EECS 442 Computer Vision: Homework 1

Instructions

- This homework is **due at 11:59:59 p.m. on Monday September 30th, 2019**.
- The submission includes two parts:
  1. **To Gradescope:** a `pdf` file as your write-up, including your plots and answers to all the questions and key choices you made.
     The write-up must be an electronic version. **No handwriting, including plotting questions.** LaTeX is recommended but not mandatory.
     You might like to combine several files to make a submission. Here is an example online link for combining multiple PDF files: https://combinepdf.com/.
  2. **To Canvas:** a `zip` file including all your code, and files specified in questions with **Submit**, all under the same directory. You can submit Python code in either `.py` or `.ipynb` format.
- Remember to format your submission as indicated at https://web.eecs.umich.edu/~fouhey/teaching/EECS442_F19/format.html and check the format with `check442.py` before you submit.

Python Environment

We are using Python 3.7 for this course. You can find references for the Python standard library here: https://docs.python.org/3.7/library/index.html. To make your life easier, we **recommend** you to install Anaconda 5.2 for Python 3.7.x (https://www.anaconda.com/download/). This is a Python package manager that includes most of the modules you need for this course.

We will make use of the following packages extensively in this course:

- Numpy (https://docs.scipy.org/doc/numpy-dev/user/quickstart.html)
- OpenCV (https://opencv.org/)
- Pytorch (https://pytorch.org/)
- Matplotlib (http://matplotlib.org/users/pyplot_tutorial.html)

We will use the following packages for this homework and refer to some of them as:

- `numpy as np`
- `matplotlib.pyplot as plt`
- `imageio` (For `gif` generation only)

1 Camera projection Matrix [30 pts]

Study the **Projection and Dolly Zoom** notebook released on course website and finish the following tasks.

(a) Write a function `rotY()` which takes an angle `theta` (in radian) and outputs the 3D rotation matrix of rotating by `theta` about the y-axis (right-hand rule). You may refer to this Wikipedia entry: https://en.wikipedia.org/wiki/Rotation_matrix#Basic_rotations After you are done, refer to the starter code to generate and **submit** `cube.gif` of a cube rotating around itself. (5 pts)

(b) Similarly, write a function `rotX()` which rotates about the x-axis. Let $\theta = \pi/4$, consider the following two transformations:

  (a) `rotX(theta)`, followed by `rotY(theta)`

  (b) `rotY(theta)`, followed by `rotX(theta)`

  Using `renderCube()` in the same way, plot the resulting view of the cube from two transformations. Are 3D rotation matrices commutative? (5 pts)

(c) Combine `rotX()` and `rotY()`, choose an appropriate order and a pair of parameters so that `renderCube()` draws a projection of the cube where one diagonal of the cube is projected to a single point, as shown in Figure 1 (left). Report the order and parameters you choose. (10 pts)

  Hint: The diagonal of the cube rotates together with the cube. When it projects to a single point in 2D, it is horizontal and perpendicular to the camera plane in 3D. You can either make a mathematical calculation or perform a numerical search.

(d) Implement an orthographic camera by either adding a branch to function `projectLines()`, or refer to it and write a new one. Then plot the same rotated cube in the previous part with this orthographic camera. It should look like Figure 1 (right). (10 pts)

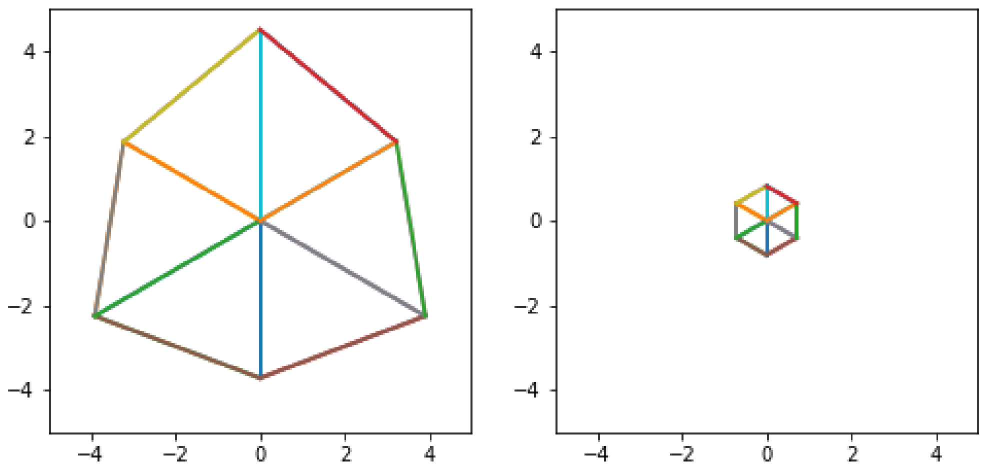

Figure 1: The diagonal of a cube projected to a single point

2 Prokudin-Gorskii: Color from grayscale photographs [50 pts]

In this part, you are tasked with implementing the dream of Russian photographer, Sergei Mikhailovich Prokudin-Gorskii (1863-1944), via a project invented by Russian-American vision researcher, Alexei A. Efros (1975-present)[1]. Sergei was a visionary who believed in a future with color photography (which we now take for granted). During his lifetime, he traveled across the Russian Empire taking photographs through custom color filters at the whim of the czar. To capture color for the photographers of the future, he decided to take three separate black-and-white pictures of every scene, each with a red, green, or blue color filter in front of the camera. His hope was that you, as a student in the future, would come along and produce beautiful color images by combining his 3 separate, filtered images.

Task 1: Combine (5 pts) We will provide you with a folder of Prokudin-Gorskii's black-and-white (grayscale) image composites (`prokudin-gorskii/` in the assignment zip). Each composite (alternatively triple-framed image or triptych) contains three grayscale photos preserved from the early 1900s. The composite looks like a three panel vertical comic strip, with each grayscale photo in the composite positioned vertically above one another. These photos represent images captured with a blue, green, and red filter. Choose a single composite from this folder (your favorite) and write a program in Python that takes the three grayscale panels and simply stacks them across the third color channel dimension to produce a single, colored image. We expect this stacked photo to look wonky and unaligned- fixing this is what you will do in Task 2. Make sure to save your images as RGB instead of BGR and include them in your report.

Specifically: Write a function that loads a grayscale tripled-framed image from `prokudin-gorskii/` with something like `plt.imread()`, chops it vertically into thirds, then saves a color image with each third as the correct color channel. Save the output colored image in your report.

Task 2: Alignment (25 pts) As you will have noticed, the photos are misaligned due to inadvertent jostling of the camera between each shot. Your second task is to fix this. You need to search over possible pixel offsets in the range of [-15, 15] to find the best alignment for the different R, G, and B channels. The simplest way is to keep one channel fixed, say R, and align the G and B channels to it by searching over the offset range both horizontally and vertically. Pick the alignment that maximizes a similarity metric (of your choice) between the channels. One such measure is dot product, i.e, R·G. Another is normalized cross-correlation, which is simply the dot product between the L2 normalized R and G vectors. After writing this function, run it on all of the images in `prokudin-gorskii/` and also on `'efros_tableau.jpg'`, so Professor Efros can have his photo restored to color. Include these aligned images and the offsets in your report. For full credit, your report needs to include properly aligned images - find a similarity metric that will accomplish this.

Specifically: Write a function to align the 3 channels of the image produced by Task 1. This function should output the (x,y) offsets required for shifting two of the color channels with respect to third. The third channel might then have a (0,0) offset. Save the newly aligned images from `prokudin-gorskii/` and `'efros_tableau.jpg'` in your report, along with the offsets for each color channel. Report the similarity metric you choose.

Hint: To offset the channels while keeping the same dimensions among them, you can use either `np.roll()` to roll over boundaries, or `np.pad()` for padding.

[1]Credit for this assignment goes to Alexei http://inst.eecs.berkeley.edu/ cs194-26/fa18/

Task 3: Pyramid (20 pts) For very large offsets (and high resolution images), comparing all the alignments for a broad range of displacements (e.g. [-30, 30]) can be computationally intensive. We will have you implement a recursive version of your algorithm that starts by estimating an image's alignment on a low-resolution version of itself, before refining it on higher resolutions. To implement this, you will build a two-level image pyramid. To do this, you must first scale the triple-frame images down by a factor of 2 (both the width and height should end up halved). Starting with your shrunk, coarse images, execute your alignment from Task 2 over the following range of offsets [-15, 15]. Choose the best alignment based on your similarity metric and treat it as the new current alignment. Then in the full resolution images, use this new current alignment as a starting place to again run the alignment from Task 2 in a small range [-15, 15]. Run this Pyramid task on the `'seoul_tableau.jpg'` and `'vancouver_tableau.jpg'` images. If your course project goes well.

Specifically: Use `cv2.resize()` to shrink each image in the triptych. Use your code from Task 2 to align them and get the intermediate offset. Shift the original images accordingly and align again at full resolution. Report the intermediate offset (at the coarse resolution), the next offset at the full resolution, and what the overall total offset was that includes both of these. Save the aligned images in color in your report.

Hint: If you're struggling, use a different color channel as your anchor!

Report You must submit a report that includes the offsets, color output images, and description required above. Your description should be such that a reader could implement what you've done after reading your report.

Extra Credit Implement some interesting alignment techniques and try to do even better than the restoration done by the Library of Congress (see: http://www.loc.gov/exhibits/empire/making.html)! To earn extra credit, include a description of your method in your report and improved versions of the composites in `prokudin-gorskii/`. Here are some ideas:

- Automatically crop borders that are not part of the image itself. This involves actively detecting what part of the grayscale image is just an artifact of poor scanning and removing that. (1 pt)
- Automatic contrasting. You can rescale image intensities so that the darkest pixel in the image (on the darkest color channel) is 0 and the brightest pixel is 1 (on its brightest color channel). Try different contrasts and see how it affects perceived image quality. (2 pts)

3 Color Spaces and illuminance [20 pts]

The same color may look different under different lighting conditions. Images `indoor.png` and `outdoor.png` are two photos of a same Rubik's cube under different illuminances.[2]

1. Load the images and plot their R, G, B channels separately as grayscale images using `plt.imshow()` (beware of normalization). Then convert them into LAB color space using `cv2.cvtColor()` and plot the three channels again. Include the plots in your report. (5 pts)
2. How do you know the illuminance change is better separated in LAB color space? (5 pts)

[2]The images are taken from this blog post: https://www.learnopencv.com/color-spaces-in-opencv-cpp-python/

3. Choose two different lighting conditions and take two photos of a non-specular object. Try to make the same color look as different as possible (a large distance on AB plane in LAB space). Below is an example of two photos of the same piece of paper, taken in the basement and by the window respectively.

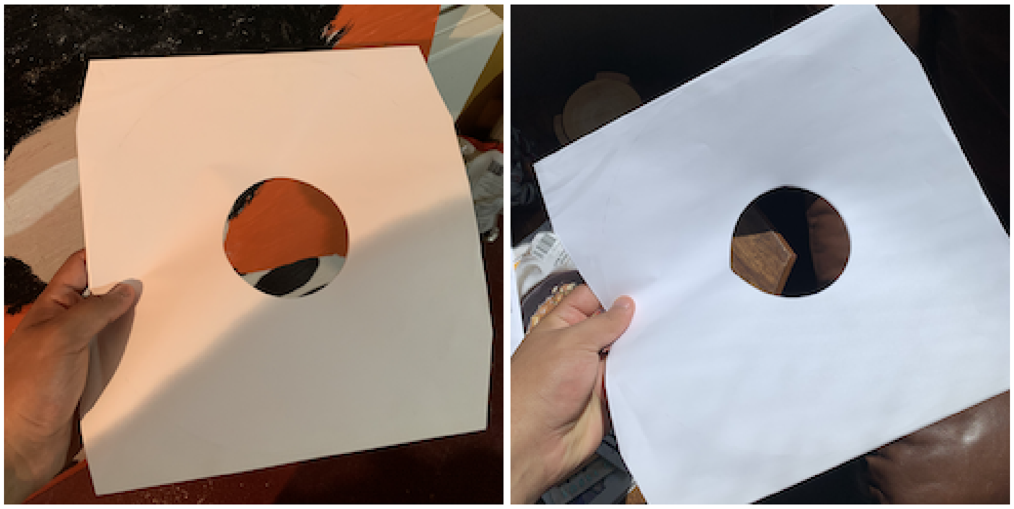

Figure 2: The same piece of paper in the basement and by the window

Submit the two images with names `im1.jpg` and `im2.jpg`, both cropped and scaled to 256×256. Under the same folder, also submit a file `info.txt` that contains two lines: Line 1 contains four integers `x1,y1,x2,y2` where we will take a 32×32 patch around the coordinate on each image and compare colors. (You can use `plt.imshow()` and `plt.show()` to bring up a window where you can select pixel with coordinates.) Line 2 is a description of the lighting conditions that you choose. An example of this file is provided for you in the folder. (10 pts)

Since the sense of color difference is subjective, we will display all images and patches on a webpage. Every student can vote for their favorite pair of images that illustrates color difference on Piazza. The winner will get **Extra Credits** (2 pts).

References

https://en.wikipedia.org/wiki/Camera_matrix
https://en.wikipedia.org/wiki/Rotation_matrix
http://inst.eecs.berkeley.edu/~cs194-26/fa18/hw/proj1/
https://sites.google.com/a/umich.edu/eecs442-winter2015/homework/color
https://www.learnopencv.com/color-spaces-in-opencv-cpp-python/