CSCI 3290 Computational Photography

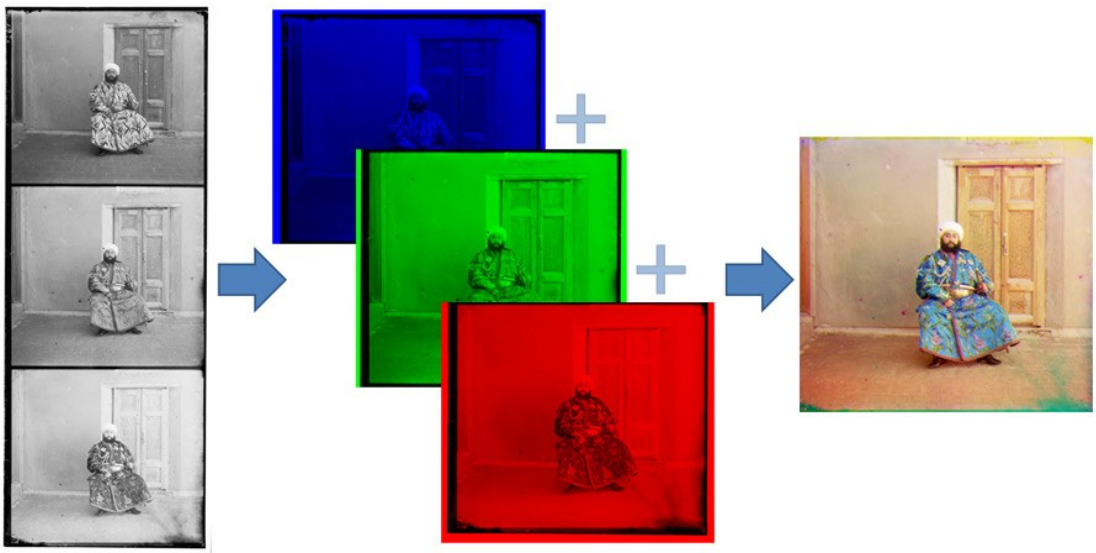
Assignment 1 – Image Alignment

Due Date: 11:59pm on Thursday, October 15th, 2015

I. Objectives

- Understand the channels of color images.
- Learn basic image processing techniques with MATLAB.
- Learn basic measurement metrics for matching images.
- Learn the concept of coarse-to-fine scheme for hierarchically matching images.
- Get familiar with MATLAB Programming (functions, scripts, looping, branching, variables)

II. Backgrounds



Sergei Mikhailovich Prokudin-Gorskii (1863-1944)[1] [Сергей Михайлович Прокудин-Горский, in Russian] 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. And he really photographed everything: people, buildings, landscapes, railroads, and 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**.

III. Details

In this assignment, your task is to take the digitized Prokudin-Gorskii glass plate images as input (one image containing 3 different channels, as shown in the figure, and automatically produce a color image by aligning all channels properly, with as few visual artifacts as possible.

You will need to extract the three color channel images and align them so that they form a single RGB color image. The basic steps are as follows:

1. Extract 3 channels (BGR channels) from the input;
2. Calculate two displacement vectors (x_{BG},y_{BG}), (x_{BR},y_{BR}) for aligning the G channel (second) and R channel (third) to the B channel (first)
3. Shift G channel and R channel based on the displacement vector
4. Combine 3 channels together to compose a color image

Moreover, the high-resolution images are quite large so you will need to have a fast and efficient aligning algorithm (read: **Image Pyramid [2]**). You are required to implement a **single-scale** and **multi-scale** aligning algorithm that searches over a user-specified window of displacements. Also, you are required to try your algorithm on other images from the Prokudin-Gorskii collection (at least **5 different images**).

a) Implementation details

Question 1. How to extract 3 channels from input images? (TODO1 in *asgn1_starter*.m)

Answer: In basic implementations, you can get full mark by simply cropping the input image into 3 even vertical parts. You are encouraged to propose better methods to get extra credit.

Question 2. How to weigh different displacement vector (x, y)? (TODO2 in *alignSingle*.m & TODO3 in *alignMulti*.m)

Answer: The purpose is to make shifted G, R-channel 'similar' to B-channel. Several possible metrics to measure similarity are proposed:

- **Sum of squared differences (SSD)**: sum((image1-image2).^2)
- **Normalized cross correlation**: dot(image1./||image1||, image2./||image2||)

Note that in this particular case, even if the images align perfectly, they do not actually have the same brightness values for corresponding pixels (they are different color channels), so you should **try other metrics** which may work better.

Question 3. How to search for the displacement vector?

Answer: The easiest way to align the parts is to **exhaustively search** over a window of possible displacements (e.g. [-15,15] pixels), score each one using some image matching metric, and take the displacement with the best score. **(TODO2 in *alignSingle*.m)**

However, exhaustive search will become prohibitively expensive if the displacement search range or image resolution are too large (which will be the case for high-resolution glass plate scans). To avoid this, you will need to implement a **coarse-to-fine search strategy** 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 displacement estimate as you go. **DO NOT** use MATLAB's *impyramid* function -- write your own function which blurs an image and then subsamples the pixels. **(TODO3 in *alignMulti*.m)**

b) Important notes

About the images:

- The images are, from top to bottom, in **B-G-R** channel order. (**Attention!**)
- Each image has a high and low resolution image available online, so trying your aligning algorithm on both.
- Assume the negatives are evenly divided into 3 plates (i.e., each plate is in exactly 1/3 height of the negative).
- Assume that a simple x, y translation model is sufficient for proper alignment.
- There are some digitized glass plate images (both hi-res and low-res versions) in the package.
- Besides the test images we provide, you are required to try your algorithms on other images from the Prokudin-Gorskii collection (at least **5 different images**).
- More image data on: **Library of Congress**

Packages we provide:

1. Specification of Assignment 1
2. MATLAB stencil code is available
3. Sample images for testing
4. Links for more data and related resources

IV. Requirements

Your submission should contain the following:

a) Codes including:

- **imgAlignSingle.m & alignSingle.m** – For Single-scale Implementation
- **imgAlignMulti.m & alignMulti.m** – For Multi-scale Implementation
- **Other codes you would like to add**.

Remarks:

- You can use functions provided by MATLAB, its Toolboxes and libraries supplied by TAs (Unless identified elsewhere). However, you still need to describe all the algorithms in your **OWN** words in the report.
- If you use others' functions or codes, please clearly claim all of them in the report, and **UPLOAD** them within your assignment. Otherwise, you might be considered as **PLAGIARISM**.
- You own work **MUST** be the majority of all your uploaded codes.

b) Reports:

- For this assignment, and all other assignments, you must do a report in **HTML**.
- In the report, you will describe your algorithm and any decisions you made to write your algorithm a particular way. Then you will show and discuss the results of your algorithm.
- Discussion on algorithms' efficiency (based on time elapsed) if you want.
- Discuss any extra credit you did if you want.
- Ideas and algorithms from others **MUST** be clearly claimed. List papers or links as **Reference** if needed.
- Feel free to add any other information you feel is relevant.

c) Handing in

The folder you hand in must contain the following:

- **README.docx** - containing anything about the project that you want to tell the TAs, including brief introduction of the usage of the codes
- **code/** - directory containing all your code for this assignment

- **html/** - directory containing all your html report for this assignment (including images). Do not turn in the large .**tiff** images. Your web page should only display compressed images (e.g. **jpg** or **png**).
- **html/index.html** - home page for your results

Compressed the folder into <*your student ID*>**-Asgn1.zip** or <*your student ID*>**-Asgn1.rar**, and upload it to e-Learning system.

d) Due Date:

11:59pm on Thursday, October 15th, 2015

V. Scoring & Extra Bonus

a) Rubric

- ✧ +55 pts: Single-scale implementation with successful results on low-res images
- ✧ +25 pts: Multi-scale image pyramid implementation with successful results on high-res images
- ✧ +20 pts: Report
- ✧ +20 pts: Extra credit (up to 20 points)
- ✧ -5*n pts: Lose 5 points for every time (after the first) you do not follow the instructions for the hand in format

b) Extra Credit

Although the color images resulting from this automatic procedure will often look strikingly real, they are still not nearly as good as the manually restored versions available on the LoC website and from other professional photographers.

However, each photograph takes days of painstaking Photoshop work, adjusting the color levels, removing the blemishes, adding contrast, etc.

Can you come up with ways to address these problems automatically? Feel free to come up with your own approaches or talk to the Professor or TAs about your ideas. There is no right answers here, just try out things and see what works.

Here are some ideas:

- Up to 8 pts: 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.
- Up to 6 pts: 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.

- Up to 6 pts: Better color mapping. There is no reason to assume (as we have) that the red, green, and blue lenses used by Produkin-Gorskii correspond directly to the R, G, and B channels in RGB color space. Try to find a mapping that produces more realistic colors.
- Up to 6 pts: Better features. Instead of aligning based on RGB similarity, try using gradients or edges.
- Up to 10 pts: Better transformations. Instead of searching for only 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.
- Up to 8 pts: Aligning and processing data from other sources. In many domains, such as astronomy, image data is still captured one channel at a time. Often the channels don't correspond to visible light, but NASA artists stack these channels together to create false color images. For example, **here is a tutorial** on how to process Hubble Space Telescope imagery yourself. Also, consider images like **this one of a coronal mass ejection** built by combining **ultraviolet images** from the Solar Dynamics Observatory. To get full credit for this, you need to demonstrate that your algorithm found a non-trivial alignment and color correction.

For all extra credit, be sure to demonstrate on your web page cases where your extra credit has improved image quality.

The upper bound score of this assignment is 100 points

VI. Hints and Advice

- For all projects, don't 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). Don't worry if one or two of the handout images don't align properly using the simpler metrics suggested here.
- The input images can be in jpg (uint8) or tiff format (uint16), remember to convert all the formats to the same scale (see im2double and im2uint8).
- Shifting a matrix is easy in MATLAB by using circshift. Filtering images using **imfilter.**
- The **borders** of the images will probably hurt your results, try computing your metric on the **internal pixels** only.
- Output all of your images to **jpg**, it'll save you a lot of disk space.

VII. Other Remarks

- **Five** late days total, to be spent wisely
- 20% off from each extra late day
- The three assignments are to be completed **INDIVIDUALLY** on your own.
- You are encouraged to use methods in related academic papers.

- Absolutely **NO sharing or copying** code! **NO sharing or copying** reports! Offender will be given a failure grade and the case will be reported to the faculty.

VIII. Reference

[1] http://en.wikipedia.org/wiki/Prokudin-Gorskii

[2] http://docs.opencv.org/doc/tutorials/imgproc/pyramids/pyramids.html