EECS 442 Computer Vision: Homework 1

Instructions

- This homework is **due at 11:59:59 p.m. on Wednesday February 10, 2021**.
- The submission includes two parts:
  1. **To Canvas**: submit a zip file of all of your code. This should contain a single directory which has the same name as your uniqname. If I (David, uniqname `fouhey`) were submitting my code, the zip file should contain a single folder `fouhey/` containing the following files:
     - Code files: `main.py` and `util.py`
     - `cube.gif` from Part 1
     - `im1.jpg`, `im2.jpg` and `info.txt` from Part 3

     We provide a script that validates the submission format here.
  2. **To Gradescope:** submit a pdf file with the answers to the questions from each section.

     The write-up must be electronic. *No handwriting!* You can use Word, Open Office, Notepad, Google Docs, LaTeX(try overleaf!), or any other form.

     You might like to combine several files. Here is an example online link for combining multiple PDF files: https://combinepdf.com/. Try also looking at this stack overflow post.

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(https://www.anaconda.com/distribution/). This is a Python package manager that includes most of the modules you need for this course.

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

- `numpy as np`
- `matplotlib.pyplot as plt` (for loading, saving and plotting images)
- `imageio` (for `gif` generation only)
- `opencv as cv2` (for processing images)

1 Camera projection Matrix [30 pts]

In this part we plot a cube from different camera placements and settings. We have provided some helper functions in `utils.py`. Some of these are taken from the **Projection and Dolly Zoom** notebook released on the course website, which you are encouraged to play with.

(a) Complete the function `rotY()` in `main.py` which takes an angle `theta` (in radian) and outputs the 3D rotation matrix of rotating by `theta` about the y-axis (right-hand rule). A good point to start looking into rotation matrix is 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, complete the 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? Include both plots in your report. (15 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.

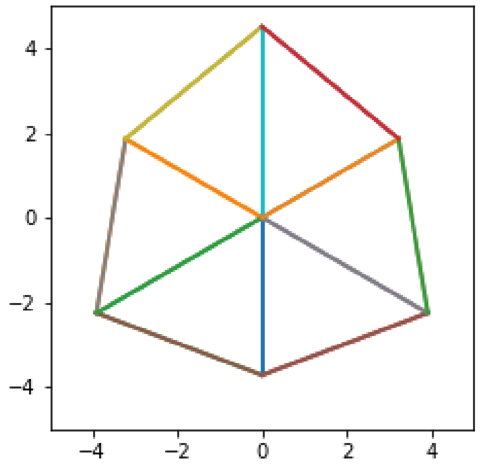

Figure 1: The diagonal of a cube projected to a single point

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. Note that the channels in the triptych are arranged as B, G, R from top to bottom, while channels in the combined image should be stacked in order R, G, B.

Specifically: Complete the function `split_triptych` that takes in an image, splits it vertically into thirds, then saves a color image with each third as the correct color channel. Choose and load an image from `prokudin-gorskii/`, then pass it to the function. Include the output colored image in your report.

Hint: Use `plt.imread()` and `plt.imsave()` to load and save images. For stacking the channels you can use `np.stack()` function.

Task 2: Alignment (20 pts) As you will have noticed, the photos are misaligned due to inadvertent jostling of the camera between each shot. To align the three channels, we can first keep one channel fixed, say R, and search for the optimal offsets (within a range of [-15,15] pixels both vertically and horizontally) for G and B channels respectively to align them to R channel. An optimal offset is one that maximizes a similarity metric between the channels. Here we use **normalized cross-correlation**, which is normalizing R and G channels with L2-norm (treating them as vectors) and then taking a dot product. Apply this method to all the images in `prokudin-gorskii/` and `"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. Figure 2 shows some examples of different alignment qualities.

(a) Complete the `normalized_cross_correlation` function to compute the similarity metric between two channels. Now, implement the `best_offset` function that takes two channels of an image and outputs the optimal offsets to align one channel to the other. (5 pts)

(b) Implement the `align_and_combine` function that calls the `best_offset` function twice to align a three-channel image with a fixed channel of your choice. Combine and save the aligned images from `prokudin-gorskii/` and `'efros_tableau.jpg'`. (10 pts)

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

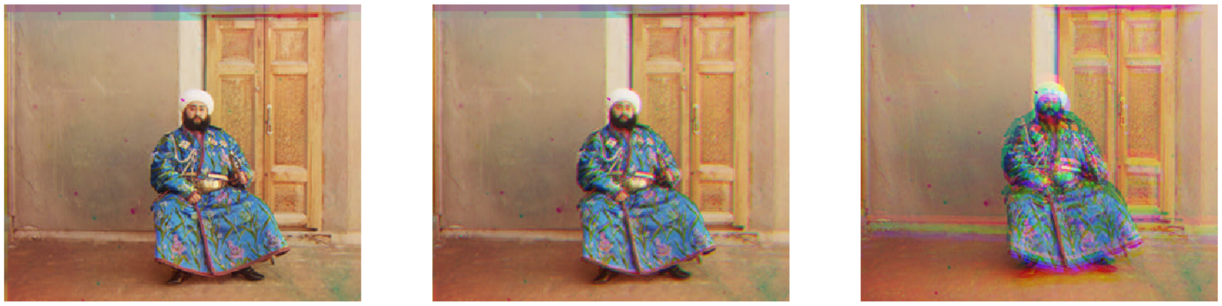

Figure 2: Examples for different alignment qualities: Good (full credit), OK (some of the credit) and Poor (you have a bug)

Hint: Use `np.roll()` to offset the channels while keeping the same dimensions among them. Keep in mind though, that if you roll the image to the left, the left edge goes to the right. Think about why this might be an issue if you're checking the alignment. To improve alignment quality, you can either calculate the similarity metric only on the region unaffected by offsetting artifacts, or try a different fixed channel. You may want to try both.

Advice: For the best_offset function the metric is a function that calculates the similarity between two channels. In the previous part, we have asked you to implement the normalized_cross_correlation function which we want to use as the similarity measure in this assignment. You can also implement and add another function that that takes into consideration the region unaffected by the offsetting artifacts. If you are unsure about how to pass a function as a variable, look here.

(c) Include these images as well as optimal offsets in your report. Additionally, experiment with another similarity metric that simply takes the dot product without normalization. Include two results in your report and comment on the quality of alignment. (5 pts)

Task 3: Pyramid (25 pts) For very large offsets (and high resolution images), comparing all the alignments for a broad range of offsets can be computationally intensive. An efficient solution is to start 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 three-level image pyramid, where each level downsamples the previous level by a factor of 4 in both height and width. Start with the level with the lowest resolution and perform alignment in Task 2 to get the optimal offsets on this level. Multiply this offsets by 4 and use it to roll the images on the previous level. Repeat this process until the full-resolution images are aligned. Search for offsets in range [-10, 10] on each level.

Specifically: Complete the function `pyramid_align` (define the input arguments as well) that **reuses** the functions from Task 2 either recursively or iteratively to perform the image pyramid alignment on images `'seoul_tableau.jpg'` and `'vancouver_tableau.jpg'`. In your report, Include the aligned color images at full resolution, the intermediate offsets on each level and the total offsets to align the original images. (15 pts) Then answer the following questions:

1. What is the equivalent range of offset in the original images that our pyramid approach searches over? (5 pts)

2. Suppose the similarity metric between two images of the same size H $\times$ W takes time $t = kHW$ to compute (k is a constant). How many times faster is the pyramid approach compared to a simple

search over the equivalent range? (5 pts)

Hint: You can use `cv2.resize()` to resize the image.

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()`. Then convert them into LAB color space using `cv2.cvtColor` and plot the three channels again. Read the documentation and be careful with value ranges. Include the plots in your report. (5 pts)

2. Which color space (RGB vs. LAB) better separates the illuminance change from other factors (such as hue)? Why? (5 pts)

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.

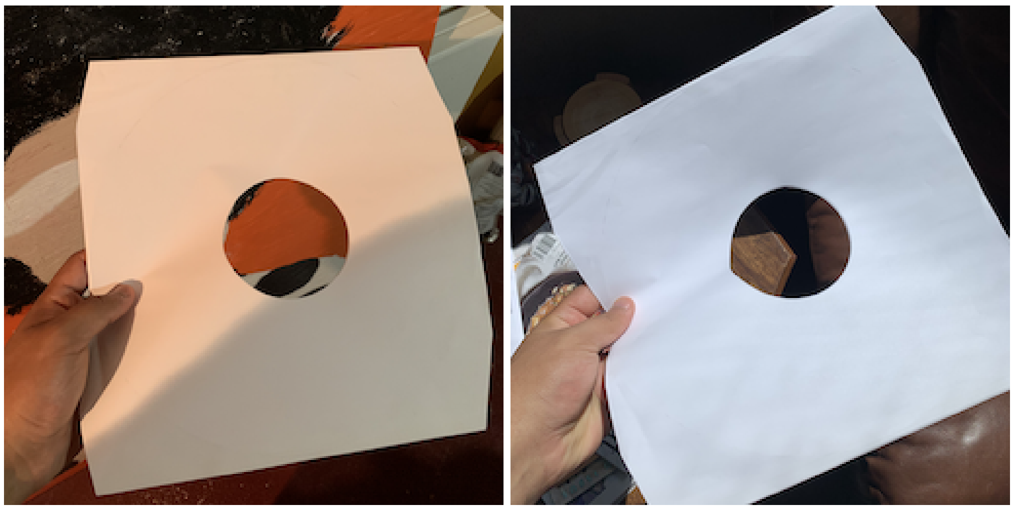

Figure 3: The same piece of paper in the basement and by the window

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 **only** 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. (10 pts)

Your `info.txt` should look like:

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

```
92 101 82 110
The first image was taken in the basement under a light.  The second
image was taken by the window under natural light.
```

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://scikit-image.org/docs/dev/api/skimage.color.html#skimage.color.rgb2lab

Changelog:

31 Jan: added advice on `metric` argument in Prudokin-Gorskii.