ECE 4580 Digital Image Processing
Spring 2017
Final Project Report

Don’t Be So Negative

Analysis of YCBCR Color Schemes on Creating Retro Negative Images

Author: Seth Balodi

Abstract: Digital Image Processing has come a long way. The evolution of imaging and photography throughout history is fascinating, and I am lucky enough to be part of a generation that grew up when developing film was still something that had to be done. Because of this, I thought about something cool I could do for my final project in for this course and decided I wanted to take it completely old school. I wanted to make a function in the MATLAB Image Processing Toolbox that would allow me to take any image and make a negative of it to harken back to the early days of photography. The project will consist of taking a black and white image (sticking with the old school theme) and colorizing it. I think that this is a fascinating thing- it’s easy to understand how to grayscale an image, but putting color back in is something interesting. I would then take the re-colored grayscale image and take the negative of it, similarly to the process employed in the intermediate steps of photo development.

I. PROCEDURE

In this project I will split up the work into two big sections, each of which I will tackle separately. I may even potentially create different functions to help make the code more readable. The first task will be to take a grayscale image and use another image as a ‘color palette’ that will colorize the original grayscale image. This could make for a really cool looking image that would have a very interesting color palette depending on what images are chosen. This will be accomplished by reading in two images and then converting them to YCBCR color space. I will proceed to maximize and minimize pixels and then normalizing them. After doing a comparison of the individual pixel’s luminance, I can convert the grayscale image to a colored one.

From here, I would do the second task, which is to create a negative image from the resulting image from the previous step. This would be accomplished by getting all of the rgb channels and subtracting the masked pixels from each of them and creating null arrays. I would then combine all of the subtracted channels to form a negative of the color image.

II. PART I (COLOR PALETTE)

The first of the assignment deals with creating a color palette and a grayscale image and using a technique of luminance comparison, taking the colors form the color palette image and “painting” the grayscale image with the proper color values. This involves many steps, including processing on the colored and grayscale image before iterating through their size to determine luminance values. This procedure requires shifting the color space from a basic RGB to a YCBCR scheme. Understanding of this color space will allow for better understanding of the actual algorithm needed to produce the “painted image”. YCBCR is a family of color spaces that is used primarily in video and digital photography system. The “Y” in the acronym is the luma component and the “CB” and the “CR” are the blue and red differences in the chroma components. Luma represents the brightness in an image and it represents the achromatic part of an image. Chroma is used in video systems to convey color information, which is separate to the Luma but used together to describe digital images and videos. The YCBCR color scheme takes in the the Y’, which is the nonlinear light intensity encoded from the gamma corrected RGB primaries.

YCBCR is related to other color schemes but provides a more “intelligent” distribution of color for images to be designed based on the luminance and chroma. An illustration can be found below in Illustration 1:

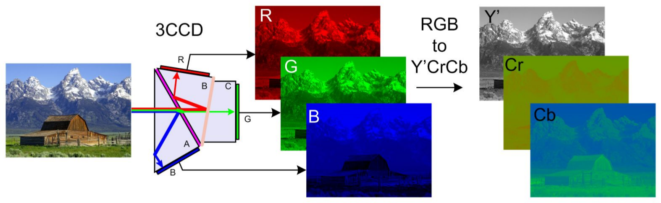


Illustration 1: RGB compared YCBCR

An actual example of how the color schemes looks on a sample image is shown in Illustration 2:

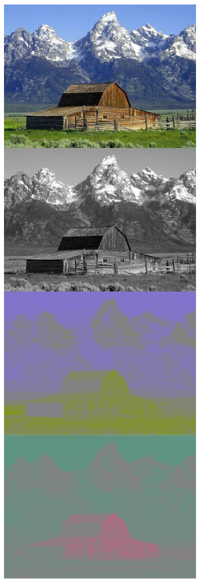
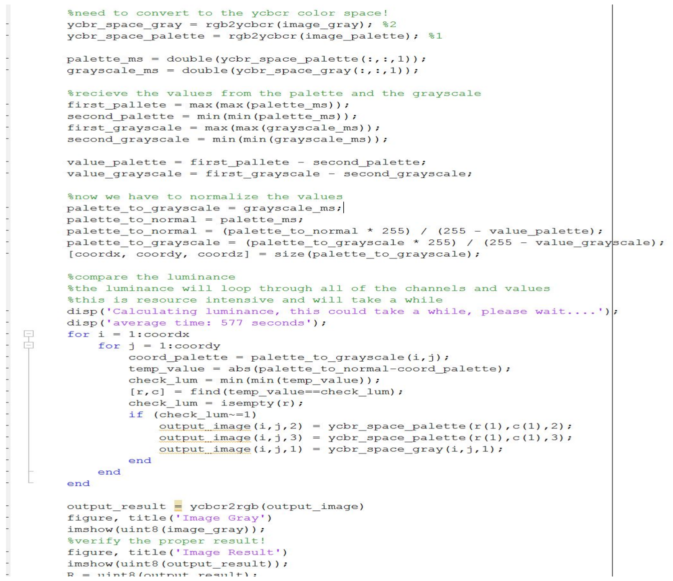


Illustration 2 : YCBCR Color Space

The YCBCR The code shown in figure 1 shows this algorithm for calculating the nonlinear light intensity values.



```
%need to convert to the ycbcr color space!
ycbr_space_gray = rgb2ycbcr(image_gray); %2
ycbr_space_palette = rgb2ycbcr(image_palette); %1

palette_ms = double(ycbr_space_palette(:,:,1));
grayscale_ms = double(ycbr_space_gray(:,:,1));

%recieve the values from the palette and the grayscale
first_pallete = max(max(palette_ms));
second_palette = min(min(palette_ms));
first_grayscale = max(max(grayscale_ms));
second_grayscale = min(min(grayscale_ms));

value_palette = first_pallete - second_palette;
value_grayscale = first_grayscale - second_grayscale;

%now we have to normalize the values
palette_to_grayscale = grayscale_ms;
palette_to_normal = palette_ms;
palette_to_normal = (palette_to_normal * 255) / (255 - value_palette);
palette_to_grayscale = (palette_to_grayscale * 255) / (255 - value_grayscale);
[coordx, coordy, coordz] = size(palette_to_grayscale);

%compare the luminance
%the luminance will loop through all of the channels and values
%this is resource intensive and will take a while
disp('Calculating luminance, this could take a while, please wait....');
disp('average time: 577 seconds');
for i = 1:coordx
    for j = 1:coordy
        coord_palette = palette_to_grayscale(i,j);
        temp_value = abs(palette_to_normal-coord_palette);
        check_lum = min(min(temp_value));
        [r,c] = find(temp_value==check_lum);
        check_lum = isempty(r);
        if (check_lum~=1)
            output_image(i,j,2) = ycbr_space_palette(r(1),c(1),2);
            output_image(i,j,3) = ycbr_space_palette(r(1),c(1),3);
            output_image(i,j,1) = ycbr_space_gray(i,j,1);
        end
    end
end

output_result = ycbcr2rgb(output_image)
figure, title('Image Gray')
imshow(uint8(image_gray));
%verify the proper result!
figure, title('Image Result')
imshow(uint8(output_result));
R = uint8(output_result);
```

Figure 1

After having checks in place to ensure that the images that are passed into the script are valid and can be used within the program, the size of each image is measured and then converted into the YCBCR color scheme using the rgb2ycbcr() function from the IPT kit in MATLAB. After converting both of them into the new space, we go through and minimize and maximize the values of the YCBCR color space variables based on the specific channels shown in figure 1. The values must then be normalized by multiplying the size of the YCBCR by the RGB 255 and then dividing it by the difference of the maximized and minimized values. The normalized results are fed into the size variables mx, my, mz.

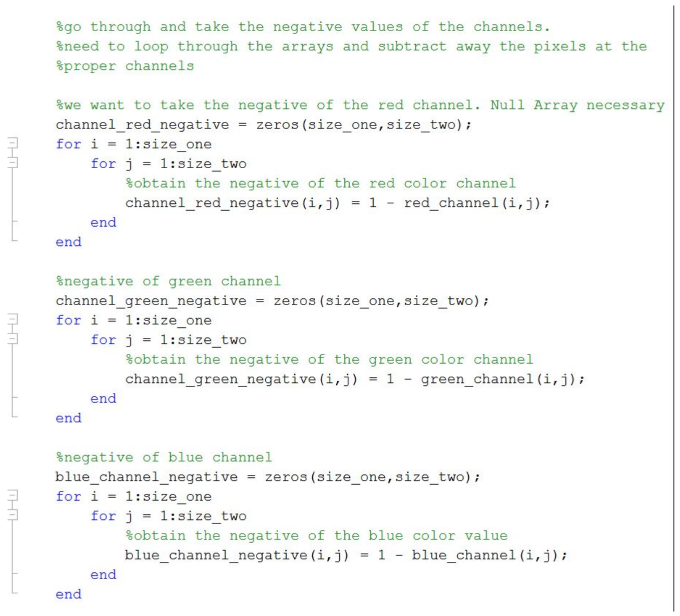
Finally, to complete the first part of the project, we need to do the luminance analysis with the Luma value elaborated upon earlier. This process is incredibly processor intensive and can take a long time as it has to iterate through the entire size of the color space and compare the pixels values to temporary values created on previous iterations. Going through all of the pixels in the images, a new color space is created with the proper luminance values to be read over to the other image. The result is then thrust back into the RGB color space and shown in a figure that can be detailed and outlined in the experimental results section.

A. Other Recommendations

Creating a more efficient algorithm is important for the luminance part of the program is important is the average runtime of the Luma comparison is 577 seconds and can go up to 600 seconds.

III. PART II (NEGATIVE)

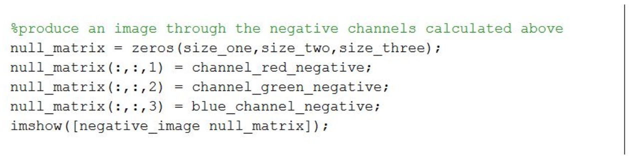
For the second part, we have to create a negative image from the image created from the previous section, which effectively “painted” a grayscale image from a color palette provided by a colored image. The YCBCR color space allowed for an intelligent distribution of color to create images that can look very striking if proper colors are used. From here, we shall simulate the early stages of photography throughout history and create a negative of the image we made in the previous section. We will use this as a simulation as to how photographs were developed in the past in lightrooms. The entire concept behind negative images is taking the color channels of the image and iterating through them, creating null arrays which will be filled by the processed channel signals. The processing is a simple subtraction from the value of 1. We load this into the iteration variables as pixel values and coordinates. An RGB space is used for this so the channels are the red, green and blue channels of the image, which can be taken by the MATLAB matrix functions or a simple value selection. The Process is outlined in figure 2 below:



```
%go through and take the negative values of the channels.
%need to loop through the arrays and subtract away the pixels at the
%proper channels

%we want to take the negative of the red channel. Null Array necessary
channel_red_negative = zeros(size_one,size_two);
for i = 1:size_one
    for j = 1:size_two
        %obtain the negative of the red color channel
        channel_red_negative(i,j) = 1 - red_channel(i,j);
    end
end

%negative of green channel
channel_green_negative = zeros(size_one,size_two);
for i = 1:size_one
    for j = 1:size_two
        %obtain the negative of the green color channel
        channel_green_negative(i,j) = 1 - green_channel(i,j);
    end
end

%negative of blue channel
blue_channel_negative = zeros(size_one,size_two);
for i = 1:size_one
    for j = 1:size_two
        %obtain the negative of the blue color value
        blue_channel_negative(i,j) = 1 - blue_channel(i,j);
    end
end
```

Figure 2

We finally combine all of the new channel values (all of which are negative). Combining all of the values created by the "negative" channels forms a negative image, which can be shown next to the original image passed in which is shown in the code from figure 3 below:



```
%produce an image through the negative channels calculated above
null_matrix = zeros(size_one,size_two,size_three);
null_matrix(:,:,1) = channel_red_negative;
null_matrix(:,:,2) = channel_green_negative;
null_matrix(:,:,3) = blue_channel_negative;
imshow([negative_image null_matrix]);
```

Figure 3

IV. PART III (EXPERIMENTAL RESULTS)

The experimental results from the process described in the report are put in order below with simple descriptions of the code.

Firstly, the two input images are shown below, one of which is a grayscale and the other one is the color palette. These are both seen respectively in Figure 4 and Figure 5:

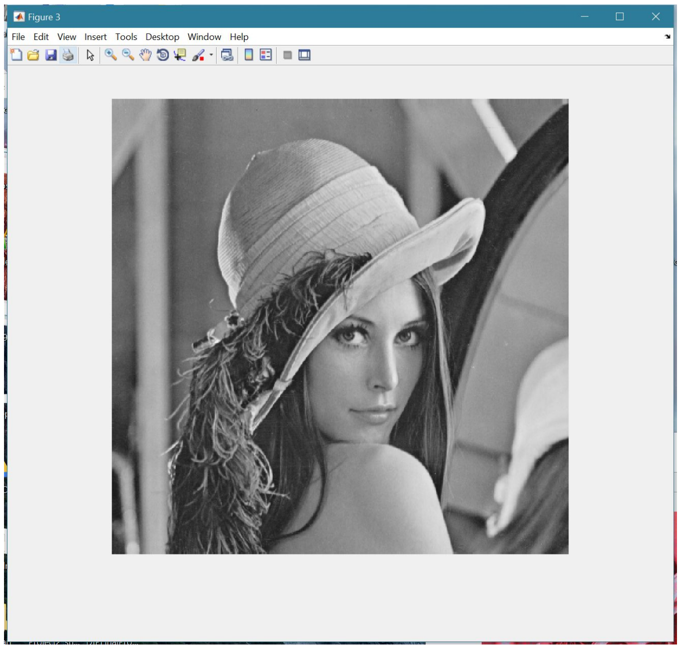


Figure 4 : Input One Grayscale

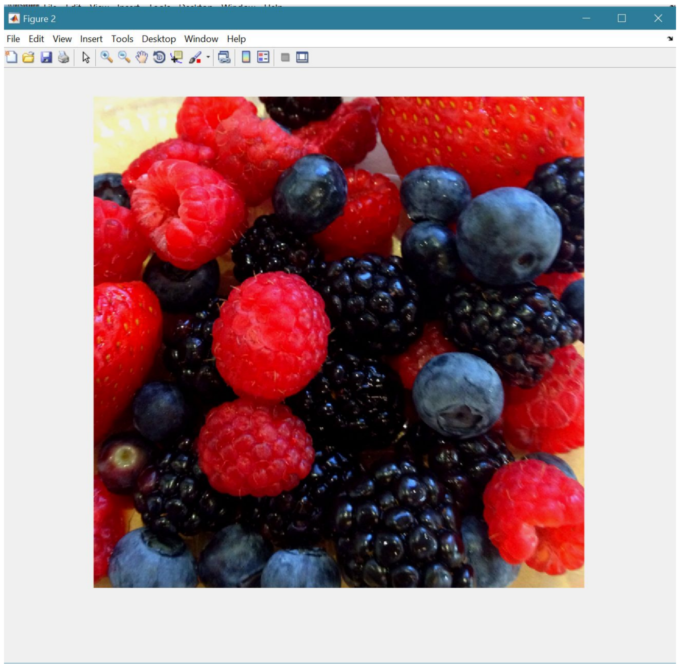


Figure 5: Input Two Colored Image (Palette)

Taking in these image values is the starting point as they are then processed into the YCBCR color space and their sizes are recorded into various variables to allow for iteration through their pixels or access to their particular locations or values. After calculating the luminance values, we obtain a "painted image" shown below in Figure 6. This picture shows the red and blue, and even black values from the palette being moved into the image. Many other images can be used, for much bigger effect, but I used these images to that it would be easy to see the colors come in and the negatives to have a human shape which would be easily recognizable.

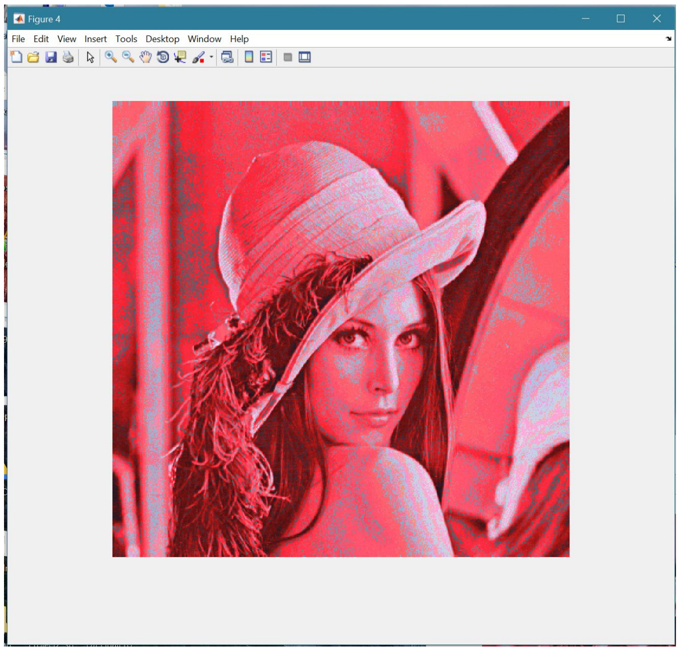


Figure 6 : Output Painted Image Based on Luma Values

After running the negative channels, we obtain the following image in Figure 7. We can compare this negative to the "painted" image.

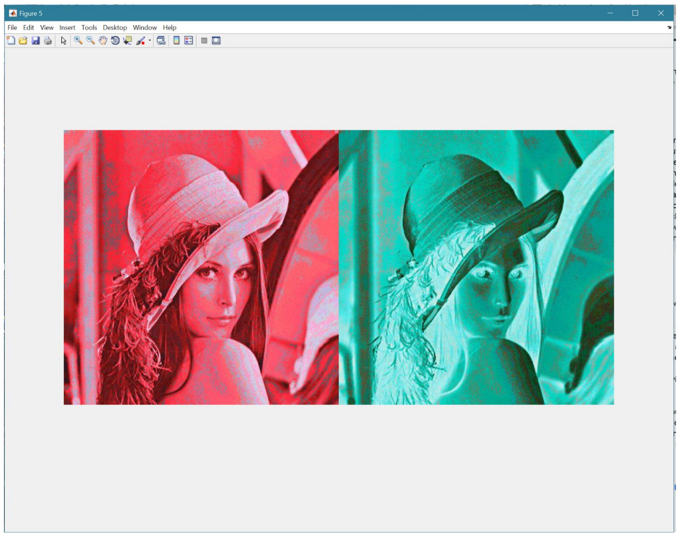


Figure 7 : Output Result of Painted Negative Image

V. CONCLUSION

In conclusion, this project was an exercise in using a different color space that I had not used previously and learning about how it works as opposed to the RGB values. The Luma and Chroma values were big in creating a properly painted image and the process of creating the negative was a straightforward implementation of a very interesting piece of photography history. Taking values from the different channels through iterating the pixels was trivial compared to the Luma comparison and analysis in part 1. The most interesting part of all of this was realizing how much processing power it took to do this iteration, which made me appreciate how cutting edge graphics and image processing is in computer science. This I found very interesting as it took me 10 minutes, or longer, just to get through the luminance iteration of the program. I added the "tic/toc" analysis methods of MATLAB to generate the time efficiency of the process and found it interesting to run it a couple times and see how it went. The average was 577 seconds, with the highest recorded instance that I tried being about 610 seconds.

For the images chosen, many other images can be used, for much bigger effect, but I used these images to that it would be easy to see the colors come in and the negatives to have a human shape which would be easily recognizable. Nature scenes in grayscale work very well with other colored nature scenes, and those provide the best looking "painted images", but for this project, it was important for the negative and the painted image to be easily recognizable and not seem too "trippy".

Acknowledgment

This report came about from the Final Project of Digital Image Processing (ECE 4580) at Virginia Tech's Department of Electrical and Computer Engineering.

References and Footnotes

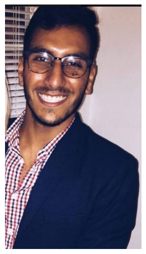
References: MATLAB Documentation (for reference images)

A. Copyright Form

citation and references to this original document and the original author.

VI. IEEE Publishing Policy

This document is free to be published with credit given to the author and citations required for all information and code provided.



Seth Balodi, *Author*
Virginia Tech, Class of 2017
Computer Engineering