CS 675 – Computer Vision – Spring 2018
Instructor: Marc Pomplun

Assignment #1

Posted on February 14 – due by February 27, 2pm

Writing Computer Vision Programs

For the programming tasks in this course, I decided to keep things simple, efficient, and convenient. Since this is not a software engineering class, we will not develop complete, user-friendly applications but just implement some computer vision algorithms and play around with them. And we will do this in the plain, old C programming language. There are several reasons for this: (1) C compilers are freely available for any platform and operating system, and the tiny image reading/writing library I wrote for you should be compatible with all of them; (2), you most likely know C already or can quickly learn it because of its similarity to Java; (3) the compiled code will execute extremely fast, which will be very helpful for some transformations and machine learning approaches; and (4) programming in a low-level language without specific libraries will force you to think about every elementary step in the algorithms and really understand how they work.

Regarding the digital images, we are going to work with the simplest file format, which is called Netpbm. These are the files with extensions ".PBM" (black-and-white, 1 bit per pixel), ".PGM" (grayscale, 8 bits per pixel), and ".PPM" (RGB color, 24 bits per pixel). You can find out all about them here:

http://en.wikipedia.org/wiki/Netpbm_format

I wrote a very small C library for reading, writing, creating, and deleting images in these formats, which should be easy to understand and use. You can download the "netpbm.zip" file containing "netpbm.c" and "netpbm.h" files, plus a sample program named "netpbm_test.c" and the sample images "sample.ppm" and "text_image.pbm" from the course homepage, section "Software." You should put all files into the same directory and include the C files into a project for your compiler. When you compile and run the sample program, it should load the "sample.ppm" image, produce several variants of it, and write them to the same directory. This program should be very easy to understand and will show you how to implement your own image processing and computer vision algorithms. Notice that this library only works with the binary versions of the Netpbm formats. There are also ASCII versions of these formats, which are horribly inefficient and should be avoided. While the Netpbm format is not as common as, for instance, the JPEG or PNG formats, there is free software available for all

common operating systems that can read, write, and convert Netpbm files. I recommend the cross-platform image editor GIMP (http://www.gimp.org), and for Windows also the image viewer IrfanView (http://www.irfanview.com/). For our purpose, the advantage of the Netpbm format is that it is easy to understand and only contains the information we need. If you have any questions about the code or the format, please do not hesitate to ask me by e-mail or in person.

Please use the Apply program to create an account for this course (see https://www.cs.umb.edu/sp/resources/other/faqs/#FAQ02). Then create subfolders "hw1" to "hw6" and put the "homework1.c" file (that you are going to create) into the "hw1" directory. You can submit the answers to the non-programming questions also as files in the same directory or hand in a hard copy at the beginning of the class on February 27.

Question 1: Binary Image Processing

After having played around with the library, you should be ready for your first project. We will start with binary image processing. Download the binary image "text_image.pbm" from the course homepage into the same directory as the library. Put all of your code, which has to use the Netpbm library, in a file named "homework1.c" in the same directory. Please do not modify the library files. Your program should do the following things:

(a) You will see that the image contains some text with a lot of positive and negative noise. In order to remove it, implement the expanding and shrinking operations and let your program execute a sequence of them on the image. Write the resulting, hopefully nicer-looking, image into a file named "text_cleaned.pbm." Note that the library loads binary images by assigning each pixel an intensity value of 0 (black) or 255 (white) and also expects these values (actually, using a threshold $\theta = 128$) for when writing an image. It is implemented this way, instead of using 0 for white and 1 for black, so that all image types can be handled equally. In other words, while images are in memory, they have no PBM/PGM/PPM type assigned.

(b) Now your program will take the cleaned-up image and label its connected components using the efficient algorithm (Algorithm 4.1.2 in Section 4.1). Use the Matrix type that is defined in the library to store all pixel labels.

(c) Let us use the label information to color and count the letters in the image text. We do not want to consider the commas and periods, and obviously we do not want to count the dots of i's or j's separately. To do this, you should find an appropriate size threshold and simply ignore all connected components that are smaller than that threshold. For the connected components of above-threshold size, give each of them an individual, random color that differs visibly from both black and white. Leave all other parts of the image unchanged. Write the resulting image into a file named "text_colored.ppm." and print out the number of above-threshold sized connected components. This should be the number of letters in the text.

Question 2: Colors

Given a range of the R, G, and B components from 0 to 255, the color defined by R = 220, G = 210, and B = 70 is yellowish. Convert this color into the HSI color space using the method we discussed in class and showing each step of your computation.

Question 3: A Useful Inaccuracy?

Please take a look at slide 11 from January 25. We perceive patch A as being darker than patch B. However, as demonstrated in slide 12, the patches A and B are actually equiluminant. Please explain in your words why this misperception occurs and why it is actually an advantage for us to perceive intensity in this way rather than in a physically accurate way.