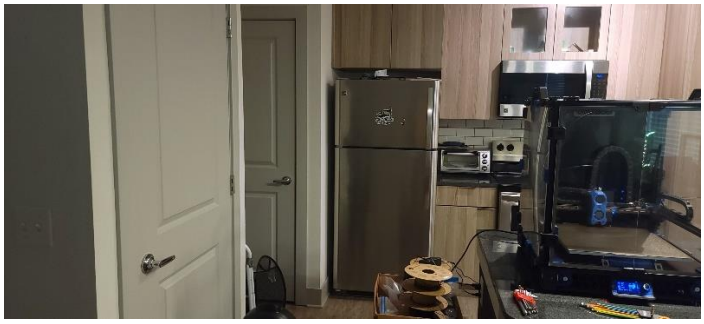


Self-Sourced Images

Kitchen

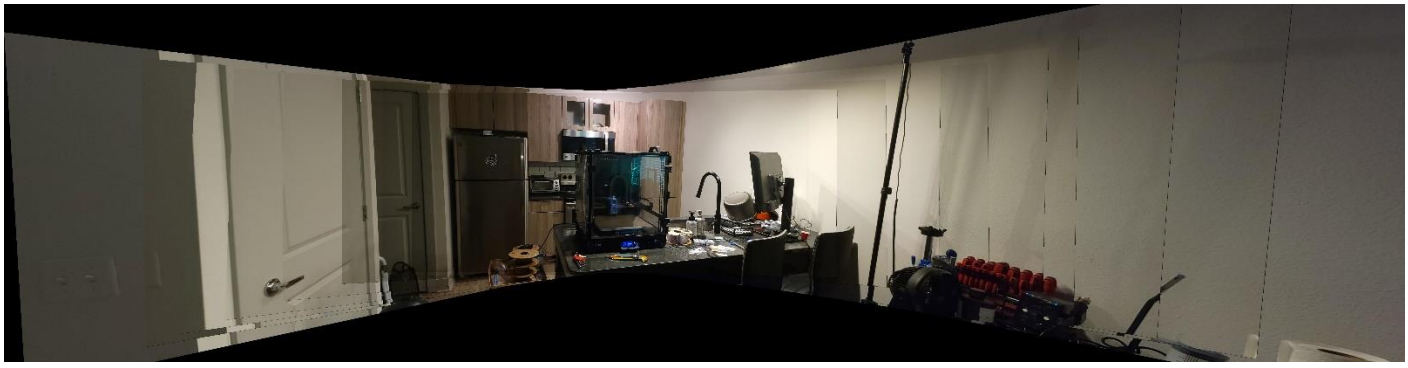
Middle Source Image

Left-most Source Image



Right-most Source Image

Resulting Mosaic



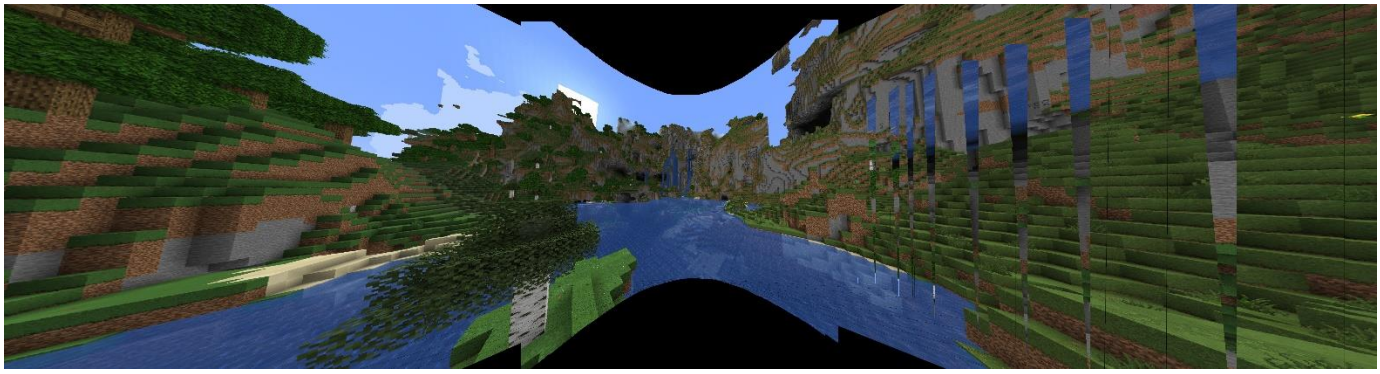
Minecraft

Middle Source Image

Left-most Source Image

Right-most Source Image

Resulting Mosaic



Concluding Analysis on Results

Provided Images

For the provided images, the mosaics turned out well with minimal noticeable noise in the mosaics.

For the Office image, some of the left side images were cut out from the resulting mosaic due to the inlier threshold I set in my code. This is because the left-most image contained no portion of the middle image, therefore any matching SIFT feature points was due to noise, and a valid homography was unable to be generated (without the threshold, the images would be stretched and translated an unreasonable amount, making a messy resulting mosaic). Unfortunately, this is an unfixable flaw with my approach to the mosaics: all the images in the sequence must contain a noticeable portion of the middle image (noticeable as in SIFT would be able to pick feature points that are from the middle image). However, this approach does have the benefit of strong results from images that do follow that criteria since there is no additional noise being applied to the image (versus an approach such as using the left-most image and building homographies continuously from there).

For the HEC image, the resulting mosaic turned out great, the only noticeable flaw that I found is the edges on the right are very noticeable. These great results came from the fact that the sequence of images followed that criteria, both "edges" of the sequence contained a significant portion of the middle image (the center building), and SIFT was able to find useful feature points.

Self-Sourced Images

For my self-sourced images, I wanted to test how well my program could perform on introducing a slight amount of translation as I capture the Kitchen images, as the whole idea of the program is based around the fact that the camera has no translation, and I wanted to test how well a sequence of in-game screenshots would perform with the Minecraft images.

For the Kitchen scene, the resulting mosaic turned out decent. Towards the edges of the mosaic, there is clear alignment issues in the edges of the door on the left and the lamp on the right. Once again, the seams from the individual images are noticeable with the black lines. However, the main portion of the scene turned out better than I expected, and I am satisfied with the results.

The Minecraft scene is much more interesting, as the "curve" that all these mosaics have is narrower compared to the rest of the mosaics. With this scene, I intentionally took much more screenshots with less rotation per screenshot compared to the rest of the scenes, as I believed that less rotation per image would result in a better mosaic. However, after analyzing all the resulting mosaics, I don't believe that helped much and it instead made the program take much longer to compute the resulting mosaic since it had almost 50 source images to process. I chose to use a video game since I can easily prevent any camera translation from occurring by just not moving, which isn't as easy in real life by holding the camera manually. The main and left-side portion of the resulting mosaic turned out well, no noticeable noise and it looks accurate. However, the right-side portion

contained much more noise than I expected, and even left out the trees on the right. I believe this may come from the fact that the scene does not have many unique details since all the blocks use the same texture and there isn't a large number of different blocks, which resulted in very noisy SIFT features that were extracted. If I were to use a scene with more unique details throughout it and less repeating patterns, I believe the program would perform better. Excluding the noise on the right, I believe the mosaic is an acceptable representation of the total scene.

Reflections and Future Improvements

From my analysis, the main issue I found with all the resulting mosaics is the clear seams that were produced from the stitching process. Unfortunately, I was not able to figure out how to apply the feathering algorithm that was referenced in the instructions in my algorithm, however I believe that it would greatly improve those seams and create even better results. I'd also like to improve on images that I filtered out due to low inliers since the image didn't contain enough feature points that matched the middle image. Instead of using the middle image for those rejected images, use a closer image in the sequence that still contains portions of the rejected image so that valid feature points can be matched between them, and calculate the homography based on that. That way, a wider mosaic would be possible since it would eliminate that middle image containing criteria. Outside of those two issues, I am satisfied with the mosaics I was able to generate with my program.

References

[1] R. Hartley and A. Zisserman, Multiple View Geometry in Computer Vision Second Edition, Cambridge: Cambridge University Press, 2003.

[2] P. Kovesi, "MATLAB and Octave functions for Computer Vision and Image Processing," [Online]. Available: https://www.peterkovesi.com/matlabfns/index.html. [Accessed 5 November 2022].