Fall 2023 CS543/ECE549

Assignment 1: Registering Prokudin-Gorskii color separations of the Russian Empire

Due date: Monday, September 11, 11:59:59 PM


This assignment was adapted from A. Efros by Prof. S. Lazebnik; updated by Daniel McKee; and gently modified by D.A. Forsyth.

Background

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. These are known as **color separations**. 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. 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.

Overview

The goal of this assignment is to learn to work with images 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.

Some details are quite important. You should notice that it matters how you crop the images when you align them -- the separations may not overlap exactly. We have provided an RGB image to check your code on at this location. You should separate this image into three layers (R, G and B), then place each of those layers inside a slightly bigger, all white, layer, at different locations. Now register these three. You can tell whether you have the right answer in two ways: first, you shifted the layers with respect to one another, so you know the right shift to register them; second, if you look at the registered picture, the colors should be pure.

You will need to implement this assignment in Python, and you should familiarize yourself with libraries for scientific computing and image processing including NumPy and PIL.

Data

A zip archive with six input images for the basic alignment experiments is available here. The high-resolution images for multiscale alignment experiments are available in this archive (the file is over 150MB). **Separations:** We have made no effort to determine what order the separations are in, and we believe that this changes from file to file. You should try each of the 6 available options and see which gives the most plausible image. This is relatively easy to do (if you get the order wrong for the example image at the top of the page, you'll find that the doors are an implausible color).

Detailed instructions

Your program should divide the image into three equal parts (channels) and align two of the channels to the third (you should try different orders of aligning the channels and figure out which one works the best). For each input image, you will need to include in your report the colorized output and the (x,y) displacement vectors that were used to align the channels.

Basic alignment. The easiest way to align the parts is to exhaustively search over a window of possible displacements (say [-15,15] pixels independently for the x and y axis), 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. The most basic one is the *L2 norm* of the pixel differences of the two channels, also known as the *sum of squared differences* (SSD), which in Python is simply `sum((image1-image2)**2)` for images loaded as NumPy arrays. Note that in our case, the images to be matched do not actually have the same brightness values (they are different color channels), so a cleverer metric might work better. One such possibility is *normalized cross-correlation (NCC)*, which is simply the dot product between the two images normalized to have zero mean and unit norm. Test your basic alignment solution on the first set of six lower resolution images.

Multiscale alignment. For the high-resolution glass plate scans provided above, exhaustive search over all possible displacements will become prohibitively expensive. To deal with this case, implement a faster search procedure using 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. It is very easy to implement by adding recursive calls to your original single-scale implementation.

For Bonus Points

Implement and test any additional ideas you may have for improving the quality of the colorized images. For example, the borders of the photograph will have strange colors since the three channels won't exactly align. See if you can devise an automatic way of cropping the border to get rid of the bad stuff. One possible idea is that the information in the good parts of the image generally agrees across the color channels, whereas at borders it does not.

If you have other ideas for further speeding up alignment of high-resolution images, you may also implement and test those.

What to turn in

You should turn in both your **code** and a **report** discussing your solution and results. The report should contain the following:

- A brief description of your implemented solution, focusing especially on the more "non-trivial" or interesting parts of the solution. What implementation choices did you make, and how did they affect the quality of the result and the speed of computation? What are some artifacts and/or limitations of your implementation, and what are possible reasons for them?

- For the multiscale solution, report on its improvement in terms of running time (feel free to use an estimate if the single-scale solution takes too long to run). For timing, you can use the python `time` module. For example:

  ```
  import time
  start_time = time.time()
  # your code
  end_time = time.time()
  total_time = end_time - start_time
  ```

- The output color image for every single input glass plate and the displacement vectors that were used to align the channels. Include outputs and displacements for both the six lower resolution images processed with your basic solution and the three high-resolution images aligned with the multiscale solution. When inserting results images into your report, you should resize/compress them appropriately to keep the file size manageable (under 20MB ideally) -- but make sure that the correctness and quality of your output can be clearly and easily judged. **You will not receive credit for any results you have obtained, but failed to include directly in the report PDF file.**

- Any bonus improvements you attempted, with output. **Any parts of the report you are submitting for extra credit should be clearly marked as such.**

Submission Instructions

To submit this assignment, you must upload the following files on **Canvas**:

1. Your code. The filename should be **lastname_firstname_a1.py** (or another Python extension).
2. A brief report **in a single PDF file** with all your results and discussion following this **template**. The filename should be **lastname_firstname_a1.pdf**.
3. All your output images **in a single zip file**. The filename should be **lastname_firstname_a1.zip**. Note that this zip file is for backup documentation only, in case we cannot see the images in your PDF report clearly enough. As stated above, **you will not receive credit for any output images that are part of the zip file but are not shown (in some form) in the report PDF.**

Please refer to course policies on academic honesty, collaboration, late days, etc.