CSCE 489/689 - Computational Photography

Assignment 2

Deadline: Feb. 14th

1 Background

Sergei Mikhailovich Prokudin-Gorskii (1863-1944) was a man well ahead of his time. Convinced, as early as 1907, that color photography was the wave of the future, he won Tzar's special permission to travel across the vast Russian Empire and take color photographs of everything he saw including the only color portrait of Leo Tolstoy. And he really photographed everything: people, buildings, landscapes, railroads, bridges... thousands of color pictures! His idea was simple: record three exposures of every scene onto a glass plate using a red, a green, and a blue filter. Never mind that there was no way to print color photographs until much later – he envisioned special projectors to be installed in "multimedia" classrooms all across Russia where the children would be able to learn about their vast country. Alas, his plans never materialized: he left Russia in 1918, right after the revolution, never to return again. Luckily, his RGB glass plate negatives, capturing the last years of the Russian Empire, survived and were purchased in 1948 by the Library of Congress. The LoC has recently digitized the negatives and made them available on-line.

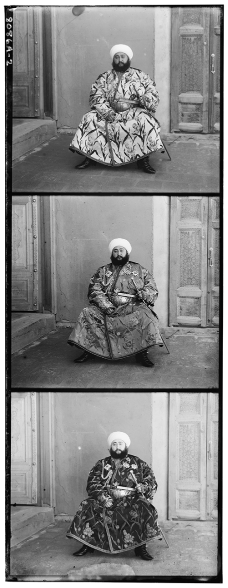
2 Goal

The goal of this assignment is to take the digitized Prokudin-Gorskii glass plate images and, using image processing techniques, automatically produce a color image with as few visual artifacts as possible. This should be done by extracting the three color channel images, placing them on top of each other, and aligning them, so that they form a single RGB color image. The starter code, provided below, performs all the operations with the exception of alignment. Your job is to perform alignment by implementing the `FindShift` function. We will assume that a simple (x, y) translation model is sufficient for proper alignment.

3 Starter Code

Starter code (in Python) along with the images can be downloaded from here. Note that, expected results for a few images are included in the "Results" folder.

4 Task1



The digitized glass plate images (an example shown on the right) are in BGR order from top to bottom. Your program takes a glass plate image as input and produce a single color image as output. The program should divide the image into three equal parts and align the second and the third parts (G and R) to the first (B). For each image, you will need to print the (x, y) translation that was used to align the parts.

The easiest way to align the parts is to exhaustively search over a window of possible displacements (say [-20, 20] pixels), score each one 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. An example of such a metric is sum of squared differences SSD defined as:

$$\text{SSD}(I_1, I_2) = \sum_{i=1}^{W} \sum_{j=1}^{H} (I_1(i,j) - I_2(i,j))^2, \quad (1)$$

To do this, you should write a `for` loop to searching over a user-specified window of displacements. For each displacement, compute the score and save it into an array. At the end, you should choose the displacement with the best score (minimum SSD). You should test your code on the "jpg".

In the next task, you will be implementing a multi scale algorithm to be able to handle "tif" images with extremely high resolution.

5 Task 2

Exhaustive search will become prohibitively expensive if the pixel displacement is too large (which will be the case for high-resolution glass plate scans). In this case, you will need to implement a faster search procedure such as an image pyramid. An image pyramid represents the image at multiple scales (usually scaled by a factor of 2) and the processing is done sequentially starting from the coarsest scale (smallest image) and going down the pyramid, updating your estimate as you go.

Specifically, you need to create an image pyramid by resizing your image multiple times (you can use `skimage.transform.resize` for resizing). For example, if you choose pyramid with 4 levels and your image has a resolution of 1024×1024, you will have four images with resolutions of 1024×1024, 512×512, 256×256, and 128×128. You start finding the shift in the smallest image. Since the image is small in this scale, a small search range ([-20, 20]) would be sufficient. Once the translation (x, y) at this scale is found, you properly scale this translation to the finer scale and use it as the initial translation vector. At this finer scale, you will only search around this initial translation vector. You continue this process until reaching the finest scale.

6 Hints

- You need to implement almost everything from scratch (except functions for reading, writing, resizing, shifting, and displaying images; some of these like reading and writing are already done in the starter code). In particular, you are not allowed to use high level functions for constructing pyramids, computing matching scores, etc.

- The average running time is expected to be less than 1 minute per image (ideally a few seconds). If it takes hours for your program to finish, you should further optimize the code.

- Avoid too many `for` loops by vectorizing/parallelizing your code (See more details here and here).

- Do not get bogged down tweaking input parameters. Most, but not all images will line up using the same parameters. Your final results should be the product of a fixed set of parameters (if you have free parameters). Do not worry if one or two of the handout images do not align properly. In particular, the emir image is difficult to align with basic SSD metric. For that better features (see extra credit section) is needed.

- Shifting a matrix can be done using the `circshift` function which is provided in the starter code.

- Compute your metric on the internal pixels only, to avoid using the invalid border pixels.

7 Extra Credit

- **Automatic cropping** Remove white, black or other color borders. Don't just crop a predefined margin off of each side – actually try to detect the borders or the edge between the border and the image.

- **Automatic contrasting** It is usually safe to rescale image intensities such that the darkest pixel is zero (on its darkest color channel) and the brightest pixel is 1 (on its brightest color channel). More drastic or non-linear mappings may improve perceived image quality.

- **Automatic white balance** This involves two problems – 1) estimating the illuminant and 2) manipulating the colors to counteract the illuminant and simulate a neutral illuminant. Step 1 is difficult in general, while step 2 is simple (see the Wikipedia page on Color Balance and section 2.3.2 in the Szeliski book). There exist some simple algorithms for step 1, which don't necessarily work well – assume that the average color or the brightest color is the illuminant and shift those to gray or white.

- **Better features** Instead of aligning based on RGB similarity, try using edges. This will particularly help with "emir.tif".
- **Better transformations** Instead of searching for the best x and y translation, additionally search over small scale changes and rotations. Adding two more dimensions to your search will slow things down, but the same course to fine progression should help alleviate this.

8 Write up

Include all the images with their corresponding (x, y) shifts in your report. Make sure you only include the jpg images for task 1 (single scale) and tif images for task 2 (multi scale). Describe the method and all the parameters used to obtain the results. Also discuss any extra credit you did.

9 Graduate Credit

Graduate students have to do at least 10 points worth of extra credit.

10 Deliverables

Your entire project should be in a folder called "`firstname_lastname`". This folder should be zipped up and submitted through Canvas. Inside the folder, you should have the followings:

- A folder named "Code" containing all the codes for this assignment. Please include a README file to explain what each file does if you add any other files to the starter code.
- A report in the pdf format. **Make sure you write your name on top of the report.**

Make sure you exclude all the results and original images from your submission.

11 Checklist

Make sure you can check all the items below before submitting your assignment. You will lose 5 points for each item that cannot be checked.

- ☐ The folder is named properly ("`firstname_lastname`"). The folder structure should be exactly as follows

  "`firstname_lastname`.zip"

  - "`firstname_lastname`"
    - "Code"
    - Report.pdf
- ☐ Inside the root folder, there is a folder called "Code" that contains your source code. Also make sure the report is in the root folder.
- ☐ The folders "Images" and "Results" are not included (you only submit your codes and a report).
- ☐ Name written on top of the report.
- ☐ The report is in pdf format. The file size should be under 10 MB.
- ☐ All the results are included in the report.
- ☐ Original images or results are not included in the package.

12 Ruberic

Total credit: [100 points]

[40 points] - Implement the single scale approach
[50 points] - Implement the multi-scale approach
[10 points] - Write up

Extra credit: [up to 10 points]

[04 points] - Automatic cropping
[03 points] - Automatic contrasting
[04 points] - Automatic white balance
[03 points] - Better features
[05 points] - Better transformations

13 Acknowledgments

This project is derived from Alexei A. Efros Computational Photography course with permission.