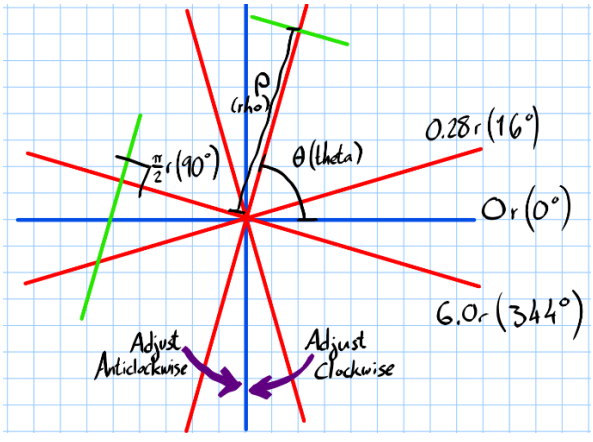
Nathaniel Roberts
CS390S – Digital Image Processing
Final Project

For my final project, I elected to work with Robb Hill on an automated image processing application. It was inspired by a real-life situation, where hundreds (thousands?) of physical family photographs had been digitally scanned, but were somewhat misaligned. To fix them all one at a time would be rather inefficient, besides being eye-wateringly tedious. Thus, it seemed that in this situation (one likely shared by many families), an algorithmic process to correct entire batches of photos was called for. Tragically, a brief internet search reveals that this software already exists (BatchCrop, for example, does both of these things and most of the other processing actions I would have liked to implement), but it nevertheless seemed a good opportunity to learn more about the process.

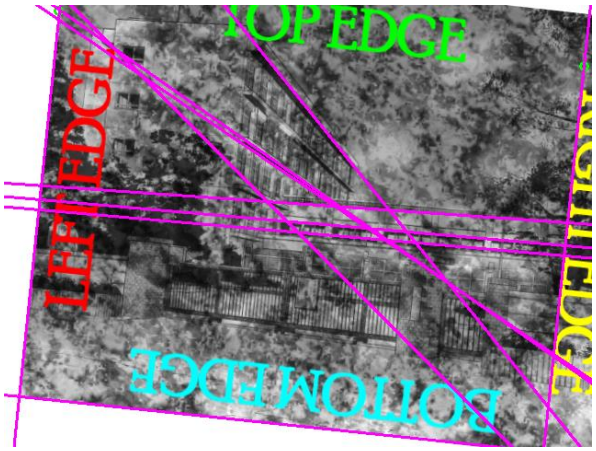
Our vision for the project started out somewhat broader in scope, but in view of the time requirements, we narrowed our goals down to two major ones: Align the photos correctly, and crop away any non-photograph information (e.g. the scanner bed). My portion of the program was to determine the alignment of the photos and adjust accordingly.

I began by establishing the foundational assumptions I would be making about our image set. Namely, I could reliably expect that 1) each image would have common markers suggesting its correct alignment, and 2) the adjustment each image needed would fall within a fairly small range of angular values. To identify these markers and the angles they suggested, we employed the Hough Line transform, as implemented in OpenCV.

I then attempted to define the markers and values. Because the images were all scanned using the same or similar methods and equipment, they were each bounded on anywhere from one to four sides by the stark white of the scanning bed. Thus, I concluded, if I were to find a set of straight lines in an image, I could sample the luminance of either side, expecting that if one side of a line were the scanner bed, it would be consistently at or near the maximum value, while the photograph would have substantially greater variance. The angles, meanwhile, which should be strictly vertical or horizontal, would vary from the cardinal directions, but not by very much. By trial and personal observation, I established a threshold of plus or minus roughly 16 degrees distance from 0/90/180/270 as the region in which I should expect to find these true alignment values.



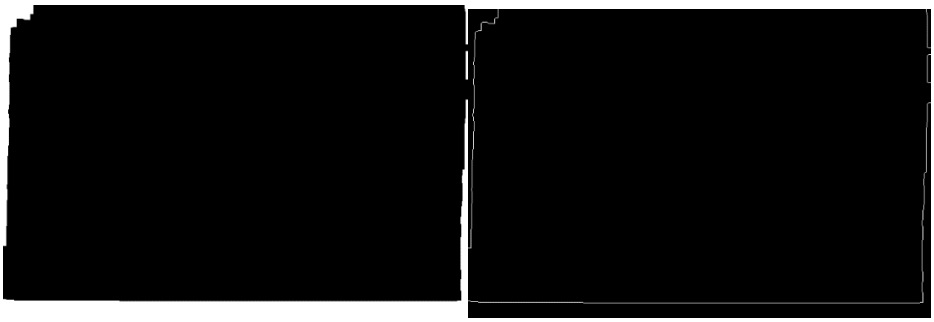
Not having had access to the photo sets in question initially, I created a test image containing identifiers for each "original" edge, then tilted it towards the various extremes of my expected range. After some tinkering, and with some help, I was able to consistently establish which lines belonged to the outer edges of the photo and find corners that might appear.



Determining how to rotate the image proved much trickier than I expected. OpenCV has methods which facilitate the rotation of matrix values, but they leave the underlying matrix structure unchanged; this was a potential problem, as a rectangular image being rotated within its original boundaries is apt to lose more information as some of it is rotated out of bounds. After some research, I came upon the Python Imutils package created by Adrian Rosebrock, which includes a function specifically meant to address this issue by expanding the size of the image matrix according to the need to maintain data. Thus equipped, I moved on, only to discover that my understanding of the values returned by a Hough Line transform was spotty at best. I was unable to find any particularly helpful resources in this regard, but after several (many) hours of trial and error, I determined that the angle returned by Hough could be used to rotate, but had to be modified first depending on its position to one side of cardinal or the other.

At this point, my code worked fairly well on most images, but not at all on others. Unable to determine why this should be, I resolved to try and rewrite the program using simpler, cleaner methods, in the hope that I would catch and correct whatever mistakes I had made in the 'first draft'. I must reluctantly confess that it was only at this point that I realized how few image processing techniques I was using, and

that I could make my life rather easier with a few more. Thus, in my second attempt at the code, I preceded my edge-finding with an extreme thresholding of the image – as I only needed the outside edges, which would be bounded by extreme luminance values, I could rule out a great deal of noise by ignoring most lower values. This, combined with using erosion and dilation to shore up small holes in the profile, leaving me with a tidy binary outline of the outer edges of the photograph alone. From this, key theta values were easily collected.



That will not be the end of this program, as there is much more to do, and I mean to do it; however, it is the point I have reached as of this moment. The rotation algorithm ought to be reworked to match the revised angle-finding, and I would have liked to be able to identify corners which lifted away from the scanning bed, giving them a distorted profile. Beyond that, I would like to (and will, in my spare time) incorporate some of the other techniques I have learned during the course which might lend themselves well to automation – noise removal, histogram analysis, gamma correction, that sort of thing. It will take time, of course; it turned out to be shockingly difficult to execute what, when it was described to me, sounded like the simplest of tasks.

I will not pretend it is the most original or innovative work in the field. After seeing some of the other presentations in class, it seems a bit shabby by comparison. That said, I stand by the work I did. We set out to create a solution to a real problem, something that would be an actual help to someone, and if we have not reached that goal yet, we are well on the way.

Language used: Python

Packages utilized: OpenCV, Imutils, Glob, NumPy

Resources utilized:

BatchCrop
https://batchcrop.com/

Imutils library
https://github.com/jrosebr1/imutils
https://www.pyimagesearch.com/2017/01/02/rotate-images-correctly-with-opencv-and-python/

OpenCV Project Documentation
https://docs.opencv.org/4.1.2/

NumPy Documentation
https://numpy.org/

List comprehension for loading image set courtesy of StackOverflow user ClydeTheGhost:
https://stackoverflow.com/questions/38675389/python-opencv-how-to-load-all-images-from-folder-in-alphabetical-order

Special thanks to Robb Hill for figuring out how to use OpenCV's HoughLines() method