CSE 468/568 Lab 3: Colorizing the Prokudin-Gorskii photo collection

Sergei Mikhailovich Prokudin-Gorskii (1863-1944) was a photographer who, between the years 1909-1915, traveled the Russian empire and took thousands of photos of everything he saw. He used an early color technology that involved recording three exposures of every scene onto a glass plate using a red, green, and blue filter. Back then, there was no way to print such photos, and they had to be displayed using a special projector. Prokudin-Gorskii left Russia in 1918, following the Russian revolution. His glass plate negatives survived and were purchased by the Library of Congress in 1948. Today, a digitized version of the Prokudin-Gorskii collection is available online at: http://www.loc.gov/exhibits/empire/gorskii.html

The goal of this assignment is to learn to work with images in Python by taking the digitized Prokudin-Gorskii glass plate images and automatically producing a color image with as few visual artifacts as possible. In order to do this, you will need to extract the three color channel images, place them on top of each other, and align them so that they form a single RGB color image. The assignment file (`lab3.tar.gz`) contains six images (image*.jpg) that you should run your algorithm on.

Your program should take glass plate images as input and produce color images as output. The program should divide the input image into three equal parts (ignoring 1 or 2 leftover rows at the end) and align the first and the second parts (B and G) to the third (R). You will also need to print the (y,x) displacement vectors that was used to align the parts. Detailed description is below. Example images are shown in Figure below.

Figure 1: Sample image given to you as three separate images, one each for each color channel (left); Unaligned color image (center); Aligned color image (right)

Simple Color Images

There should be six images (`image*.jpg`) in lab3.tar.gz. Each image is a concatenation of three separate glass plate images, one for each color channel (B, G and R). Your first task is to create a file `main.py` that accepts an image `filename.jpg` as an argument, reads the image file, extract the three plate images as three color channel images, and save the resultant color image. The color image will look blurred as the channels are not aligned correctly. This is OK. Save the output image with the name `filename-color.jpg`
By executing:
`python main.py image1.jpg`
your program should run and generate: `image1-color.jpg`

Aligning Images

As described above, just overlaying the individual channel images creates blurred images. In this step, we will find the correct alignment. The easiest way to align the parts is to exhaustively search over a window of possible displacements (say [-15..15] pixels in X and Y directions). Score each displacement using some image matching metric, and take the displacement with the best score.

There is a number of possible metrics that one could use to score how well the images match. The simplest one is just the L2 norm distance, also known as the Sum of Squared Differences (SSD). Another metric is normalized cross-correlation (NCC), which is simply a dot product between two normalized vectors. We will describe and use both metrics in this assignment.

SSD is calculated as:

$$\sum_{x=1}^{n}\sum_{y=1}^{m}(image1[y][x] - image2[y'][x'])^2$$

NCC is calculated as:

$$\sum_{x=1}^{n}\sum_{y=1}^{m} normImage1[y][x] * normImage2[y'][x']$$

Where n is the width of the image and m is the height of the image. x' and y' are coordinates on image2, that we are overlaying on coordinates x and y from image1. e.g. for a displacement vector $[v, u]$, we have $x' = x - u$ and $y' = y - v$.

The formula for NCC is similar to that of SSD. However, you can see that the NCC formula expects *normImage* rather than *image*. This is the normalized image, in order to obtain better results. You need to normalize the pixels in each image (color channel).

Normalize each channel by subtracting the mean value of intensities, and dividing by the standard deviation. Each pixel of the normalized image is calculated as below:

$$normImage[y][x] = \frac{image[y][x] - averageImagePixelValue}{\sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m}(image[j][i] - averageImagePixelValue)^2}}$$

Note that the $averageImagePixelValue$ is constant for all pixels, and is initially calculated as below:

$$averageImagePixelValue = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} image[j][i]}{m * n}$$

Write two methods (one for SSD and one for NCC), that take two images (two color channels) and an displacement vector $[v, u]$, and calculate matching score of the two images based on the offset. Using these methods, calculate the best alignment and print out the results. The result is in the form of $[y, x]$ displacement vectors; two for SSD and two for NCC.

Finally, you should create a color image based on the calculated alignments for each algorithm, and write them as `filename-ssd.jpg` and `filename-ncc.jpg`.

Some tips:

- Note that the image from top to bottom is BGR, but the order of channels is RGB!
- When working with the images, note the order of columns (x) and rows (y).
- Your task is to align the G and B channels with the R channel
- Depending on how you shift the images, the border pixels will mess with the calculation (Especially SSD), giving false optima. Test different methods for shifting and/or drop the border pixels in the calculation. e.g. you can consider only matching the center part of the images.

Submission Instructions

You will submit `lab3.tar.gz`, a compressed archive file containing the lab3 folder. The folder should contain the original images, along with the main.py file.

Upon running:
`python main.py filename.jpg`
your program should open filename.jpg and produce 3 color images, and print a total of 4 displacements vectors:

- `filename-color.jpg` is the unaligned color image
- Best R-G displacement based on SSD
- Best R-B displacement based on SSD
- `filename-ssd.jpg` is the aligned color images based on SSD
- Best R-G displacement based on NCC
- Best R-B displacement based on NCC
- `filename-ncc.jpg` is the aligned color images based on NCC

Be descriptive in your print statements:

```
(filename) SSD Alignment
Green = [7,5]
Blue = [-3,2]


(filename) NCC Alignment
Green = [6,4]
Blue = [-4,3]
```

Note the way displacement is reported. We want to know how much the G image should be moved on the R image, in order to obtain best results. The R-G displacement is the translation from R origin, to G origin. The R-G displacement of [7,5] here, means that the G image is best aligned on R image, when the first pixel of the G image (row=0, column=0), is translated to overlap on this pixel on the R image (row=7, column=5).

Please take care to follow the instructions carefully so we can script our tests. Problems in running will result in loss of points.

Here is the reference command to compress your folder (the result will be in the home folder):

```
$ tar -czvf ~/lab3.tar.gz lab3
```

Please use CSE autolab for submission.
The assignment is due Friday, April 9 before midnight.

Acknowledgements

This assignment is adopted from Radhakrishna Dasari with permission, though originally designed by Aloysha Efros at CMU.

Supplementary

Python libraries
You will need to use PIL and numpy for this lab. See the code example:

```
from PIL import Image
import numpy as np

im = np.array(Image.open('ava.jpeg'))

print(im.shape)
print(im[0][0])

for i in range(500, 2000):
    for j in range(500, 2000):
        im[i][j][0] = 0

new_im = Image.fromarray(im)
new_im.show()

new_im.save("new_cat.png")
```

For how you should read, manipulate, and save an image using these two libraries. If, for some reason, you don't like this method of reading and writing images, you can ask us if you can use a different library.

You cannot use openCV.
The task of writing your own feature detection, feature matching, homography estimation, and image alignment is far harder than doing this assignment with the SSD and NCC method. Currently, the acceptable libraries would include official python libraries (not github repositories) that help you with SSD and NCC calculation. Unacceptable libraries are those that do template matching.

Slow run time
It's possible that your code can run for a long time (around 10 minutes). This is fine, because this is a computationally heavy task. If this bothers you, you can look into list vectorization, a technique to re-write your for loops into operations that map across a list. This can speed up run-time by quite a bit, but is a little unintuitive to do. But, again, you don't have to worry about making your code run faster.