CSCI 3290 Computational Photography

Assignment 1 – Image Alignment

Due Date: **11:59pm on Monday, October 7th, 2013**

1. Backgrounds

Sergei Mikhailovich Prokudin-Gorskii (1863-1944) was a photographer ahead of his time. He saw color photography as the wave of the future and came up with a simple idea to produce color photos: record three exposures of every scene onto a glass plate using a red, a green, and a blue filter and then project the monochrome pictures with correctly coloured light to reproduce the color image; color printing of photos was very difficult at the time. Due to the fame he got from his color photos, including the only color portrait of Leo Tolstoy (a famous Russian author), he won the Tzar's permission and funding to travel across the Russian Empire and document it in 'color' photographs. His RGB glass plate negatives were purchased in 1948 by the Library of Congress. They are now digitized and available on-line.

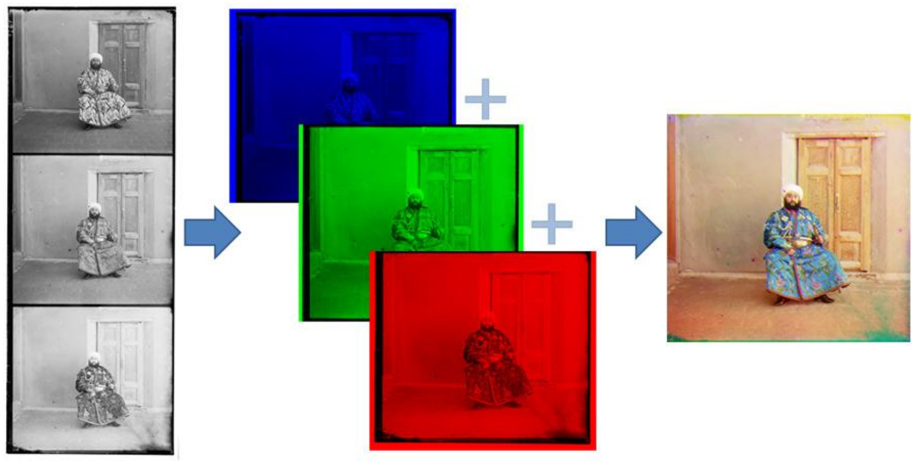

Figure 1: **Left**: input image; **Middle**: extracted 3 channels; **Right**: output image.

2. Problem

Your task in this assignment is to take one image containing 3 image channels as input (as shown in the figure on the right), and output a color image by aligning all channels together. The basic steps of alignment 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 and R channel to the B channel **(this step is the main task you need to do)**
3. Shift G channel and R channel based on the displacement vector
4. Combine 3 channels together to compose a color image

3. Implementation Details

Question 1. How to extract 3 channels from the input image?

Answer: In the basic requirement, you can get full mark by uniformly cropping the input image into 3 parts vertically (it is already implemented in our skeleton code). You can improve this step to get extra credit.

Question 2. How to measure whether a displacement vector (x,y) is correct or not?

Answer: To make your alignment results contains as few visual artifacts as possible, you need to have a way to measure how accurate the alignment are, i.e. how accurate the displacement vector (x,y) is. After shifting one image to match another based on the displacement vector, there are several possible metrics to measure how well the images match:

1. **Sum of squared differences (SSD)**: sum ((image1-image2).^2)
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 (because they are different color channels), so **you can try if there is other metrics that work better.**

Question 3. How to search for the displacement vector?

Answer: The easiest way to do aligning 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. However, exhaustive search will become prohibitively expensive if the displacement search range or image resolution is 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 that blurs an image and then subsamples the pixels.

4. TODO List

We provide you the following files as the skeleton code (/CSCI3290_Assignment1/StarterCode/): **imgAlignSingle.m**, **imgAlignMulti.m**, **alignSingle.m**, **alignMulti.m**, **extractChannels.m**

You need to add your own code to complete the program:

TODO #1. Exhaustively search for the best displacement vector. (See **alignSingle.m**)

TODO #2. Build two image pyramids. (See **alignMulti.m**)

TODO #3. Search for the best displacement vector in the coarse-to-fine scheme. (See **alignMulti.m**)

We also provide you the testing images for you to test your program (/CSCI3290_Assignment1/data/). You can also find more image data on: **Library of Congress**

5. Hints

1. The images are, from top to bottom, in **B-G-R** order.
2. Each image has a high-resolution and low-resolution image available online, so trying your aligning algorithm on both.
3. Besides the test image we provide, you are required to try your algorithm on other images from the Prokudin-Gorskii collection (at least **5 different images**).
4. Don't get your coordinate order mixed up – MATLAB matrices are accessed (y, x).
5. 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).
6. Shifting a matrix is easy to do in MATLAB by using **circshift**. Filtering images using **imfilter.**
7. The **borders** of the images will probably hurt your results, try computing the metric on the **internal pixels** only.

6. Marking

The upper bound score of this assignment is 120.

Basic Part (100%)

(55%) Single-scale implementation with successful results on low-res images. (TODO #1)

(25%) Multi-scale image pyramid implementation with successful results on high-res images. (TODO #2/3)

(20%) Report

Extra Credit (20%)

The upper bound of extra credit is 20 percent. Be sure to demonstrate on your web page cases where your extra credit has improved image quality. You can consider the following ideas:

Bonus 1 (8%): 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.

Bonus 2 (6%): 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.

Bonus 3 (6%): 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.

Bonus 4 (6%): Better features. Instead of aligning based on RGB similarity, try using gradients or edges.

Bonus 5 (10%): 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.

Bonus 6 (8%): 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.

7. Submission

Your submission should contain two things: **code files (in MATLAB)** and **a report (in HTML)**

MATLAB Code

1. The completed code files: **imgAlignSingle.m**, **imgAlignMulti.m**, **alignSingle.m**, **alignMulti.m**, **extractChannels.m** (PLEASE submit all files even if you have no change on some of them)
2. Any other code you would like to add.

Report Requirements

1. The report must be written in **HTML**.
2. In the report, you should describe your algorithm and any decisions you made to write your algorithm a particular way. You should also show and discuss the results of your algorithm.
3. Discussion on algorithms' efficiency (based on time elapsed).

4. Discuss any extra credit you did.
5. Ideas and algorithms from others **MUST** be clearly claimed. List papers or links in the **Reference** section if needed.
6. Feel free to add any other information you feel is relevant.

Submission Format

The folder you hand in must contain the following:

1. **README.txt -** containing anything about the project that you want to tell the TAs, including brief introduction of the usage of the codes
2. **code/** - directory containing all your code for this assignment.
3. **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).
4. **html/index.html** - home page for your results

Please compressed the folder into <*your student ID*>-asgn1**.zip** or <*your student ID*>-asgn1**.rar**, and upload it to e-Learning system.

8. Remarks

1. Five late days total, to be spent wisely.
2. 20% off per day for late submission.
3. Your mark will be deducted if you do not follow the instructions for the submission format.
4. The three assignments are to be completed **INDIVIDUALLY** on your own.
5. You are encouraged to use methods in related academic papers.
6. Absolutely **NO sharing or copying** of code! **NO sharing or copying** of reports! Offender will be given a failure grade and the case will be reported to the faculty.

Reference

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

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