CS543/ECE549 Assignment 1

Name: fanmin shi
NetId: fshi5

Part 1 : Implementation Description

For single alignment. My algorithm is the following:

1. Load image
2. If TIF image, then convert “L” mode.
3. Apply edge filter
4. Split image into B, G, R images
5. Normalize those images
6. Find best alignments for G and R over B.
    a. Shift +/- 15 pixels on X and Y axis
    b. Use cross correlation or SSD to calculate similarities
    c. Return the X and Y shifts with best cross correlation or SSD (will talk this in detail later)
7. Create the composited image from the B, G, R images usings computed alignments

Initially, I didn't have an edge filter nor normalization. I use similarities function “from sewar.full_ref import *” to see which function can produce the best result. The Spatial Correlation Coefficient produces fantastic results but takes 186.68171000480652 seconds to compute. This is too slow. However, if using only SSD or Cross Correlation, the image is not aligned.

SCC	SSD

Then I noticed that the alignment can become much easier if only edges are present because black and white pixels when aligned gives a lesser SSD error. So applying an Edge filter with an SSD. The compute time reduces to .6 seconds. The result is

SSD	SSD + Edge Filter
	

I noticed that for most images there is no difference from SSD or Cross Correlation. However, for the following image. The only SSD works well.

SSD + Edge filter	Cross Correlation + Edge filter
	

The above is the limitation of my algorithm because I cannot be 100 percent sure if SSD or Cross Correlation will work. So I have to eyeball the result and adjust the similarity function accordingly.

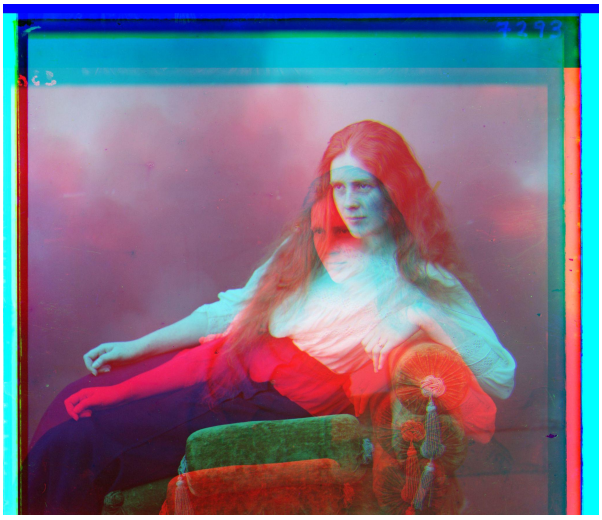
For Multi-Alignment, My algorithm is the below.

1. Load image
2. If TIF image, then convert “L” mode.
3. Iterate pyramid image with height of 3 (will discuss this later) starting at the lowest resolution.
    a. Apply edge filter and normalization on the image.
    b. For the smallest image
        i. Compute the best alignment from a +/- 15 pixels search box centering at (0, 0)
    c. For rest of the images
        i. Use the previous alignment to determine the starting position. E.g Prev alignment (5, 6) -> (10, 12) because the current image is scaled by 2.
        ii. Use the new centering position and compute the best alignment from a +/- (**5 X pyramid_depth)** pixels search box
4. Use the computed best alignment for G and R channels and compose the final image.

Initially I have the searching box +/-5 for the rest of the images. The issue with that is that as the resolution increases a fixed (5,5) searching box is too small to find the optimal alignment. However, if I scale the search box by **pyramid_depth**, then the search box is large enough to find the optimal alignment. In theory, I should double the search box e.g (5,5) -> (10, 10) -> (20, 20) so that the search box stays relatively the same in comparison to increasing image resolution as we go down the image pyramid. However, the larger search box makes the runtime too slow. Hence I optimized it to use linear increase of search range.

+/-5	+/-5 * **pyramid_depth**
	

I found SSD and Cross Correlation made no difference on most of larger images except 01657u.tif.

SSD	Cross Correlation
	

Hence, the limitation of my algorithm is that I need to eyeball to see which similarities function works the best.

Part 2: Basic Alignment Outputs

For each of the 6 images, include channel offsets and output images. Replace <C1>, <C2>, <C3> appropriately with B, G, R depending on which you use as the base channel.

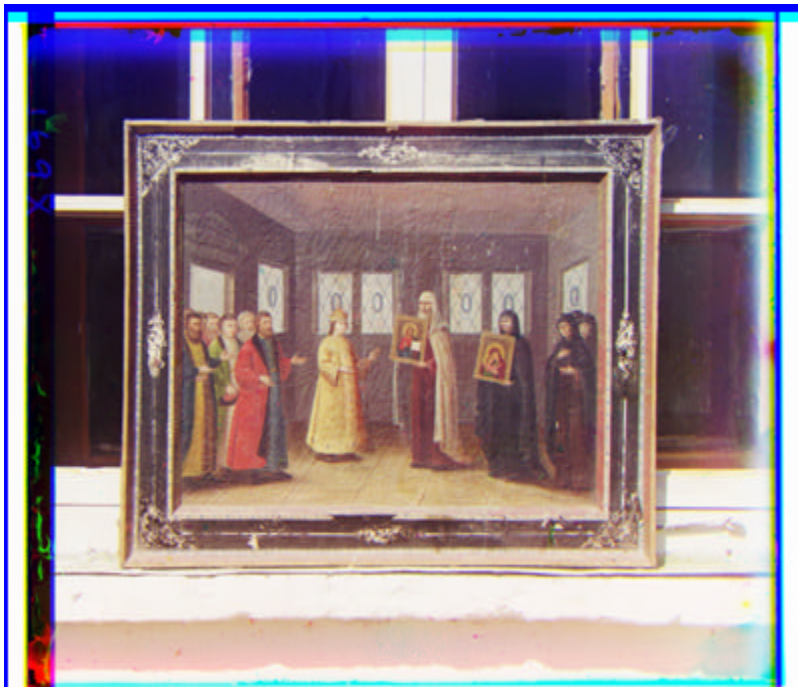
A: Channel Offsets

Using channel <C1> as base channel:

Image	g (h,w) offset	r (h,w) offset
00125v.jpg	(-1, 5)	(0, 10)
00149v.jpg	(2, 4)	(2, 9)
00153v.jpg	(2, 7)	(3, 15)
00351v.jpg	(0, 4)	(1, 13)
00398v.jpg	(0, 5)	(4, 11)
01112v.jpg	(0, 0)	(1, 5)

B: Output Images

Insert the aligned colorized outputs for each image below (in compressed jpeg format):



Part 3: Multiscale Alignment Outputs

For each of the 3 high resolution images, include channel offsets and output images. Replace <C1>, <C2>, <C3> appropriately with B, G, R depending on which you use as the base channel. You will also need to provide an estimate of running time improvement using this solution.

A: Channel Offsets

Using channel <C1> as base channel:

Image	G (h,w) offset	R (h,w) offset
01047u.tif	(18, 24)	(32, 72)
01657u.tif	(0, 58)	(0, 122)
01861a.tif	(38, 70)	(62, 146)

B: Output Images

Insert the aligned colorized outputs for each image below (in compressed jpeg format):


7293

C: Multiscale Running Time improvement

Single Alignment: 749.4506392478943 seconds
Multiscale Alignment: 20.944703817367554 seconds

Part 4 : Bonus Improvements

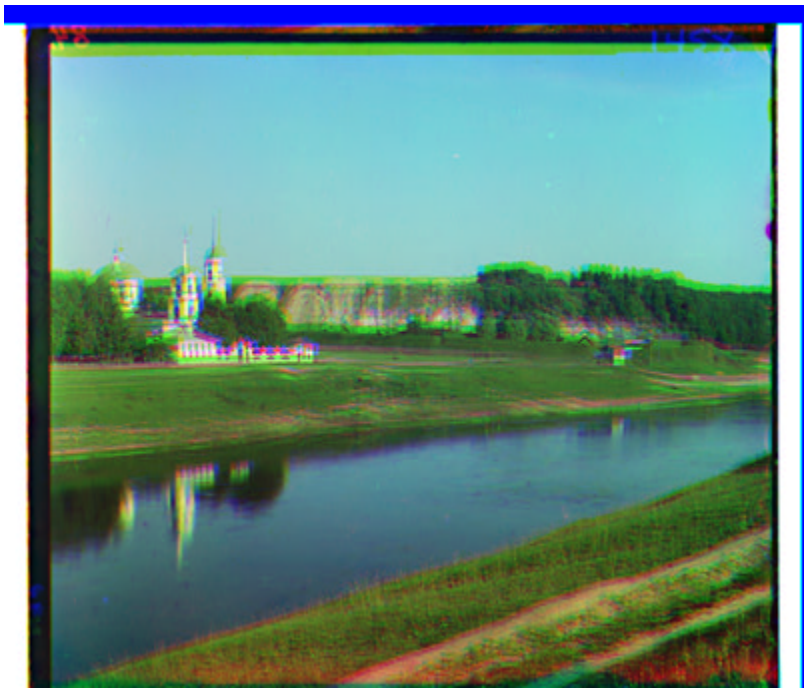
Post any extra credit details with outputs here.

Improvement: Add edge filter + normalization.

Adding an edge filter improves the alignment. I suspect adding edge filters simplify the alignment process when only edges need to be aligned. When edges are aligned, the SSD will

produce a much lesser value than the non-edge filtered version. Hence, finding the right alignment will be obvious.

Before Edge filter,



After edge filter

Use provided Similarity Function the Spatial Correlation Coefficient. The result is high quality right out of the box with no pre-processing which is quite impressive.