Homework 1

CMPSCI 370 Spring 2018, UMass Amherst
Due: February 15, 11:59 PM
Instructor: Subhransu Maji
TA: Matheus Gadelha

Guidelines

Submission. Submit a *zip file* via moodle that includes a pdf with your solutions and source code. You have a total of 3 late days for all your homework assignments. Delay by even one minute counts as a full late day. Submissions beyond the late days will not be given *any* credit.

Plagiarism. We might reuse problem set questions from previous years, covered by papers and webpages, we expect the students not to copy, refer to, or look at the solutions in preparing their answers. We expect students to want to learn and not google for answers. Any instance of cheating will lead to zero credit for the homework, and possibly a failure grade for the entire course.

Collaboration. The homework must be done individually, except where otherwise noted in the assignments. 'Individually' means each student must hand in their own answers, and each student must write their own code in the programming part of the assignment. It is acceptable, however, for students to collaborate in figuring out answers and helping each other solve the problems. We will be assuming that you will be taking the responsibility to make sure you personally understand the solution to any work arising from such a collaboration.

Using other programming languages. We made the starter code available in Python and Matlab. You are free to use other languages such as Octave or Julia with the caveat that we won't be able to answer or debug non Matlab/Python questions.

Python requirements. We will be using Python 2.7. The Python starter code requires `scipy` for loading images and `numpy` for matrix operations (at least v1.12). If you are not familiar with installing those libraries through some package manager (like `pip`), the easiest way of using them is installing Anaconda.

1 Image formation [10 points]

Assume you are making a pinhole camera with a shoebox of size 20cm x 20cm x 30cm. You make a hole on the 20cm x 20 cm side and an image is formed on the opposite side. Assume that an 1 meter tall object is at a distance of 15 meters from the pinhole along the projection axis.

1. *[6 points]* Draw a picture illustrating the object and the image formed in the pinhole camera. Clearly show the marked dimensions.

2. *[2 points]* What is the size of the object in the pinhole image?

3. *[2 points]* At what distance does the image of the object *entirely* occupy the projection screen (*note:* the size of the screen is 20cm x 20 cm).

2 Aligning Prokudin-Gorskii images [25 points]

Sergei Mikhailovich Prokudin-Gorskii was a photographer far ahead of his time. He undertook a photographic survey of the Russian Empire for Tsar Nicholas II. Even though color photography was not yet invented he had the brilliant idea to capture color pictures by simply taking three pictures of each scene, each with a red, green and blue color filter. There was no way of printing these back in the day, so he envisioned complex display devices to show these. However these were never made, but his pictures survived. In this homework you will reconstruct a color image from his photographs as seen in Figure 1.

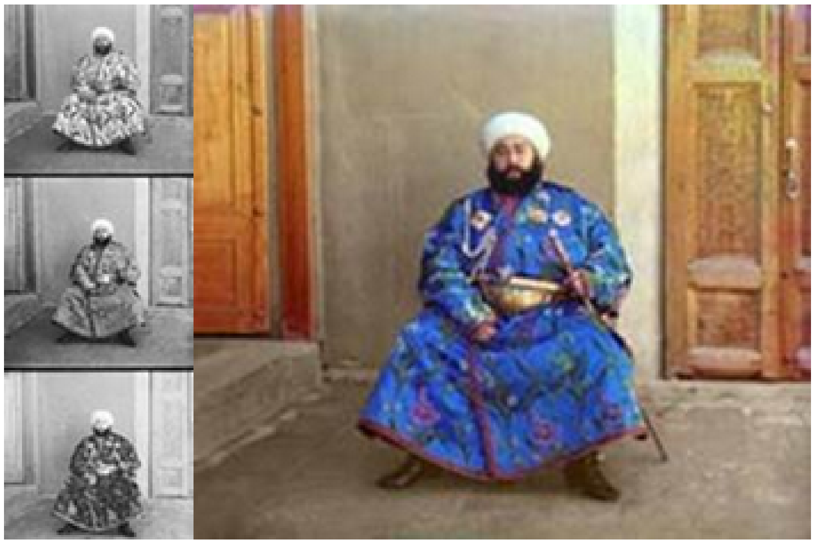

Figure 1: Example image from the Prokudin-Gorskii collection. On the left are the three images captured individually. On the right is a reconstructed color photograph. Note the colors are in B, G, R order from the top to bottom (and not R, G, B)!

The key step to compute the color image is to align the three photographs. Alas, since the camera moved between each shot, these channels are slightly displaced from each other. The simplest way is to keep one channel fixed say R, and align the G, B channels to it by searching over displacements in the range [-15, 15] both horizontally and vertically and pick the displacement that maximizes similarity between the channels. One such measure is the angle between vectors representing the pixels in each channel.

2.a Code

Download the `p1.tar.gz` from the moodle page. Move them to your homework directory and extract the files (e.g. by typing `tar -xvf p1.tar.gz`). The code, data, and latex source files are in the `p1/code`, `p1/data`, and `p1/latex` folders respectively.

Before you start aligning the Prokudin-Gorskii images (in `data/prokudin-gorskii`), you will test your code on synthetic images which have been randomly shifted. Your code should correctly discover the inverse of the shift.

Run `evalAlignment` inside the code directory. This should produce the following output. Note the actual 'gt shift' might be different since it is randomly generated.

```
    Evaluating alignment ..
    1 balloon.jpeg
   gt shift: ( 1,11) ( 4, 5)
 pred shift: ( 0, 0) ( 0, 0)
    2 cat.jpg
   gt shift: (-5, 5) (-6,-2)
 pred shift: ( 0, 0) ( 0, 0)
```

```
      ...
```

The code loads a set of images, randomly shifts the color channels and provides them as input to `alignChannels`. Your goal is to implement this function. A correct implementation should obtain the shifts that is the negative of the ground-truth shifts, i.e. the following output:

```
    Evaluating alignment ..
    1 balloon.jpeg
   gt shift: ( 13, 11)  ( 4, 5)
 pred shift: (-13,-11)  (-4,-5)
    2 cat.jpg
   gt shift: (-5, 5) (-6,-2)
 pred shift: ( 5,-5) ( 6, 2)
      ...
```

Once you are done with that, run `alignProkudin`. This will call your function to align images from the Prokudin-Gorskii collection. The output is saved to the `outDir`. Note: if this directory does not exist, you will have to create it first. In your report, show all the aligned images as well as the shifts that were computed by your algorithm.

Tips: Look at functions `circshift()` and `padarray()` to deal with shifting images in Matlab. In Python, look for `np.roll` and `np.pad`.

2.b What to submit?

To get full credit for this part you have to

- include your implementation of `alignChannels`,
- verify that the `evalAlignment` correctly recovers the color image and shifts for the toy example,
- include the aligned color images from the output of `alignProkudin`, *including* the computed shift vectors for each image.

2.c Extra credit

Here are some ideas for extra credit.

- Shifting images can cause ugly boundary artifacts. Come up with of a way of avoiding this.
- Searching over displacements can be slow. Think of a way of aligning them faster in a coarse-to-fine manner. For example, you may align the channels by resizing them to half the size and then refining the estimate. This can be done by multiple calls to your `alignChannels()` function.

If you do these, or any other, for extra credit make sure to include the code, description and any other details that you think are relevant for grading such as figures, run-time analysis, etc.

3 Color image demosaicing [30 points]

Recall that in digital cameras the red, blue, and green sensors are interlaced in the Bayer pattern (Figure 2). The missing values are interpolated to obtain a full color image. For this part you will implement a simple interpolation algorithm called the *nearest-neighbor* interpolation. The input to the algorithm is a single image im, a NxM array of numbers between 0 and 1. These are measurements in the format shown in Figure 2, i.e., top left `im(1,1)` is red, `im(1,2)` is green, `im(2,1)` is green, `im(2,2)` is blue, etc. Your goal is to create a single color image C from these measurements. Note that in Python/NumPy, those coordiantes are expressed as `im[0,0]`, `im[0,1]`, `im[1,0]`, `im[1,1]`.

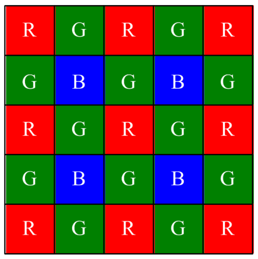

Figure 2: Bayer pattern

Your entry point for this part of the homework in `evalDemosaic`. The code loads images from the data directory (in `data/demosaic`), artificially mosaics them (`mosaicImage` file), and provides them as input to the demosaicing algorithm (`demosaicImage` file). By comparing the result with the input we can also compute the reconstruction error measured as the distance between the reconstructed image and the true image. This is what the algorithm reports. Once you have implemented the `mosaicImage` function run the `evalDemosaic` and you should see the output below.

Right now only the `demosaicImage(im, 'baseline')` is implemented which simply replaces all the missing values for a channel with the average value of that channel. You will implement the option `demosaicImage(im, 'nn')` which hopefully will obtain a significantly lower error and visually pleasing results. Right now it simply calls the baseline algorithm hence the two methods produce identical results.

```
--------------------------------------------
#   image            baseline   nn
--------------------------------------------
1   balloon.jpeg     0.179239   0.179239
2   cat.jpg          0.099966   0.099966
3   ip.jpg           0.231587   0.231587
4   puppy.jpg        0.094093   0.094093
5   squirrel.jpg     0.121964   0.121964
6   pencils.jpg      0.183101   0.183101
7   house.png        0.117667   0.117667
8   light.png        0.097868   0.097868
9   sails.png        0.074946   0.074946
10  tree.jpeg        0.167812   0.167812
--------------------------------------------
    average          0.136824   0.136824
--------------------------------------------
```

You will implement the two functions:

1. *[10 points]* Implement `mosaicImage(im)`. This function takes an image `im` with three color channels and returns an output `mosim` which has a single channel where the red, green, and blue pixels are copied according to the Bayer pattern shown in the figure.

2. *[20 points]* Implement `demosaicImage(im, 'nn')`. This function takes the demosaiced image and reconstructs the three color channels. The 'nn' option stands for *nearest-neighbor* interpolation, i.e. you simply copy the value of the nearest available pixel. For example, each missing green pixel can copy the value of the green pixel to its left (note that this doesn't work on the left boundary where you can copy from the top or bottom). This very simple method should produce reasonable results and substantially lower errors (around 0.025 average error) when you run `evalDemosaic`

3.a What to submit?

To get full credit for this part you have to

- include your implementation of `mosaicImage` and `demosaicImage`,
- include the output of `evalDemosaic`.

3.b Extra credit

There are numerous other approaches for interpolation which you can read on the internet. You can use the ideas but do not copy the code (we can easily detect if you have simply copied the code from the web). You can implement this as other options `demosaicImage(im, 'yourmethod')` and include the list of options in the `evalDemosaic` code. It is possible to get errors as low as 0.017 without too much effort.