EEP 596 Computer Vision

HW #1

The purpose of this homework is to familiarize you with some basic image processing and computer vision concepts using Python and associated libraries. You are free to work with fellow students but should turn in your own work. Specifically, you should be able to explain anything in your assignment if asked.

Submit your assignment as "Assignment1.py" and "Report.pdf" on Gradescope. The report should include all function outputs (e.g., images or printed text). There's no strict format or need for explanations. The report won't be graded but helps ensure transparency and assist in case of grading conflicts.

You should develop code with the skeleton provided and should not import libraries other than the ones listed below. You could use `print()` and `cv2.imsave()` to help you to generate outputs for the report, but these should be commented out before submission.

Warning: Python is an easy language, but Python packaging / ecosystem is a mess. It is best to keep your expectations low.

The questions are as follows.

0. **Set up.** Install the latest version of Python, NumPy, PyTorch, Matplotlib, OpenCV. It is recommended to install Anaconda, as it includes several of these automatically.

   You can test your setup by running Python from the command line, then importing modules and printing their versions.

   ```
   import numpy as np
   import matplotlib
   import cv2

   print(np.__version__)
   print(matplotlib.__version__)
   print(cv2.__version__)
   ```

   There are two popular ways to run Python:
   - Command line/ .py file. Edit code with your favorite editor (Visual Studio Code, or VSCode, is a good choice.) Save to `foo.py` (for example). Then, type `python foo.py` at the command prompt to run (or use the integrated interpreter via the IDE).
   - Jupyter / IPython. Type `jupyter notebook` at the command prompt, and your browser should automatically open to a page that allows you to work interactively. Alternatively you could use Google colab to edit/run IPython files. It's a common practice to use integrate Google drive with Google colab when your code accesses files (e.g., read or write images)

1. **Image load / save.** Load the `original_image.png` image provided. Print the data type of the image, and the data type of the pixels in the image. Print the image dimensions. Check if the datatype of the image and a pixel in the image are in `uint8` format. Return the data types and image dimensions.
2. **Color channels.** Create a new RGB image by setting the green and blue pixel values to zero everywhere. The resulting image should look red. Note for this and following tasks: The returned image from the function should be in BGR format for channels and `uint8` for pixels (the native cv2 format). Return the red image.
3. **Photographic negative.** Create a new RGB image (from the original image) by subtracting each pixel value from 255. The resulting image should look like a photographic negative. Return the new negative image.
4. **Swap color channels.** Create a new RGB image (from the original image) by swapping the red and blue channels. Return the new image.
5. **Foliage detection.** Create a new binary image that contains an `ON` pixel wherever the original input pixel is approximately green, and `OFF` everywhere else. Threshold in such a way that green is greater than or equal to 50 and others are less than 50. Set `ON` pixels to 255, and `OFF` pixels to 0. Return the thresholded image.
6. **Shift.** Translate the original image to the right by 200 pixels, and down by 100 pixels. Fill in the missing values with zero. Return the shifted image.
7. **Rotate.** Rotate the original image clockwise by 90 degrees. Return the rotated image.
8. **Similarity transform.** Write a function to apply a similarity transform to an input image, yielding an output image of the same size. The function should take scale, rotation angle, and translation as input (all as floating point values). It should scale, rotate, then translate, in that order. Use nearest neighbor interpolation for simplicity but be sure to use "inverse mapping" to ensure that every pixel in the output gets painted. Test this on the original image above by passing the values scale=2.0, theta=45.0 (degrees), shift=[100,100]. Return the transformed image.
9. **Grayscale conversion.** Convert the original image into grayscale. Use the formula `gray = (3 * R + 6 * G + 1 * B) / 10`. (Note that this formula has at least two fundamental problems with it, but it's commonly used so we'll ignore those problems for now.) Return the grayscale image.

The next two tasks will be applied to the binary image `glasses_outline.png`.

10. **Moments.** Write a function to compute the first- and second-order moments (both standard and centralized) of a binary image. Apply the function to the image `glasses_outline.png`. Return the results.
11. **Binary image processing.** Using the function from the previous problem, write a function to compute the orientation and eccentricity of `glasses_outline.png`. Orientation should be clockwise w.r.t. the positive horizontal axis, in degrees. Additionally, draw a "best-fitting ellipse" on the BGR `glasses_outline.png` in the red channel. Please use the `cv2.ellipse` method with a line thickness of `1px`. Do not specify a line type or shift. The ellipse should be drawn in red `(0, 0, 255)` and should appear on top of the white `(255, 255, 255)` outline of the glasses. Return the orientation, eccentricity, and modified image with the ellipse.