CSE803 HW1

Javen W. Zamojcin
zamojci1@msu.edu

2024, September 12th

1 Question 1

1.1 Part A.

To implement `rotY(theta)` I simply used the y-axis 3D rotational matrix equations from wikipedia, passing theta as the parameter to the wave functions, returning a 3x3 numpy matrix.

1.2 Part B.

For `rotX(theta)`, I repeated the process for `rotY(theta)` but using the x-axis rotation equations.

To implement successive rotations across different axes, I used the numpy `matmul(A, B)` function where A and B are the generated rotation matrices given in the order I wish. I then rendered two different cubes each using a product rotation matrix with a different axis-ordering.

Q: Are 3D rotation matrices commutative?
A: No. The axis order matters with 3D rotations and will affect the final rotation outcome.

1.3 Part C.

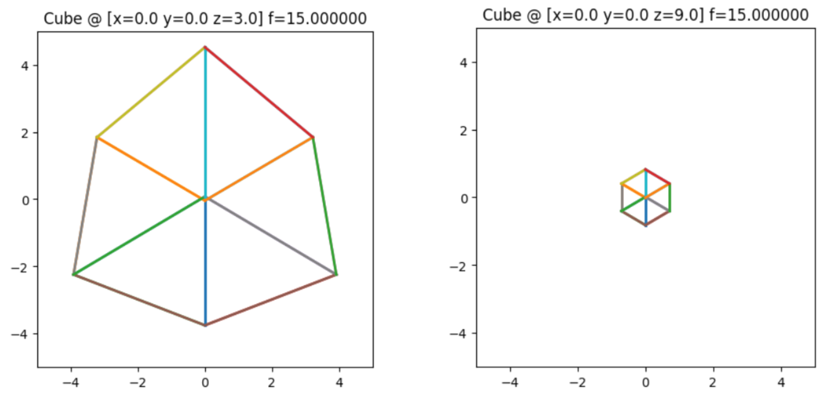
To achieve the cube rendering where one diagonal is projected to a single point, I just did trial-and-error to find the appropriate axis-ordering and axis-angles.

Order: x-axis first, y-axis second.
$\theta_x = \frac{\pi}{5}$
$\theta_y = \frac{\pi}{4}$

1.4 Part D.

To implement an orthographic camera, I rewrote the provided `render_cube()` and `project_lines()` functions.

To project the orthographic lines, I first still apply the rotation and translation to the respective line points. Then, I take the inner-product of a 2x3 Identity matrix by the respective 3x1 homogeneous coordinates to create the 2x1 projected points.



(a) Diagonal Projected to Single Point (b) Orthographic Camera

Figure 1: Q1 PartD Rendered Cubes

2 Question 2

2.1 Part A.

To achieve converting a grayscale tripled-framed image into a stacked colored version, I implemented a function `slice()`.

This function first calculates the dimensions of the provided 2D image matrix, and then calculates an offset variable as one-third of the image height. The RGB channels are then circumscribed into respective 2D matrices by vertically dividing the image by the offset value (B-G-R). The channels are then stacked into a single 3D matrix (Channel x Offset x Width). See figure 2.

2.2 Part B.

To align the colored photos, I first implement a function `load_images()` to load and slice all of the images in a given directory and return them as an array.

I then created a static list of tuple combinations for offsets in the range [-15, 15]

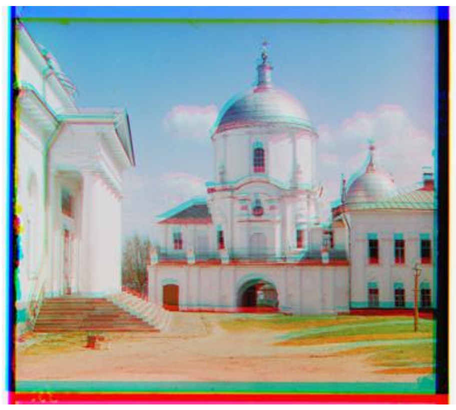


Figure 2: Unaligned-Colored 01112v.jpg

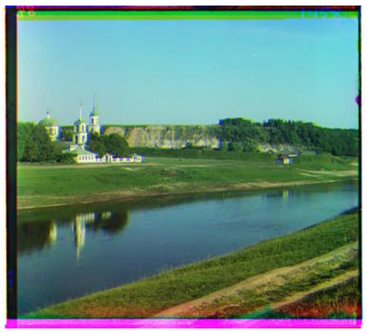
using the itertools library. This simplifies the process of iterating and scoring offsets between two channels. And for actually offsetting or shifting images, I created a `shift()` function which utilizes numpy's `roll()` to shift the given channel matrix by the given offset tuple (X, Y).

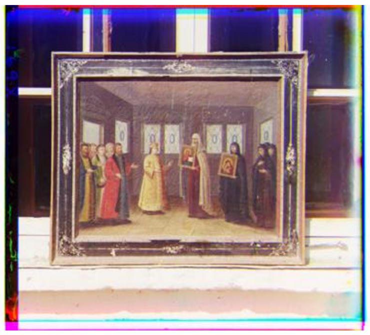
For scoring the alignment of two given channels, I implemented a function `score()` which can either use Normalized Cross-Correlation (the default) or Sum of Squared Distances. I also created the helper function `score_offsets()` which iterates through each offset, shifting one channel image by the offset, scoring and saving the result, as well as `find_offset()` which finds the best scoring (argmax) offset tuple from `score_offsets()`.

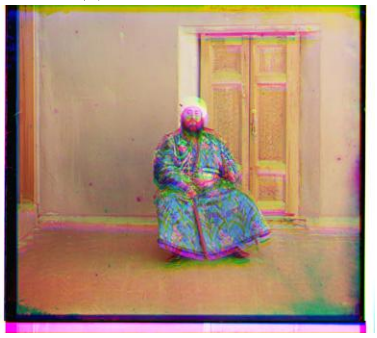
Finally, the `align()` method which first finds the best offsets between channel-1 and channel-0, and channel-2 and channel-0, before shifting these two channels to their final matrix form. See Table 1 and Figure 3.

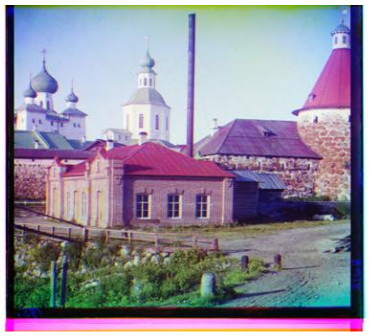
Image	Metric	R-Shift	G-Shift	B-Shift
00351v	NCC	(0, 0)	(-9, 0)	(-13, 1)
00398v	NCC	(0, 0)	(0, 1)	(-8, 2)
00153v	NCC	(0, 0)	(-13, 2)	(-11, -3)
00149v	NCC	(0, 0)	(-5, 0)	(-9, -1)
00125v	NCC	(0, 0)	(-4, 1)	(-10, 2)
01112v	NCC	(0, 0)	(-8, -1)	(-8, -3)
efros-tableau	NCC	(0, 0)	(0, 10)	(0, 5)

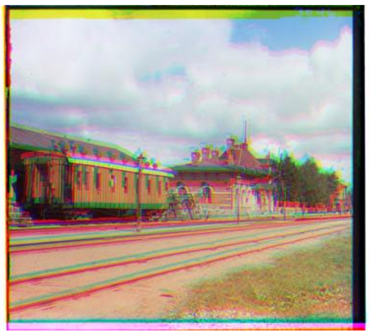
Table 1: Q2 PartB Photo Alignment Parameters

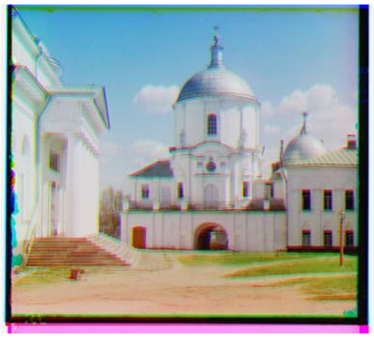

(a) Aligned-00125v.jpg


(b) Aligned-00149v.jpg


(c) Aligned-00153v.jpg


(d) Aligned-00351v.jpg


(e) Aligned-00398v.jpg


(f) Aligned-01112v.jpg

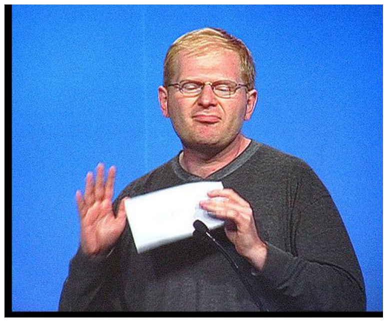

(g) Aligned-Efros.jpg

Figure 3: Q2 PartB Aligned-Colored Photos

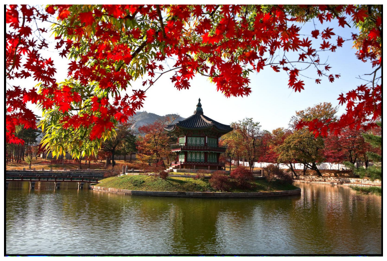
2.3 Part C.

To create a recursive version of the photo alignment, I first implement a `resize()` function which simply scales the individual channel dimensions in half. Then the two photos from the ”tableau” directory are loaded, sliced, copied and resized, and have the modified alignment algorithm applied to them.

The recursive alignment algorithm essentially finds first the offsets for the resized (half-scaled) image copies, applies this first offset to the full-scale images, then performs the same alignment process as expected from the previous task. The offsets for each step are recorded, and the total offsets are calculated as the sum of the step offsets, for each image. See Table 2 and Figure 4.

Image	Metric	R-Shift	G-Shift	B-Shift
seoul-tableau	NCC	(0, 0)	(0, 6)	(-2, -1)
vancouver-tableau	NCC	(0, 0)	(-1, 22)	(8, 12)

Table 2: Q2 PartC Photo Alignment Parameters


(a) Aligned-Seoul.jpg

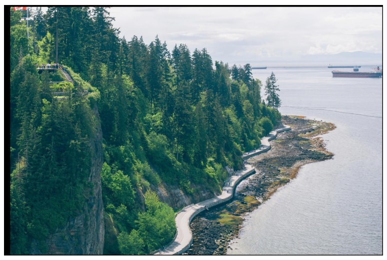

(b) Aligned-Vancouver.jpg

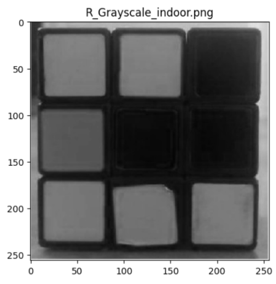
Figure 4: Q2 PartC Aligned-Colored Photos

3 Question 3

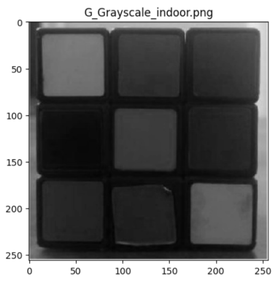
3.1 Part A.

For displaying the RGB grayscale channels, I simply loaded the image matrices and used matplotlib’s `imshow()` function on the respective channel sub-matrices, using the `cmap='gray'` parameter. I also multiplied the channel matrices by their norms as I wasn’t sure if the [0, 1] normalization would affect how the images are displayed. See Figure 5.

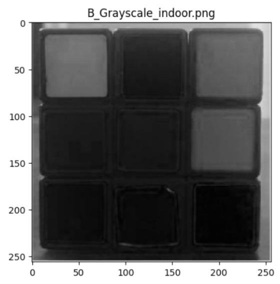
For displaying the LAB channels, I did the same process except for first converting the loaded images to LAB through the `cv2.cvtColor(img, cv2.COLOR_BGR2LAB)` function, and not multiplying the individual channels by their respective norms either. I also left the `cmap='gray'` parameter for displaying the images but I wasn't sure on this from the instructions. See Figure 6.



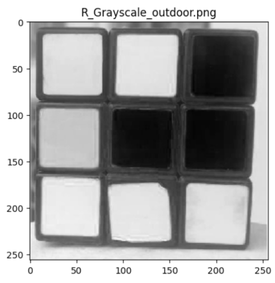
(a) R-Grayscale-Indoor



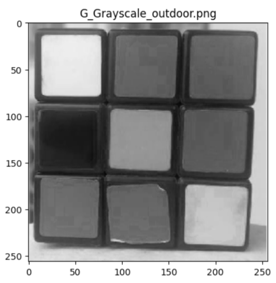
(b) G-Grayscale-Indoor



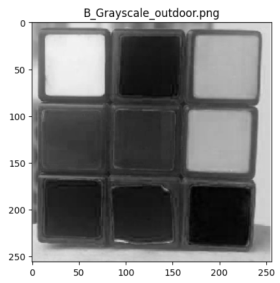
(c) B-Grayscale-Indoor



(d) R-Grayscale-Outdoor



(e) G-Grayscale-Outdoor



(f) B-Grayscale-Outdoor

Figure 5: Q3 PartA RGB Grayscale Photos

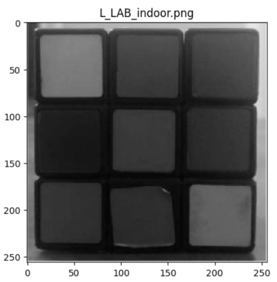
3.2 Part B.

Q: How do you know the illuminance change is better separated in LAB color space?

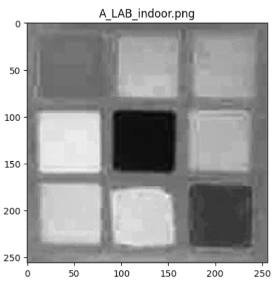
A: In the RGB color space, brightness information is directly coupled with color information across all three color channels. In LAB, the L channel is independent of color and will only capture brightness information. Between the indoor and outdoor LAB photos, there is a noticeable change in the L channel display but not in the A or B channels, which encode color information and not luminosity.

3.3 Part C.

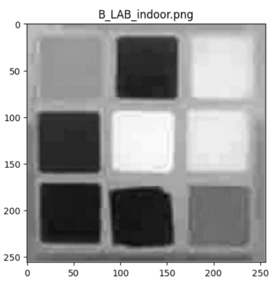
The 256x256 photos and info text file may be found in the `Q3_P3` sub-directory.



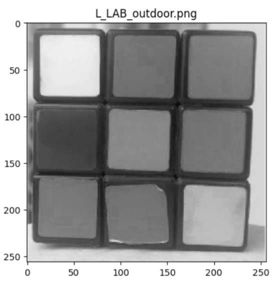
(a) L-LAB-Indoor



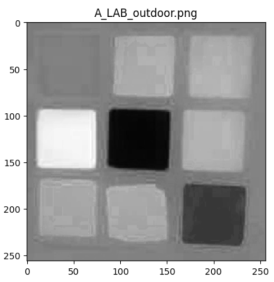
(b) A-LAB-Indoor



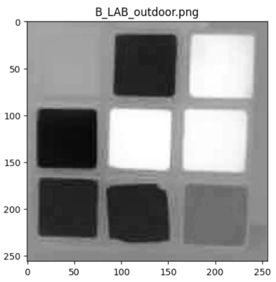
(c) B-LAB-Indoor



(d) L-LAB-Outdoor



(e) A-LAB-Outdoor



(f) B-LAB-Outdoor

Figure 6: Q3 PartA LAB Grayscale Photos