Homework 1

CMPSCI 370 Spring 2019, UMass Amherst
Instructor: Subhransu Maji
TA: Tsung-Yu Lin

Guidelines

Submission. Submit your answers via Gradescope that includes a pdf with your solutions and source code. You have a total of 3 late days for all your homework assignments which you can use in any way you like. However note that delay by even one minute counts as a full late day and submissions beyond the late days will not be given *any* credit. Please contact the instructor and obtain permission ahead of time if you seek exceptions to the policy.

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.

Matlab requirements. The code should work any Matlab version ($\geq$ 2011a) and relies on the Image Processing Toolbox for some functions.

Python requirements. We will be using Python 2.7. The Python starter code requires `scipy`, `matplotlib` and `pillow` 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. Contact the course staff if you are having trouble installing these.

Using other programming languages. We made the starter code in Python and Matlab. You are free to use other languages such as Octave or Julia with the caveat that these will not be supported by the course staff.

Linear algebra review [0 points]

We will review some basic concepts in linear algebra relevant to this homework. Linear algebra is the math of vectors and matrices. Let n be a positive integer and let $\mathbb{R}$ denote the set of real numbers, then $\mathbb{R}^n$ is the set of all n-tuples (an ordered list of size n) of real numbers. A vector $\vec{v} \in \mathbb{R}^n$ is an n-tuple of real numbers. For example, consider a vector that has two components: $\vec{v} = (v_1, v_2) \in (\mathbb{R}, \mathbb{R}) \in \mathbb{R}^2$.

A matrix $A \in \mathbb{R}^{m \times n}$ is a rectangular array of real numbers with m rows and n columns. For example, a 2×3 matrix looks like this:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{11} & a_{12} & a_{13} \end{bmatrix} \in \begin{bmatrix} \mathbb{R} & \mathbb{R} & \mathbb{R} \\ \mathbb{R} & \mathbb{R} & \mathbb{R} \end{bmatrix} \in \mathbb{R}^{2\times 3}$$

0.a Vector operations

Given vectors $\vec{u} = (u_1, u_2, \ldots, u_n)$ and $\vec{v} = (v_1, v_2, \ldots, v_n) \in \mathbb{R}^n$ we can define various vector operations:

- Addition: $\vec{u} + \vec{v} = (u_1 + v_1, u_2 + v_2, \ldots, u_n + v_n)$
- Subtraction: $\vec{u} - \vec{v} = (u_1 - v_1, u_2 - v_2, \ldots, u_n - v_n)$
- Dot product: $\vec{u} \cdot \vec{v} = u_1 v_1 + u_2 v_2 + \ldots + u_n v_n$
- Length of a vector: $||\vec{u}|| = \sqrt{u_1^2 + u_2^2 + \ldots + u_n^2}$. The $||.||$ notation is also called the "norm".
- Distance between two vectors: $d(\vec{u}, \vec{v}) = ||\vec{u} - \vec{v}|| = \sqrt{(u_1 - v_1)^2 + (u_2 - v_2)^2 + \ldots + (u_n - v_n)^2}$
- Relationship between dot product and distance: $||\vec{u} - \vec{v}||^2 = ||\vec{u}||^2 + ||\vec{v}||^2 - 2\vec{u} \cdot \vec{v}$. You can verify this using the definitions above and the identity $(a - b)^2 = a^2 + b^2 - 2ab$.

Assume that $\vec{u}, \vec{v}$ are of unit length, i.e. $||\vec{u}|| = ||\vec{v}|| = 1$. The relationship between the dot product and distance can be simplified in this case to $||\vec{u} - \vec{v}||^2 = 1 + 1 - 2\vec{u} \cdot \vec{v} = 2(1 - \vec{u} \cdot \vec{v})$. Thus if the distance between the two vectors is small the dot product is close to 1, and if the distance is large then the dot product is close to zero. Hence, the dot product can be seen as a *similarity measure*. In fact the dot product normalized by the norm of the vectors measures the cosine of the angle θ between them, i.e., $\cos\theta = \frac{\vec{u}\cdot\vec{v}}{||\vec{u}||||\vec{v}||}$.

0.b Matrix operations

Given two matrices A, B denote a_{ij} and b_{ij} as the element in the i^{th} row and j^{th} column of matrix A and B respectively. We can define the following matrix operations:

- Addition: $C = A + B$ with $c_{ij} = a_{ij} + b_{ij}$.
- Subtraction: $C = A - B$ with $c_{ij} = a_{ij} - b_{ij}$. Note that for addition and subtraction both A and B have to be of the same dimensions.
- Matrix multiplication: $C = AB$ with $c_{ij} = \sum_k a_{ik} b_{kj}$. Thus, c_{ij} is the dot product of the i^{th} row of A with the j^{th} column of B. Thus in order to be compatible the number of columns of A must match the number of rows of B. If $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n \times p}$ the $C = AB \in \mathbb{R}^{m \times p}$. Here is an example:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \\ a_{31}b_{11} + a_{32}b_{21} & a_{31}b_{12} + a_{32}b_{22} \end{bmatrix}$$

- Matrix transpose of A (denoted by A^T) is obtained by transposing the rows and columns:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}^T = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \end{bmatrix}$$

1 Matrix manipulation [8 points]

The following will test your basic matrix manipulation skills in Matlab or Python. Note that if you are using a different programming language, e.g. C or Octave, write down the equivalent code to do this in that language.

1. *[1 point]* Create a $m \times n$ array of all zeros (double precision)
2. *[1 point]* Create a $m \times n$ array with each entry uniformly distributed in the interval $[0, 1]$, i.e. each value between 0 and 1 is equally likely (double precision)
3. *[2 points]* You can represent vectors as matrices of size $n \times 1$. Given a vector represented as a variable v, write down the code to compute its length (or norm). Avoid using for loops if possible. You may find the functions `sum()` and element-wise squaring `.^2` useful for this.
4. Given variables u and v representing arrays of size $n \times 1$, write down code to compute their
   (a) *[1 point]* dot product
   (b) *[1 point]* angle
   (c) *[1 point]* distance
5. *[1 point]* Given an array $a \in \mathbb{R}^{m\times n}$ write code to reshape it to a vector of size $nm \times 1$.

2 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. *[4 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).
4. *[2 points]* Suppose it took 10 milliseconds to capture an image using a pinhole camera with aperture radius of 1mm. How long would it take to capture the same amount of light using a camera with a lens of radius 10mm?

3 Aligning Prokudin-Gorskii images [25 points]

Sergei Mikhailovich Prokudin-Gorskii was a Russian chemist and photographer ahead of his time. Even though color photography was not yet invented, he had the brilliant idea to capture color pictures by simply taking pictures of a scene, with a red, green and blue 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 during his lifetime, but his pictures survived. In this homework you will reconstruct a color image taken from a collection of images from a photographic survey of the Russian Empire he conducted for Tsar Nicholas II. Figure 1 illustrates the idea.

Recall that color images are represented digitally as RGB color channels. Thus if we stack the three channels in the right order then the color image will be displayed. Alas, since the three pictures weren't taken from the same position due to the camera motion in between shots, the images are likely to be misaligned. The key to obtain a crisp color image is to align the three photographs. For this homework you will assume that the motion is a pure translation.

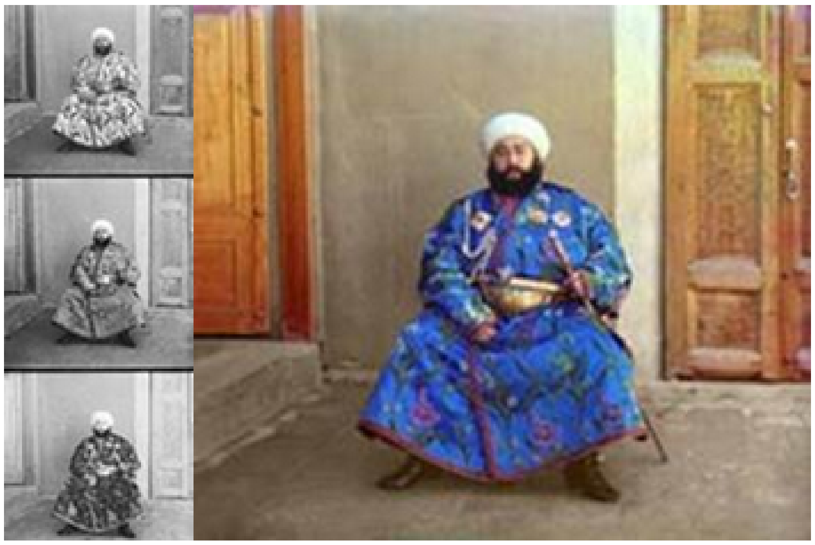

Figure 1: Example image from the Prokudin-Gorskii collection. On the left are the three images captured individually. On the right is a reconstructed color photograph. Note the colors are in B, G, R order from the top to bottom (and not R, G, B)!

Your strategy for alignment will be to search over a range of displacements in X and Y directions for each channel. 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 of pixels in each channel. Recall that the angle between two vectors a and b given by

$$\cos\theta = \frac{a^T b}{||a|| \cdot ||b||},$$

where $||a|| = \sqrt{a^T a}$ denotes the norm (or length) of a vector.

3.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 randomly shifted synthetic images. 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 that 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`.

3.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.

3.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.

4 Color image demosaicing [35 points]

Recall that in digital cameras the red, blue, and green sensors are interlaced in the Bayer pattern (Figure 2) and missing values are interpolated to obtain a full color image. For this part you will implement the *nearest-neighbor* interpolation algorithm. 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 coordinates are expressed as `im[0,0]`, `im[0,1]`, `im[1,0]`, `im[1,1]`.

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 measure an error computed as the distance between the reconstructed image and the true image in pixel space. This is what the algorithm reports. Once you have implemented the `mosaicImage` function run the `evalDemosaic` and you should see the output below.

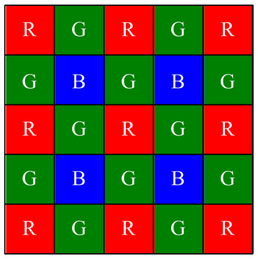

Figure 2: Bayer pattern

Only the `demosaicImage(im, 'baseline')` is implemented in the provided code. The baseline simply replaces all the missing values for a channel with the average value of that channel. Your goal is to implement the nearest-neighbor interpolation approach (`demosaicImage(im, 'nn')`), which should obtain significantly lower errors. 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 `mosim` that has a single channel where the red, green, and blue pixels are copied according to the Bayer pattern shown in the figure. This function simulates the sensor in the digital camera.

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. *[5 points]* The nearest neighbor interpolation relies on smoothness of natural images, i.e., nearly pixels are roughly of the same value. Create a plot where the value of a pixel for the red channel is plotted against the value of the pixel to its right for the red channel. Similarly create plots for the green and blue channels. Use the image `data/demosaic/puppy.jpg` for this plot. What do you observe?

4.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`.
- include the three plots for the `puppy.jpg` image.

4.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.

5 Submission and rubric

- Follow the outline on Gradescope for your submission. This will make it easier for us to grade different parts of your solutions.
- A fraction of the points for each coding assignment will be for readability and efficiency of your code. Thus it is important to properly document parts of the code you implemented.