University at Buffalo, Computer Science and Engineering

CSE 468/568: Robot Algorithms

Lab 3: Colorizing the Prokudin-Gorskii photo collection*

In this assignment, you will be implementing some basic image processing algorithms to detect features to frame alignment. To simplify this portion, we will implement these algorithms in OCTAVE. Octave is an open-source version of Matlab, and can faithfully run most Matlab scripts.

Octave Installation and Tutorials

In your VMWare image, run the following two commands
`$ sudo apt-get update`
`$ sudo apt-get install octave octave-image`
This should install Octave on your machine.
MATLAB novices can get started using the tutorial from here.
`http://cseweb.ucsd.edu/~sjb/classes/matlab/matlab.intro.html`
Octave novices can get started using the tutorial from here.
`https://en.wikibooks.org/wiki/Octave_Programming_Tutorial`
I would recommend you learn and understand the following:

- Programming basics in Matlab and some examples
- Matrices and arrays in Matlab
- Basic image processing methods in Matlab
- Specific commands to look up in addition to the tutorials above include `normxcorr2, cat, im2double, circshift`

Here's another tutorial (deck of slides) that should get you started quickly with this assignment.

*1We adapted this problem from Radhakrishna Dasari with permission, though originally designed by Aloysha Efros at CMU

Assignment Overview

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 on- line at http://www.loc.gov/exhibits/empire/gorskii.html.
The goal of this assignment is to learn to work with images in Matlab/Octave 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 (img*.jpg) that you should run your algorithm on.
Your program should take 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 also need to print the (x,y) displacement vector 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 each for each color channel (R, G and B). Your first task is to take each of the six images, align the three plate images as three color channel images and save the resultant color image. These images will look blurred as they are not aligned correctly. This is ok. Save all six images with the names `image*-color.jpg` i.e., save `image1.jpg` as `image1-color.jpg` and so on.
Create a file `main.m` that iterates through each image, creates a color image and can write the color image into a file.

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), 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 simplest one is just the L2 norm also known as the Sum of Squared Differences (SSD) distance which is simply sum(sum((image1-image2)^2)). Another is normalized cross-correlation (NCC), which is simply a dot product between two normalized vectors: (image1./|image1| and image2./|image2|). See the Matlab/Octave function `normxcorr2`.
Write two methods - `im_align1.m` and `im_align2.m`. `im_align1.m` should find the alignment using SSD described above. `im_align2.m` should find the alignment using the normalized cross-correlation. You should call both of them from the main file (`main.m`) and print out the calculated alignments. Finally, you should create a color image for each of the alignments and write them as files. Name the SSD aligned image as `image*-ssd.jpg` and name the NCC-aligned image as `image*-ncc.jpg`.
Some tips:

- Note that the filter order from top to bottom is BGR, not RGB!
- Your task is to align the second and third channels (G and R) with the B channel
- Depending on how you shift the images, the border pixels will mess with the SSD calculation, giving false optima. Test different methods for shifting and/or drop the border pixels in the SSD calculation.

Feature-based Alignment

A third way to perform alignment is to perform feature detection and align the images based on the best fit of features. For this task, you will use corners as the feature detector. Create a file `harris.m` that takes an image, computes the cornerness function on an image, and chooses the top 200 features in the given image. Create another file called `im_align3.m` that runs the RANSAC algorithm. The algorithm randomly picks a feature in image 1 (lets say the B channel image) and assumes it aligns with a random feature in image 2 (lets say, the G channel image). Calculate the pixel shift for this alignment. Then apply the same pixel shift to every feature in image 1, and search for a corresponding feature in image 2 within a threshold (a small window). If you find a feature within that window, you can count that as an inlier; else you it is not. Run this several times, and pick the best alignment (highest number of inliers). Output this alignment, along with an aligned color image. Label the color image `image*-corner.jpg`.

Each alignment method will be worth 5% for a total of 15% of the grade.

Report

Finally, you'll need to write up a report (as a PDF document). You should briefly describe your method for alignment for the three parts. After this, please mention the alignment shift that you calculated. This report should be labeled `lab3.pdf`.

Submission

You will submit the following files
`main.m`
`im_align1.m`
`im_align2.m`
`im_align3.m`
`image1-color.jpg, image2-color.jpg, image3-color.jpg,`

`image4-color.jpg, image5-color.jpg, image6-color.jpg`
`image1-ssd.jpg, image2-ssd.jpg, image3-ssd.jpg,`
`image4-ssd.jpg, image5-ssd.jpg, image6-ssd.jpg`
`image1-ncc.jpg, image2-ncc.jpg, image3-ncc.jpg,`
`image4-ncc.jpg, image5-ncc.jpg, image6-ncc.jpg`
`lab3.docx` .
Submit all the files as `lab3.zip` using Autograder as before. We will likely test on other images as well. The assignment is due on Nov 20, 2021 before 11:59:59 PM ET.