CS 194-26: Image Manipulation and Computational Photography, Fall 2022

Project 4: [Auto]Stitching Photo Mosaics

Ethan Gnibus

Part 1: IMAGE WARPING and MOSAICING

Overview

In this project I will take images collected with my phone and warp them together into a seamless mosaic. To do this, I must first take pictures, define correspondences inside of each image, use these correspondences to recover homographies, then use these homographies to warp the image so that I could turn them into mosaics. I will create both rectified images and mosaics.

Part 1.1: Shoot the Pictures

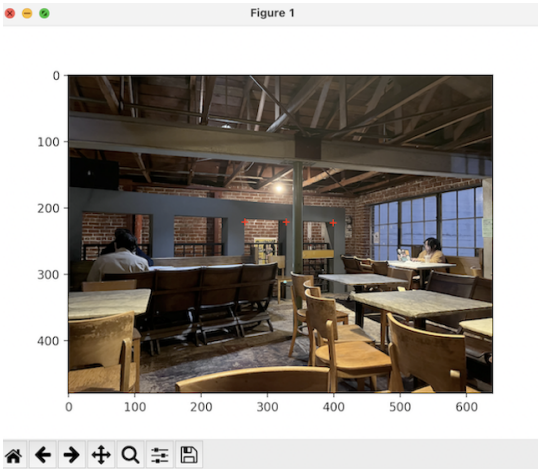
I chose to shoot two pictures outside of FSM in Berkeley. These are what the pictures look like with no editing:

FSM 1

FSM 2

Part 1.2: Recover Homographies

To make a mosaic between both images, I chose to forward warp from FSM 1 to FSM 2. This involved me finding the matrix H, such that if we have a point p on FSM 1 and a point p' on FSM 2, then H @ p ≈ p'. Doing this required me to make a GUI to select 4 or more correspondences on both images. I ended up choosing 19 correspondences and used least squares to approximate the best H for the homography. Below is the GUI I made.



GUI

Part 1.3: Warp the Images

Now that I have a homography, I can warp my original image into the homography!

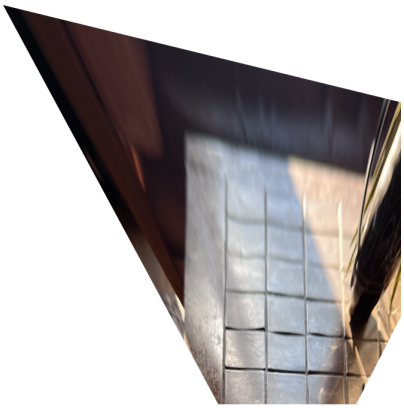
FSM 1

FSM 1 Warped

Part 1.4: Image Rectification

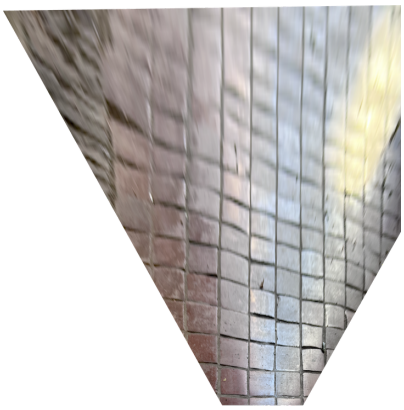
Using this warping technique, we can now rectify images so that we can change the perspective of the camera looking at them!

Pattern on my table



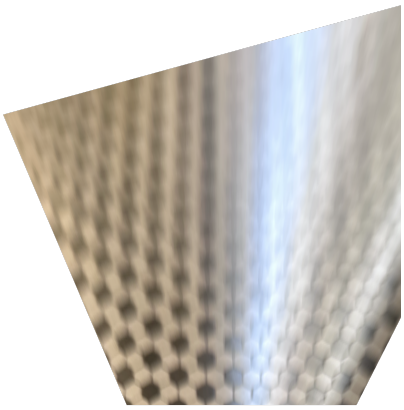
Rectified pattern on my table

Pattern on my walkway



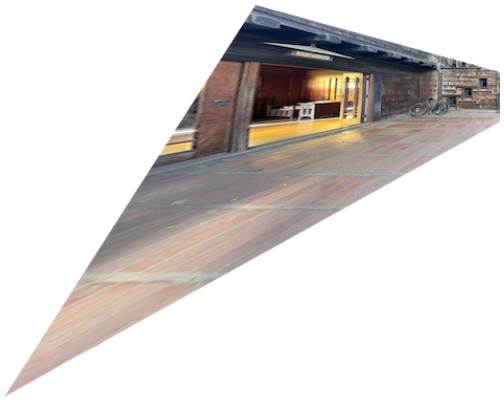
Rectified pattern on my walkway

Pattern on my bathroom floor



Rectified pattern on my bathroom floor

My back wall



My back wall rectified to face front

If we crop the rectified image, we can see the back wall facing towards us

My back wall rectified and cropped to face front

Part 1.5: Blend the images into a mosaic

Now that we can rectify images, we can warp one image then translate the other to overlap with the first. By Using a laplacian pyramid, we can blend both of these images to make a smooth mosaic

FSM 1

FSM 2

Warped FSM 1

FSM 2

Mosaic

Part 1.6: What I learned

In this project, the coolest thing I learned is how to compute homographies! Using homographies, I can warp the perspective of a photo so that I could view it at different angles. My favorite example of this is when I warped the back wall of a historic house in Berkeley so that you could view it from the front. I think this has super cool use cases and I'm excited to use it more in the future.

Part 1.7: Bells and Whistles

I did not complete any bells and whistles.

Part 2: FEATURE MATCHING for AUTOSTITCHING

Overview

In this project I extend the previous part of the project by automatically finding correspondence points on the input images. This entails implementing automatic corner detection, finding feature descriptors for those corners, matching corners across images using

these feature descriptors, using RANSAC to compute the homography between images, then passing the result into my previous code to get a mosaic.

Part 2.1: Detecting corner features in an image

Here I implement a single-scale Harris Interest Point Detector without sub-pixel accuracy. Below I display a figure of the Harris corners overlaid on the input images.

FSM 1

FSM 2

FSM 1 with points

FSM 2 with points

Part 2.2: Extracting a Feature Descriptor for each feature point

Here I impement Adaptive Non-Maximal Suppression to extract feature descriptors for each feature point. Below I include the chosen corners overlaid on the input images.

FSM 1

FSM 2

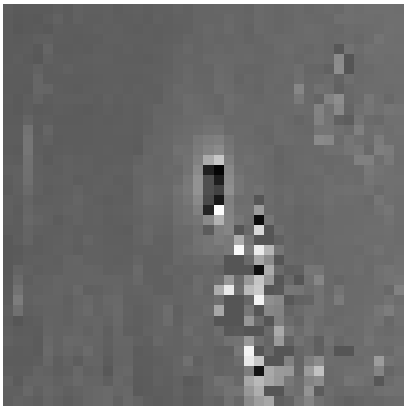

FSM 1 with all points

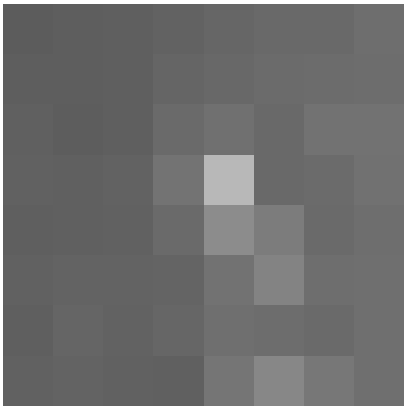

FSM 2 with all points

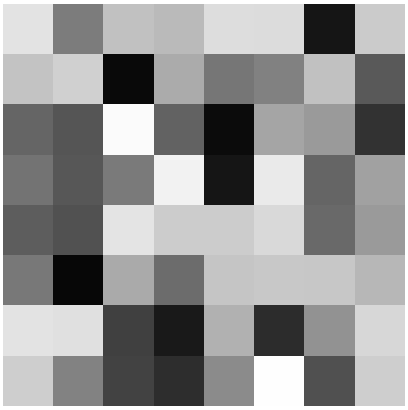

FSM 1 with chosen points


FSM 2 with chosen points

Here I implement feature descriptor extraction with no rotation invariance nor wavelet transform. To do this, I extract 8x8 patches sampled from larger 40x40 windows. I bias/gain-normalize the descriptors to improve accuracy. This is what one looks like:


40x40 patch


8x8 patch


Bias/gain normalized 8x8 patch

Part 2.3: Matching these feature descriptors between two images

Here I implement feature matching. To do this, I used nearest neighbors to find corresponding points on both images. For each point in Image A, I found it's first and second nearest neigbor points in Image B, 1-NN and 2-NN respectively. Next I calculated the Lowe Ratio (1-NN / 2-NN). I then filtered out points that had high Lowe Ratios by discarding them if they are above a certain threshold of my choosing. The reasoning behind this is that if two points on Image B have similar Squared-Sums of Differences with a point on Image A, then it's more likely that both aren't matches to the point on Image A. Below I show the result of this process, with FSM 1 acting as Image A and FSM 2 acting as Image B:


Matched points on FSM 1


Matched points on FSM 2

Part 2.4: Use a robust method (RANSAC) to compute a homography

In this part I implement RANSAC to compute a homography between my images. In the next part, I will feed this homography into my code from Part 1 to produce a mosaic!


4 points used to get homography on im1


4 points used to get homography on im2

Part 2.5: Produce some mosaics

Below I feed some sample images into my code from Part 2 then Part 1 to produce some mosiacs. I also compare these mosaics to the one I produce by hand-selecting their correspondences.

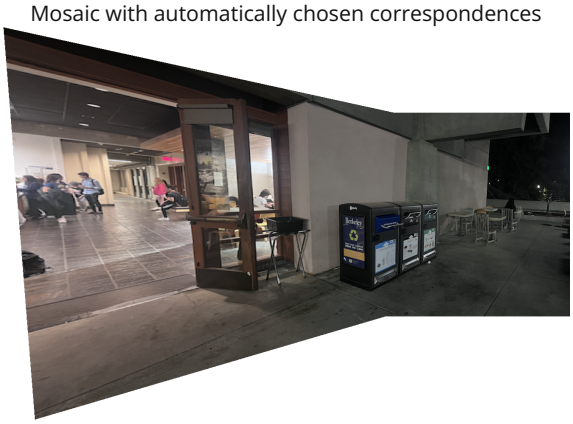
FSM


FSM 1


FSM 2

FSM 1 FSM 2

Mosaic with automatically chosen correspondences



Mosaic with manually chosen correspondences

Takeaways

Both images look good, but the points chosen by hand looks slightly cleaner than the points chosen automatically. In the automatic case, there is some fading around the "Berkeley" letters, because the points chosen are kind of close to each other. This makes the homography less accurate the further you go from those points. Since the "Berkeley" letters aren't in the points, they are subject to this artifact.

Butterfly


Butterfly 1


Butterfly 2


Mosaic with automatically chosen correspondences


Mosaic with manually chosen correspondences

Takeaways

Both look bad. If I fail to hold my camera steadily while taking the 2 pictures, I might change it's position too much resuling in a huge change in perspective. This makes the panorama bad even if my transform and automatic point recognition works perfectly.

Cory Courtyard


Cory 1


Cory 2

Mosaic with automatically chosen correspondences

Mosaic with manually chosen correspondences

Takeaways

Both work perfect! If I take pictures correctly and my automatic point recognition works, Then you can't decipher the transition from one to the next at all.

Part 2.6: What I learned

Making panoramas is much more difficult than I ever thought. I really liked writing the code to find points, but I found it really hard to write the code to find hompagraphies. I ended up enjoying the entire process after I understood it. I also found out that the quality of your panorama is highly dependent on how good you are at stabalizing your camera when you take pictures.

Part 2.7: Bells and Whistles

I did not complete any Bells and Whistles

https://inst.eecs.berkeley.edu/~cs194-26/fa22/upload/files/proj4A/cs194-26-ahn/