Project 3 Writeup

George Fang, Lily Davoren

Responsibility

Lily and George collaborated on all the code, but George focused more on the going further portion while Lily worked more on the required code.

Process

The key to this matter is that there was no alternation between generating data and editing code. We have a sample dataset to adapt all of our code to, and when it works on the sample code, then we can move on to our own data. So, in essence, Lily and George both wrote the starter stitcher.py (renamed to manual_stitcher.py) until it worked on the sample data, then we implemented the going further portion on the sample data, and finally we collected our own data. We just assumed it would work on a different dataset because, well, that's what these algorithms were designed to do. And if it didn't, we would adapt and tune the parameters for the algorithms (we didn't end up having to do this).

Difficulties

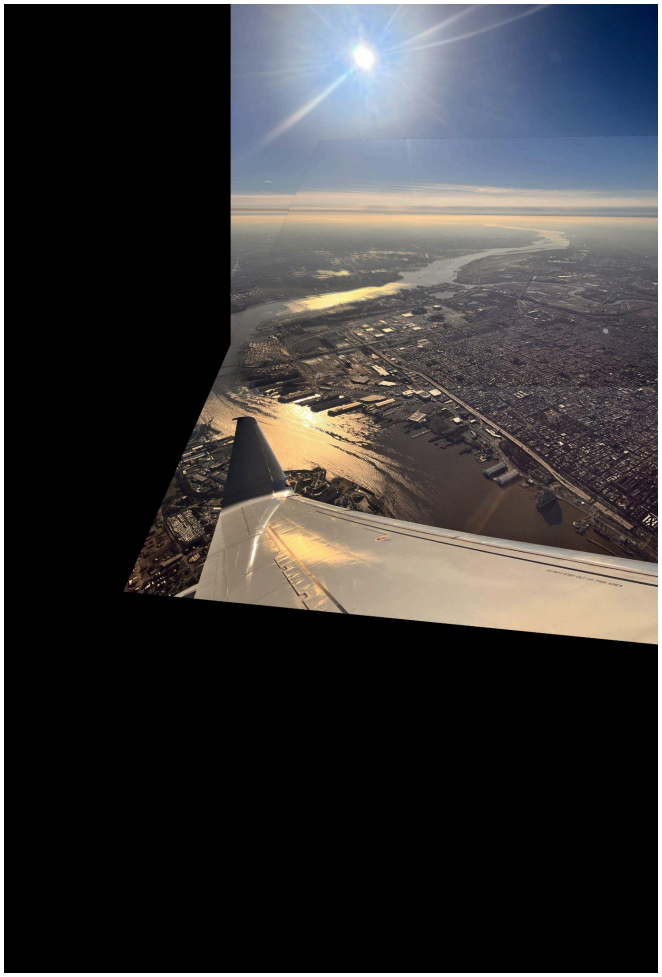
We encountered a few difficulties on the way, namely getting the viewport the right size, and understanding the ins and outs of the methods for finding homographies and generating points. The documentation for OpenCV isn't the best when it comes to figuring out point type conversions. We originally started with ORB because it was the best documented one, but we switched to using SIFT for better accuracy (and a less blurry final product on our data). So, we originally had to figure out how to do the point type conversions from the ORB output to the findHomography method because there wasn't a lot of documentation on it online (or we missed it). Then we discovered that SIFT found better matches between the images and had better documentations for type conversions but took more processing time (but you didn't specify a performance metric thankfully). When we started adding our own images, the process of converting an iOS HEIC to a compressed JPG was kind of awful, and manually naming the images to be matched was a hassle, so we automated the process (using 2x the amount of time) by changing the 2 images in the command-line arguments to a simple folder path. This also allows for easy expansion beyond 2 images (we didn't have time to do that, but future-proofing is nice). The biggest problems we had were with the viewport size. At first, George didn't know that the viewport was supposed to be the dimensions of the image, not the target points, so we had a lot of trouble getting the output viewport to size correctly. Then, the instructions told us to use Image B's points and the mapping of Image A's points through the perspective warp, but we found that also clipped the image in a way that was inconsistent with your sample output data for the weird jpegs. So, after quite a bit of reasoning and trial and error, we swapped the points being mapped through the homography and, voila, it worked. One thing that we will mention is that even though we fixed this on the sample datasets, some of our collected datasets still frame the stitched image output weirdly, and we're not sure why.

Going Further

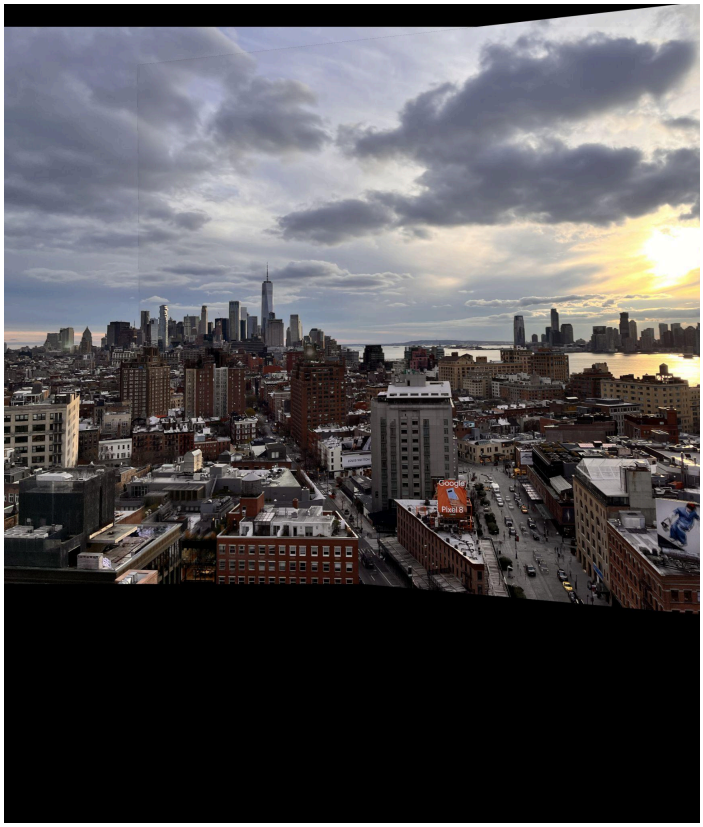
We used one of your examples to go further: “Find OpenCV code on the web and adapt it into your program to automatically find point correspondences between two images.” As mentioned earlier, we renamed the original stitcher program to manual_stitcher.py and created a new auto_stitcher.py. Also mentioned earlier is that the command-line usage is by path, not by images, so for example, you have to run “python auto_stitcher.py .\data\weird” to get started. There were a few more complexities to the going further portion. Instead of trying to just average the images, we attempted to blend them using the alpha blend from the past project, but found that it didn’t quite work on some images, but did on others (if we remember correctly it worked on the e52 board but not on the weird ones, for example). We trialed and errored a little bit, then switched to your suggested method in the instructions. Implementing that was more difficult because the normal bitwise operators wouldn’t work, so we had to find the more performant OpenCV ones, but even then, the ways they were defined were strange and what we ended up submitting probably isn’t the most performant code.

Program Outputs

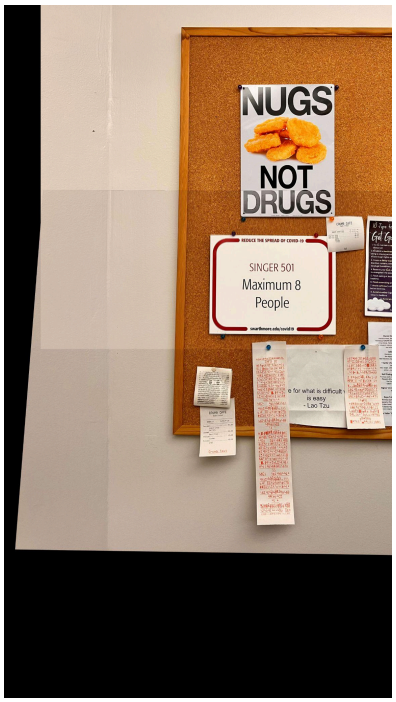
Plane output



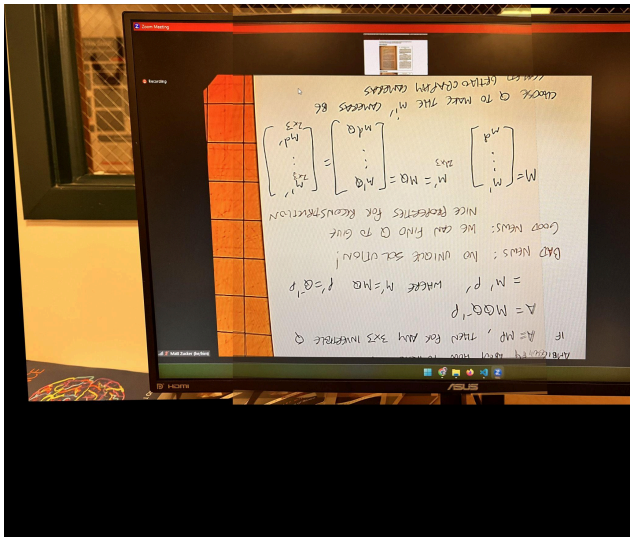
City output



SCCS board output



Lecture output



Lake output (cropped)

Drawing output