EEE-6512 Image Processing and Computer Vision

Fall 2020 Homework #5

October 24th, 2020

**Due: November 7th, 2020, 11:59 PM**

This assignment should be completed individually by the student. Proper citation should be provided for any references used.

**Part I: Textbook Questions [50 points]**

Answer the following questions from the textbook:

7.3, 7.4, 7.5, 7.7, 7.8, 7.9, 7.10, 7.14, 7.15, 7.18

**Part II MATLAB Programming [50 points]**

Please read requirements carefully. Solutions that do not follow provided specifications will not receive credit. MATLAB toolbox functions which detect edges or corners or create a psuedocolor image are disallowed in all code intended for submission (e.g. functions such as corner(), edge(), and imoverlay() are not allowed). All other MATLAB toolbox functions are allowed. Loops are also allowed, just make sure *hw5.m* runs within 5 minutes. Test your functions using *hw5.m*, *img1.pgm, img2.pgm, and img3.pgm.*

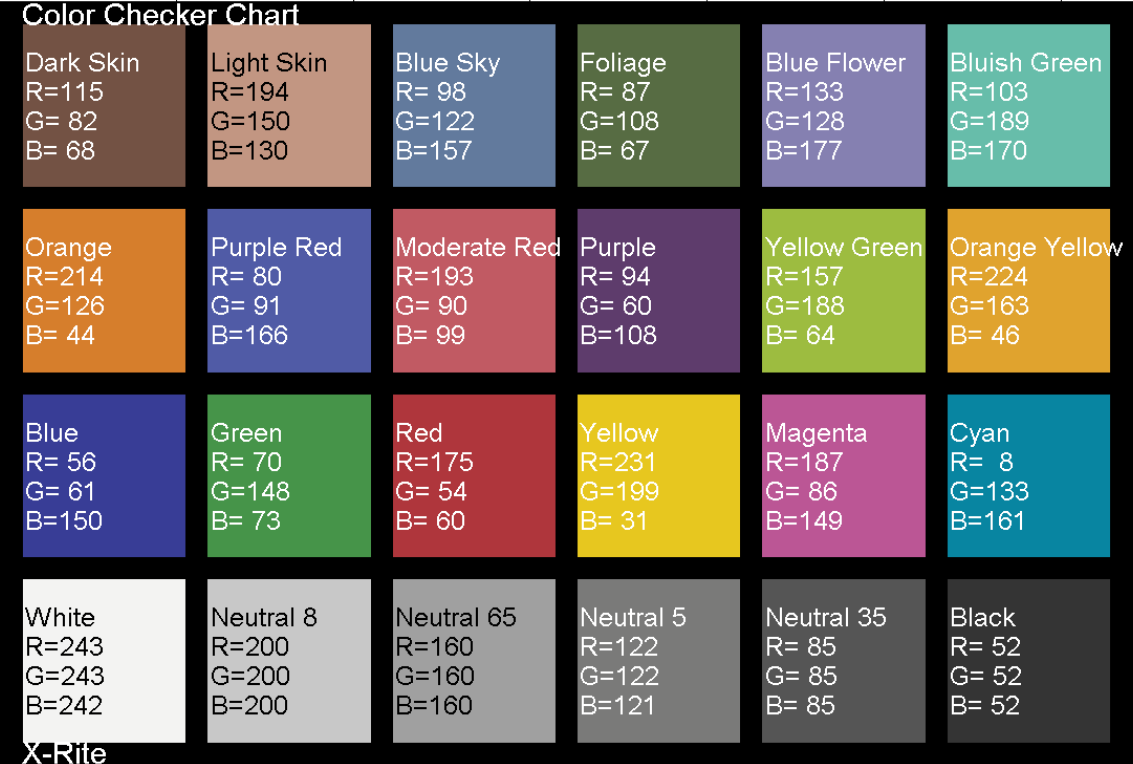
* Write a function, *myCannyEdgeDetector*, which:
  + accepts a grayscale image and any additional hyperparameters you wish to change between runs
  + detects edges in an image using the canny algorithm, as described in the book. There are multiple way to program this algorithm, feel free to make some design decisions.
  + Displays and returns the final result as a pseudocolor image with edges in blue (i.e. RGB = (0, 0, 255)). Make sure the lines are thick enough to clearly see the detected edges. All other pixels should have the same grayscale values as the input image
* Write a function, *myHarrisCornerDetector*, which
  + accepts a grayscale image and any additional hyperparameters you wish to change between runs
  + detects corners in an image using the harris algorithm, as described in the book. There are multiple way to program this algorithm, feel free to make some design decisions.
  + Displays and returns the final result as a pseudocolor image with corners in red (i.e. RGB = (255, 0, 0). Make sure the dots/squares are thick enough to clearly see the detected corners. All other pixels should have the same grayscale values as the input image
* Using the given script, *hw5.m*, edit ONLY:
  + the commented code at the top with your name and username
  + the parameters in the *myCannyEdgeDetector* function calls (lines 11-13). Tune the parameters for each image such that most edges are detected.
  + the parameters in *myHarrisCornerDetector* function calls (lines 16-18). Tune the parameters for each image such that most corners are detected.
  + Do not touch anything else. Ensure if all program files and input images are in the same folder, that *hw5.m* has exactly 18 lines, runs without error, and outputs exactly 6 images.
* Programming questions (PUT IN WRITEUP):
  + For each of the 3 input images, include the following intermediate/output images in your report:
    - *myCannyEdgeDetector*: gradient magnitude, gradient phase, thresholded gradient magnitude (as in pp. 336), and the output pseudocolor image with edges in blue, as described above. You may sub-image the 3 Canny intermediate images to save space in the document.
    - *myHarrisCornerDetector*: Harris cornerness response before non-max suppression, and the output pseudocolor image with corners in red, as described above
    - Reminder: turn off imshow functionalities for the intermediate images before submitting the code by commenting the lines out
  + Questions:
    - *myCannyEdgeDetector*
      * What input parameters does the program accept (besides the input image) and what purpose does each serve in the algorithm?
      * Which method did you use to compute the gradient and why did you choose this method over others?
      * Looking at the results of the given images, where does your algorithm perform well and why? Where does your algorithm perform poorly and why?
      * How could your algorithm be improved?
    - *myHarrisCornerDetector*
      * What input parameters does the program accept (besides the input image) and what purpose does each serve in the algorithm?
      * Looking at the results of the given images, where does your algorithm perform well and why? Where does your algorithm perform poorly and why?
      * How could your algorithm be improved?

To receive full credit for this assignment, you should submit four files. 1.) A document containing answers to the textbook and programming questions (.DOC, .DOCX, or PDF file) 2.) Commented MATLAB function *myCannyEdgeDetector.m*. 3.) Commented MATLAB function *myHarrisCornerDetector.m*. 4) MATLAB script, *hw5.m*. Students should ensure hw5.m has exactly 18 lines, runs without error, and outputs exactly 6 images to avoid receiving point deductions.

**Part III Extra Credit [25 points]**

Please read requirements carefully. Solutions that do not follow provided specifications will not receive credit. You are free to use any built-in/toolbox functions within MATLAB to accomplish this task, except functions from the deep learning toolbox. Data and background were taken from [1].

**Background**: In computer vision and machine learning, it is vital to normalize images taken under different photography conditions. Such normalization may be achieved by using a standard Color Checker (CC), a palette of distinctly colored square patches with known RGB reference values. To normalize data, an image is first taken of the CC and a color profile is extracted. Then, this color profile can be compared with a published standard, such as the Color Checker Chart shown in the image below [2] and used to normalize subsequent images taken under similar conditions. More details can be found at [3-4]. One of the greatest challenges in CC normalization is automatically extracting the color palette.



**Data:** Afifi et. al. has generated a dataset of 65,416 sRGB images rendered using different white-balance presets in the camera (e.g., Fluorescent, Incandescent, Dayligh) with different camera picture styles (e.g., Vivid, Standard, Neutral, Landscape) in [1]. For this homework, you are only concerned with 8 images, which are provided in canvas.

**Challenge**: Write a function, *CCProfileExtractor*, which:

* accepts any RGB color image of a CC and outputs a 4-pixel row, 6-pixel column RGB image, with each pixel representing the average color of each color swatch in the CC, as shown below:



* You must:
  + Use at one method for edge detection
  + Use at least one method for corner detection
  + Include in your report
    - The output of all 8 images
    - Where does the algorithm perform well and why? poorly and why?
    - How can it be improved?
* Note: code will be tested on 2 randomly sampled images taken from the provided dataset. The extracted profiles must match the profiles images included in the report.

To receive full credit for the extra credit part, you should submit two files. 1.) A document containing an explanation of how your code works, with enough information/intermediate images for another student in the class to roughly duplicate your work and understand why each step was taken (.DOC, .DOCX, or PDF file) 2.) An M-file containing commented MATLAB code for the program *CCProfileExtractor*. Students should ensure that their M-files execute without errors to avoid receiving point deductions.

**References**

[1] Mahmoud Afifi, Brian Price, Scott Cohen, and Michael S. Brown, "When Color Constancy Goes Wrong: Correcting Improperly White-Balanced Images", CVPR, 2019.

[2] ColorCheckerChart.LibraryCatalog: [www.mathworks.com](http://www.mathworks.com). [Online]. Available: <https://www.mathworks.com/matlabcentral/fileexchange/38236-color-checker-chart>

[3] <https://plantcv.readthedocs.io/en/latest/transform_color_correction_tutorial/>

[4] R. Baumann, “Automatic ColorChecker Detection, a Survey,” Jul. 2015.[Online]. Available: <https://ryanfb.github.io/etc/2015/07/08/automaticcolorcheckerdetection.html>