Project 1

Maryam Darei **Computational Photography**
Dr. Pless

Aligning and compositing the images of the Prokudin-Gorskii photo collection

Background

Sergey Mikhaylovich Prokudin-Gorsky was a chemist and photographer of the Russian Empire. He is best known for his pioneering work in color photography and his effort to document early 20th-century Russia.

Starting in 2000, the negatives were digitized and the color triples for each subject were digitally combined to produce hundreds of high-quality color images. He is famous because of his idea about color photography, while it was first expressed by James Clerk Maxwell in 1855. James Clerk Maxwell could not reach good results because of the lack of availability of photographic materials.

This idea is based on the method the human eye perceives color that the visible spectrum of colors was divided into three channels of information. According to this, he recorded three black and white images, one through a red filter, one through a green one, and one through a blue filter.

Data Description:

Data Type: Black and white Images
Abstract: Color images are aesthetically important, and many black-and-white images of the past are not obscured by Brexit.

Data Characteristics:

- Each image includes 3 images that are stick to each other vertically and are, from top to bottom, in blue, green, and red order.
- Attribute breakdown: 8 quantitative input variables, and 1 quantitative output variable
- Missing Attribute Values: None

Sample of input:

Data pre-processing and Feature Engineering:

1- **removing borders:** It is important that 3 channels of each image align exactly on each other. Some of those pictures have borders. So, removing white, black, or other color borders can be very useful, and it can be a kind of data pre-processing.

2- **Adjust contrast:** We can modify the image contrast with some methods to improve the images' quality.

3- **Horizontal and vertical transitions:** The difference in the spatial position of the color channels (B, G, and R) causes a slip in the process of overlapping the three channels to form an RGB image, and we have color shadows in the produced images. In other words, the photo will not be of good quality. To solve this problem, the coordinate positions of all three channels must be exactly the same.

4- **Equalize the scales of all three channels:** If the dimensions of the three channels in a photo are different, aligning these 3 Blue, Green, and, Red channels will not produce the correct RGB image. Therefore, it is important to rescale these channels to obtain the same dimension.

5- **Rotation correction:** It is possible that all three channels of a photo are not in the same direction or there is any kind of rotation or angle difference in each of them compared to the other. This difference must be corrected in the direction to align channels and have a correct RGB image.

Code description:

I have written it in **MATLAB**.

In the first step, after reading the input image, I divide each image into 3 parts horizontally in the main body of the program.

```
%main
%Read the image
img = imread('/inputimage.jpg');
[n,m] =size(img);

% divide the input image to 3 equal parts:
n1=floor(n/3);
n2=2*n1;
n3=3*n1;

B=img(1:n1,1:m); % Blue channel
G=img(n1+1:n2,1:m); % Green channel
```

I developed a function to merge these 3 channels to each other and produce an RGB image. I developed it in two different approaches ("**MergetoRGB", and "MergetoRGB2**") that are very similar.

In the first one, based on the Professor's advice, I assumed the Green as a reference and cropped a small window of the red and blue channel, for example, a 20X20, 50X50, or 100X100 in the middle of each red and blue channel. It is parametric and you should set it as input when you call the function. For each red and blue channel, I use a two-nested "For" loop and shift the selected parts to reach the best position. The amount of the shift ((e.g. [-5,5], [-15,15], or [20,20] pixels) is another input that you should set when you call the function. To acquire the best position, I use the Sum Squared Differences as a score and decide the best amount of the shift based on this value. In the first approach, I use the "circshift" for a shift matrix.

In the second approach, everything is the same as a first approach, but I made a little difference. Instead of using the "circshift" for a shift an image matrix, I use a new copy of the blue and red channel based on "i" and "j (the amount of search window in every step of the two-nested "For" loop). It means, I did not use a "circshift". The reason of using this method is because with using of the "circshift", in every shift step we replace the left amount with the rightest amount and indeed produce extra value.

I run these two functions and the program with a different value for search window length and different dimensions cropped of the image's channels and compare the result.

First Approach (“MergetoRGB”, d is the cropped window length, a is the search window length):

1- d = 20 pixels, a = 2:

Total time: 0.683 s

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
P1	1	0.551	0.002	
MergetoRGB	1	0.515	0.019	
subplot	5	0.268	0.159	
imshow	5	0.211	0.039	

Original RGB Image

Blue channel

Green channel

Red channel

RGB Image

2- d = 20 pixels, a = 2:

Total time: 0.803 s

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
P1	1	0.806	0.001	
MergetoRGB	1	0.772	0.141	
subplot	5	0.386	0.244	
imshow	5	0.229	0.043	

Original RGB Image

Blue channel

Green channel

Red channel

RGB Image

3- d = 20 pixels, a = 50:

Total time: 0.832 s

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
P1	1	0.672	0.002	
MergetoRGB	1	0.609	0.097	
subplot	5	0.274	0.150	
imshow	5	0.220	0.042	

Original RGB Image

Blue channel

Green channel

Red channel

RGB Image

4- d = 20 pixels, a = 100:

Total time: 0.969 s

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
P1	1	0.824	0.002	
MergetoRGB	1	0.768	0.295	
subplot	5	0.253	0.117	
imshow	5	0.205	0.040	

Original RGB Image

Blue channel

Green channel

Red channel

RGB Image

5- d = 50 pixels, a = 2:

Total time: 0.718 s

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
P1	1	0.563	0.003	
MergetoRGB	1	0.498	0.020	
subplot	5	0.247	0.111	
imshow	5	0.214	0.041	

Original RGB Image

Blue channel

Red channel

RGB Image

6- d = 50 pixels, a = 20:

Total time: 0.780 s

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
P1	1	0.639	0.002	
MergetoRGB	1	0.573	0.031	
subplot	5	0.316	0.147	
imshow	5	0.207	0.038	

Original RGB Image

Blue channel

Green channel

Red channel

RGB Image

7- d = 50 pixels, a = 50:

Total time: 0.879 s

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
P1	1	0.740	0.003	
MergetoRGB	1	0.684	0.177	
subplot	5	0.299	0.166	
imshow	5	0.192	0.035	

Original RGB Image

Blue channel

Green channel

Red channel

RGB Image

8- d = 50 pixels, a = 100:

Total time: 0.850 s

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
P1	1	0.708	0.002	
MergetoRGB	1	0.664	0.260	
subplot	5	0.215	0.098	
imshow	5	0.175	0.033	

Original RGB Image

Blue channel

Green channel

Red channel

RGB Image

Second Approach ("MergetoRGB2"):

1- d = 20 pixels, α = 2:

Total time: 0.652 s

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
P1	1	0.504	0.004	
MergetoRGB2	1	0.422	0.016	
subplot	5	0.210	0.096	
imshow	5	0.181	0.033	

Original RGB Image

Blue channel

Green channel

Red channel

RGB Image

2- d = 20 pixels, a = 20:

Total time: 0.653 s

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
P1	1	0.481	0.002	
MergetoRGB2	1	0.431	0.021	
subplot	5	0.214	0.097	
imshow	5	0.182	0.035	

Original RGB Image

Blue channel

Green channel

Red channel

RGB Image

3- d = 20 pixels, a = 50:

Total time: 0.684 s

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
P1	1	0.557	0.002	
MergetoRGB2	1	0.512	0.048	
subplot	5	0.228	0.108	
imshow	5	0.220	0.040	

Original RGB Image

Blue channel

Green channel

Red channel

4- d = 20 pixels, a = 100:

Total time: 0.739 s

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
P1	1	0.613	0.002	
MergetoRGB2	1	0.564	0.152	
subplot	5	0.213	0.099	
imshow	5	0.185	0.036	

Original RGB Image

Blue channel

Green channel

Red channel

RGB Image

5- d = 50 pixels, a = 2:

Total time: 0.595 s

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
P1	1	0.467	0.002	
MergetoRGB2	1	0.423	0.019	
subplot	5	0.211	0.095	
imshow	5	0.179	0.033	

Original RGB Image

Blue channel

Green channel

Red channel

RGB Image

6- d = 50 pixels, a = 20:

Total time: 0.627 s

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
P1	1	0.505	0.002	
MergetoRGB2	1	0.449	0.036	
subplot	5	0.216	0.101	
imshow	5	0.182	0.035	

Original RGB Image

Blue channel

Green channel

Red channel

RGB Image

7- d = 50 pixels, a = 50:

Total time: 0.714 s

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
P1	1	0.589	0.002	
MergetoRGB2	1	0.522	0.104	
subplot	5	0.216	0.099	
imshow	5	0.187	0.036	

Original RGB Image

Blue channel

Green channel

Red channel

RGB Image

8- d = 50 pixels, a = 100:

Total time: 0.953 s

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
P1	1	0.820	0.002	
MergetoRGB2	1	0.777	0.368	
subplot	5	0.216	0.098	
imshow	5	0.178	0.033	

Original RGB Image

Blue channel

Green channel

Red channel

RGB Image

Result Analyze:

It seems the second approach is better than the first one. It is completely clear that in the first approach with increasing the amount of “a” (the Search window in for loops) we could mess in the RGB image check the resulting numbers 2, 3, 4, 7, and 8 in the first approach). For the first approach, the best result is when we choose d=50 and a=20 (It is good to mention again that a is the dimension of the search window and d is the dimension of the cropped image).
In the second approach, we have a better result and less mess in the RGB images. In this case, the best results are for the d= 50 and a=20 or a=50.

Result summary table:

		D = 20	D = 50
First approach	a=2		
	a=20		
	a=50		

	a=100		
Second approach	a=2		
	a=20		
	a=50		
	a=100		

Run Time:

If you check the run time in 16 cases, you can understand the run time is increased with a bigger cropped image and with the bigger shift window.

High-Resolution Images:

Unfortunately, these two idea does not work really well for high-resolution images because we cropped just a part of the images.

Other ideas:

1- **Optical Flow Estimation:** It is the problem of finding pixel-wise motions between consecutive images. Approaches for optical flow estimation include correlation-based, block-matching, feature tracking, energy-based, and more recently gradient-based. Using this idea can be useful in finding the amount of pixel relocation in each channel in comparison to other channels.
2- **Second Idea:** We can calculate the sum of all the values in each column and have a value for each. Then order all these values and produce a vector. Then we have a vector for every channel. It may be possible to understand the amount of pixel relocation in each channel in comparison to other channels by the correlations between these tree vectors.