Trey Blankenship   
CS 6384.001  
3/27/2017  
  
My overall implementation strategy differed from what was shown in class, owing largely to the unreliability of the Windows-compatible library for OpenCV (It made my anti-virus nervous all the time, and while I got a working project going in Visual Studio, not a week later did it completely fail. I gave up on the C++ implementation around the time Visual Studio 2015 started “opening” source files, only to not show me the contents of the files.   
  
To that end, with my growing interest in the small British microcomputer, I decided to implement the project with Python 3 bindings on the Raspberry Pi. While my implementation is a great deal slower than a comparable C++ implementation (loops being something Python seems to have an irrational hatred for), Python proved to be easier to write and debug. While I had made attempts to optimize the code, I cannot say that these attempts were successful, owing to the limited hardware given to me by the Raspberry Pi 3. I cannot imagine trying to run this program on a Raspberry Pi Zero W; that has trouble opening more than 3 tabs in the built-in web browser.   
  
In general, to prevent out of bounds values (such as those caused by division by zero), I just set anything that would divide by zero to zero, as I did in the homework assignments. I also clipped values that went outside the bounds specified in the assignment.

Program 1:

|  |  |
| --- | --- |
| C:\Users\Unimp\AppData\Local\Microsoft\Windows\INetCacheContent.Word\Luv.png | C:\Users\Unimp\AppData\Local\Microsoft\Windows\INetCacheContent.Word\xyY.PNG |
| Luv | xyY |

My conversion formula was as follows:   
Luv 🡪 XYZ 🡪 sRGB  
xyY 🡪XYZ 🡪 sRGB

As a final stress test to ensure the correctness of my implementation of the conversion function, I experimented in the other programs, and found that by following the conversion algorithm:  
  
BGR 🡪 sRGB 🡪 Luv 🡪sRGB 🡪BGR   
  
I obtained the same BGR vector as I did from the initial image. Thus, I know that my conversion method is correct. In my first submission, I made a major error. I reversed the order of the matrix operations described in the handouts. Thus, I was getting incorrect color values. In the end, it was a simple fix, resulting in a change to exactly 4 lines of code per file.

Program 2:

|  |  |
| --- | --- |
| C:\Users\Unimp\AppData\Local\Microsoft\Windows\INetCacheContent.Word\fruits.jpg | C:\Users\Unimp\AppData\Local\Microsoft\Windows\INetCacheContent.Word\fruity_tootie.jpg |
| Original Image | After linear stretching in L |

Program 2 showed some interesting features, possible quirks that extend from the method used to convert from sRGB to Luv and back again (that is, sRGB 🡪 XYZ 🡪 Luv 🡪XYZ 🡪sRGB). After making the window centered on the image, I found that it made the entire image a little brighter.

Program 3:

|  |  |
| --- | --- |
| C:\Users\Unimp\AppData\Local\Microsoft\Windows\INetCacheContent.Word\fruits.jpg | C:\Users\Unimp\AppData\Local\Microsoft\Windows\INetCacheContent.Word\fruity_tootie.jpg |
| Original Image | After Histogram equalization in Luv |

As expected, histogram equalization resulted in an image with higher contrast. By comparison to the first image, the output of a histogram balanced by a small window centered at the middle of the image reveals a great deal more detail that was hidden by the shadows. The lemon appears translucent, showing the color and detail of the rind in which its flesh sits. The image is also flattened in some areas, as with the lime towards the back.

Program 4:

|  |  |
| --- | --- |
| C:\Users\Unimp\AppData\Local\Microsoft\Windows\INetCacheContent.Word\fruits.jpg | C:\Users\Unimp\AppData\Local\Microsoft\Windows\INetCacheContent.Word\fruity_tootie.jpg |
| Original Image |  |

The linear scaling function on a window focused at the center of the image in the xyY color space shows little difference between it and the original image, except that it is again slightly brighter.