ELEC/COMP 447/546
Assignment 1
Due: Jan 31, 2023, 11:59 PM

1.0 Basic Image Operations (10 points)

In this problem, you will gain some experience working with NumPy and OpenCV to perform basic image manipulations.

1.1 Combining Two Images

a. Read in two large (> 256 x 256) images, A and B into your Colab notebook (see sample Colab notebook that was shared with the class earlier).

b. Resize A to 256x256 and crop B at the center to 256x256.

c. Create a new image C such that the left half of C is the left half of A and the right half of C is the right half of B.

d. Using a loop, create a new image D such that every odd numbered row is the corresponding row from A and every even row is the corresponding row from B.

e. Accomplish the same task in part d without using a loop. Describe your process.

1.2 Color Spaces

a. Download the peppers image from this link. Return a binary image (only 0s and 1s), with 1s corresponding to only the yellow peppers. Do this by setting a minimum and maximum threshold value on pixel values in the R,G,B channels. Note that you won't be able to perfectly capture the yellow peppers, but give your best shot!

b. While RGB is the most common color space for images, it is not the only one. For example, one popular color space is HSV (Hue-Saturation-Value). Hue encodes color, value encodes lightness/darkness, and saturation encodes the intensity of the color. For a visual, see Fig. 1 of this wiki article. Convert the image to the HSV color space using OpenCV's cvtColor() function, and try to perform the same task by setting a threshold in the Hue channel.

c. Add both binary images to your report. Which colorspace was easier to work with for this task, and why?

2.0 2D Geometric Transforms (15 points)

Geometric transformations are fundamental tools used in a variety of computer vision and computer graphics applications. In this problem, you will write your own code to warp images using 2D geometric transforms.

2.1 Write functions to produce transformation matrices

Write separate functions that output the 3 x 3 transformation matrices for the following transforms: translation, rotation, similarity (translation, rotation, and scale), and affine. The functions should take as input the following arguments:

1. Translation: horizontal and vertical displacements

2. Rotation: angle
3. Similarity: angle, horizontal/vertical displacements, and scale factor (assume equal scaling for horizontal and vertical dimensions)
4. Affine: 6 parameters

The output of each function will be a 3 x 3 matrix.

2.2 Write a function that warps an image with a given transformation matrix
Next, write a function *imwarp(I, T)* that warps image *I* with transformation matrix *T.* The function should produce an output image of the same size as *I*. See Fig. 1 for an example of a warp induced by a rotation transformation matrix. Make the origin of the coordinate system correspond to the CENTER of the image, not the top-left corner. This will result in more intuitive results, such as how the image is rotated around its center in Fig. 1.

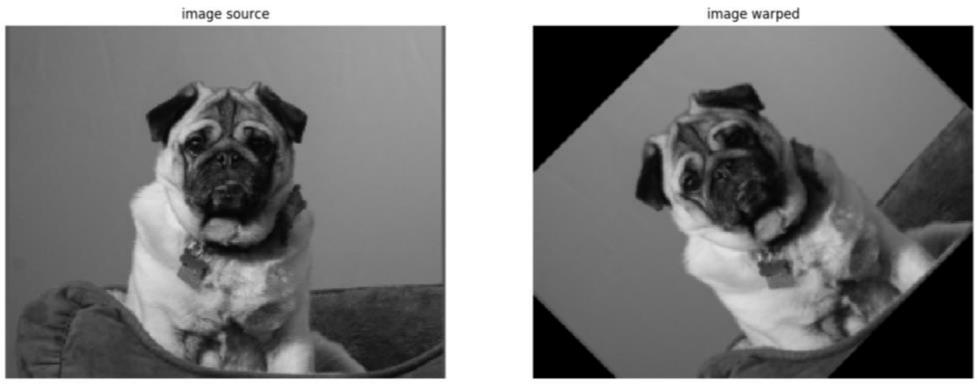

Fig. 1: Example of an input image (left) transformed by a rotation matrix, resulting in a 'warped' image (right).

Hint 1: Consider the transformation matrix *T* to describe the mapping from each pixel in the *output* image back to the original image. By defining *T* in this way, you can account for each output pixel in the warp, resulting in no 'holes' in the output image (see Lec. 03 slides).

Hint 2: What happens when the transformation matrix maps an output pixel to a non-integer location in the input image? You will need to perform *bilinear interpolation* to handle this correctly (see Lec. 03 slides).

Hint 3: You may find NumPy's meshgrid function useful to generate all pixel coordinates at once, without a loop.

2.3 Demonstrate your warping code on two color images of your choice
For each of the two images, show 2-3 transformations of each type (translation, rotation, similarity, affine) in your report.

3.0 Cameras (15 points)

3.1 Camera Matrix Computation

a. Calculate the camera intrinsic matrix **K**, extrinsic matrix **E**, and full rank 4×4 projection matrix **P = KE** for the following scenario with a pinhole camera:
    - The camera is rotated 90 degrees around the x-axis, and is located at $(1, 0, 2)^T$.

    - The focal lengths f_x, f_y are 100.
    - The principal point $(c_x, c_y)^T$ is (25, 25).

b. For the above defined projection, find the world point in inhomogeneous coordinates x_w which corresponds to the projected homogeneous point in image space $x_I = (25, 50, 1, 0.25)^T$.

3.2 Field of view and focal length
You are given two cameras with the exact same sensor. The first camera has a wider field-of-view (FOV) than the second, with all other camera parameters being the same. Which camera has a shorter focal length and why?

4.0 Relighting (10 points) (ELEC/COMP 546 ONLY)

In this problem, you will perform a simple version of *image relighting*, the task of changing the lighting on a scene. To do this experiment, you will need two light sources (such as ceiling lights, floor lamps, flashlights etc.) and a couple of scene objects. Set up a static scene similar to the one shown in Fig. 2 (the light sources do not have to be seen in the frame, but try to have them illuminating the scene at two different angles), and a camera such that it is stationary throughout the experiment (cell phone leaning against heavy object or wall is fine). Let us label the two lamps as LAMP1 and LAMP2.

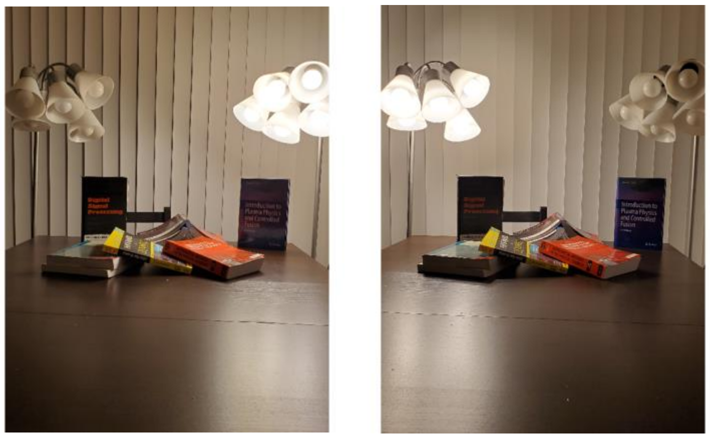

Fig. 2: Example setup with two lamps.

a. Capture the image of the scene by turning on LAMP1 only (image *I1*). Now capture an image by turning on LAMP2 only (image *I2*). Finally, capture the image with both LAMP1 and LAMP2 on (image *I12*). Load and display these images into your Colab notebook.

b. Now, you will create a synthetic photo (*I12_synth*) depicting the scene when both of the lamps are turned on by simply summing *I1* and *I2* together: *I12_synth* = *I1* + *I2*. Also compute an image depicting the difference between the synthetic and real images: D = *I12_synth* - *I12.*

c. In your report, show *I1, I2, I12, I12_synth,* and *D* side by side. When displaying *D*, make sure to rescale *D*'s values to fill the full available dynamic range ([0,1] for float, or [0,255] for uint8). You can do this with the following operation:

(D - min(D))/(max(D) - min(D)).

d. How good is your synthetic image compared to the real one? Where do they differ the most?

Submission Instructions

All code must be written using Google Colab (see course website). Every student must submit a zip file for this assignment in Canvas with 2 items:

1. An organized report submitted as a PDF document. The report should contain all image results (intermediate and final), and answer any questions asked in this document. It should also contain any issues (problems encountered, surprises) you may have found as you solved the problems. The heading of the PDF file should contain:
    a. Your name and Net ID.
    b. Names of anyone you collaborated with on this assignment.
    c. A link to your Colab notebook (remember to change permissions on your notebook to allow viewers).
2. A pdf copy of your Colab notebook.

Collaboration Policy

I encourage collaboration both inside and outside class. You may talk to other students for general ideas and concepts, but you should write your own code, answer questions independently, and submit your own work.

Plagiarism

Plagiarism of any form will not be tolerated. You are expected to credit all sources explicitly. If you have any doubts regarding what is and is not plagiarism, talk to me.