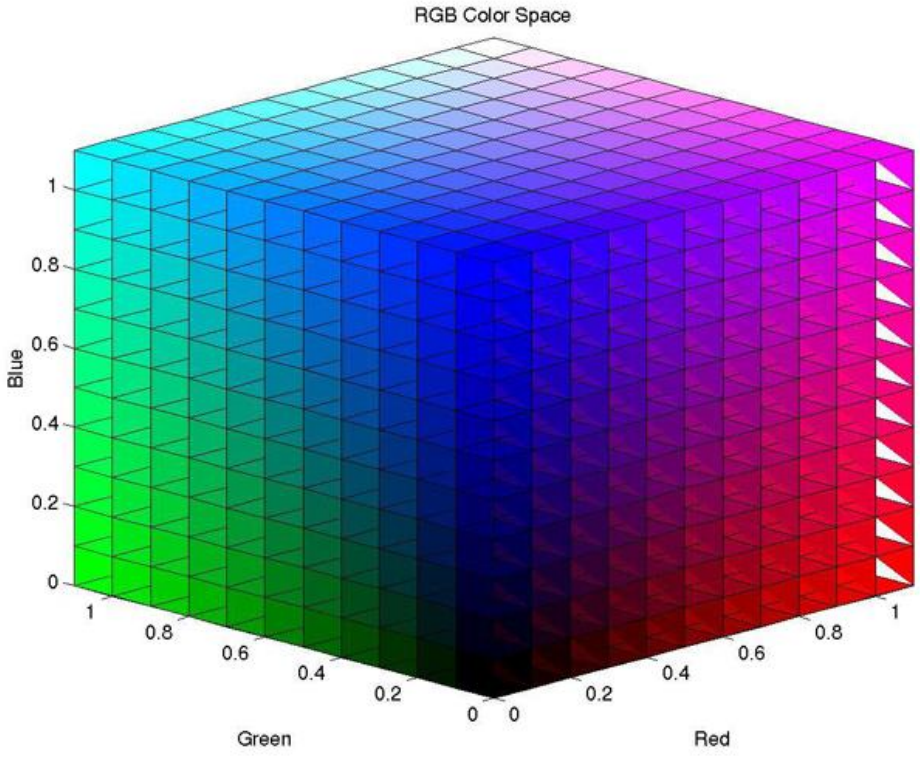
Assignment #2 – Color spaces

Faculty of Engineering Science
Dept. of Electrical and Computer Engineering
Introduction to Digital Image Processing, 361-1-4751

Maor Assayag 318550746

Refahel Shetrit 204654891



1 RGB and Grayscale

In this exercise we will examine the relation between the grayscale and RGB image.

1.1 Reading the Image

Read a color image in MATLAB using the *'imread()'* function. Convert the image to double and normalize to the range [0,1].

Explanation: We simply read the image and divided it by 255 (using double) to transform all the data to values in range of [0, 1].

1.2 Display the image

Display the image using MATLAB's *'imshow()'* function.

Figure 1.2

1.3 RGB Channels

Extract the separate R, G and B channels from the image and display them.

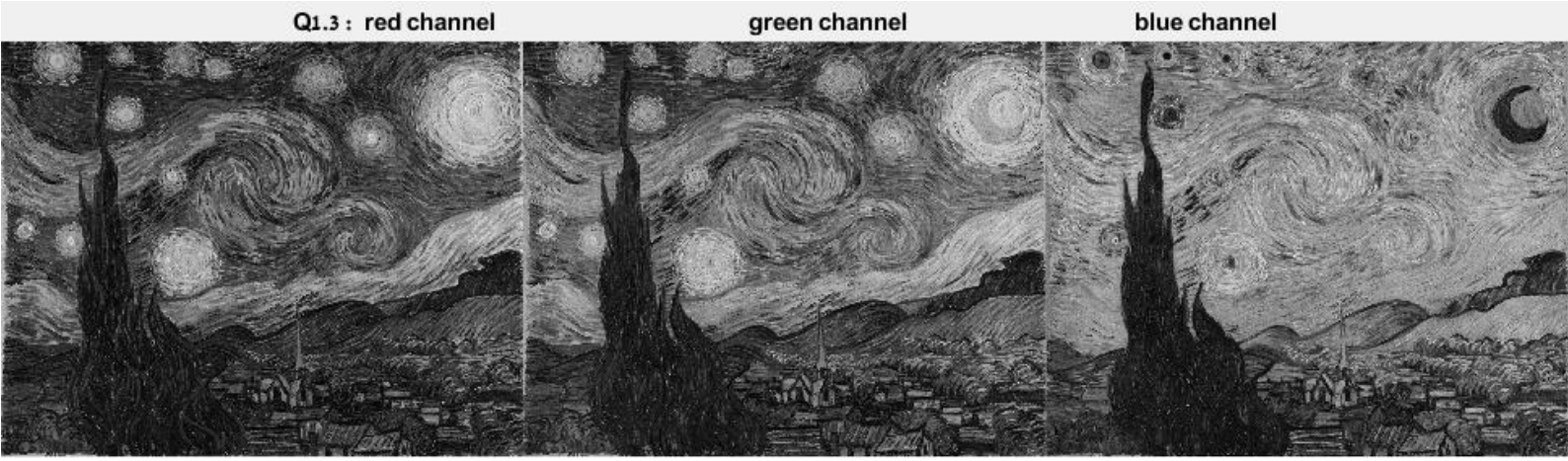


Figure 1.3.1

Explanation : In MATLAB we can simply extract the RGB channels from the image. We can notice that in the blue channel for example the blue areas in the original image have values closer to 1 (equivalent to white pixels in Grayscale).

1.4 RGB to Grayscale

Write your own function that converts an RGB image to grayscale 'dip rgb2gray()'.
(a) Gray = 0.2989 * R + 0.5870 * G + 0.1140 * B

Explanation: After extracting each RGB channel, we construct 1 matrix from this formula to represent the image in Grayscale values.

1.5 Comparing to MATLAB function

Compare your results to MATLABs *'rgb2gray()'*.

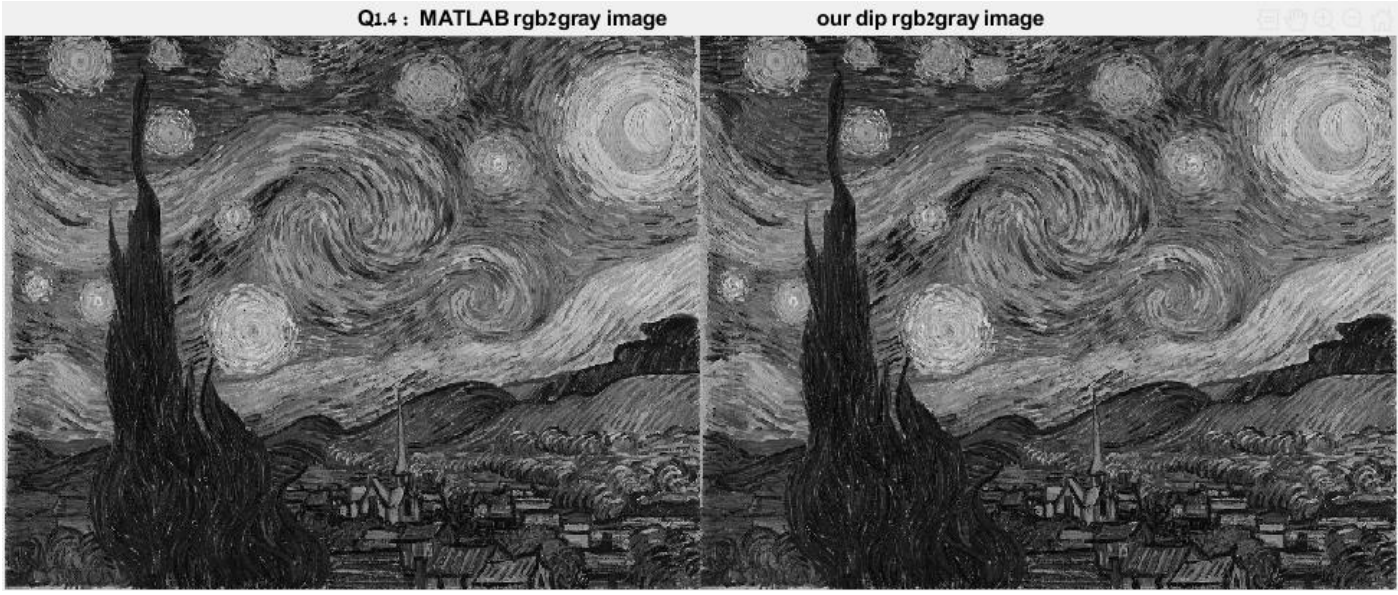


Figure 1.5.1

```
%Q1.5
%test if Q1.4  - dip_rgb2gray is accurate
Q1_4_image_test = round(gray_img - grey_scale_img,3);
if (mean(Q1_4_image_test) == 0)
    disp('Q1.4 is accurate');
else
    disp('Q1.4 is not accurate');
end
```

Figure 1.5.2

Explanation: We can see that our Grayscale conversion is like the build-in MATLAB *rgb2gray()* function. In addition, we added a mean test to test the subtract Grayscale images.

1.6 Manipulation

Manipulate the values and the order of the channels and display the grayscale image. Show 3 different manipulation. Explain the manipulations you chose and their effects in the report. You may choose to explain on only 2 manipulations in your report. For example:
(a) Add a constant to a single channel.
(b) Switch between the channels.
(c) Apply some function on a specific channel. (Make sure that the output of the function stays in the range [0,1])

1.6.1 Add a constant to a single channel.

Figure 1.6.1

Explanation : Adding a positive constant to an image will brighter it. As the results are shown, the image got brighter (the blue color is lighter and the easy one to see the difference on). In addition, the grayscale got brighter as expected.

1.6.2 Switch between the channels.

Figure 1.6.2

Explanation: We choose to switch between the blue and the green channels. The color image is now having a green tone where it was blue and vice versa. In the grayscale image we can see the impact of changing the dominant color. Previous it was the dark blue, hence the grayscale was darker – but now the light green of the moon is dominant and the over whole image looks brighter.

1.6.3 Apply some function on a specific channel

Figure 1.6.3

Explanation: We choose the following function : $F(x) = 0.6x - 0.1$. Overall, this function will decrease each pixel value (rang [0, 1] which will logically transform each color to its darker tones. As expected, we can see that the image got darker as well the grayscale image.

2 Additive vs Subtractive Color space

In this exercise we will examine the difference between additive and subtractive color spaces. We will examine the visual properties of the CYMK colorspace.

2.1 CYMK

Briefly explain about the CYMK color space.

Explanation: Colors can be created with color spaces based on the CMYK color model, using the subtractive primary colors of pigment (cyan (C), magenta (M), yellow (Y), and black (K)). To create a three-dimensional representation of a given color space, we can assign the amount of magenta color to the representation's X axis, the amount of cyan to its Y axis, and the amount of yellow to its Z axis. The resulting 3-D space provides a unique position for every possible color that can be created by combining those three pigments.

2.2 CYMK Channels

Create the CYMK channels from the RGB channels and display each separately:
(a) Black = minimum(1-Red,1-Green,1-Blue) (b) Cyan = (1-Red-Black)/(1-Black)
(c) Magenta = (1-Green-Black)/(1-Black) (d) Yellow = (1-Blue-Black)/(1-Black)

2.2.1 Black Channel

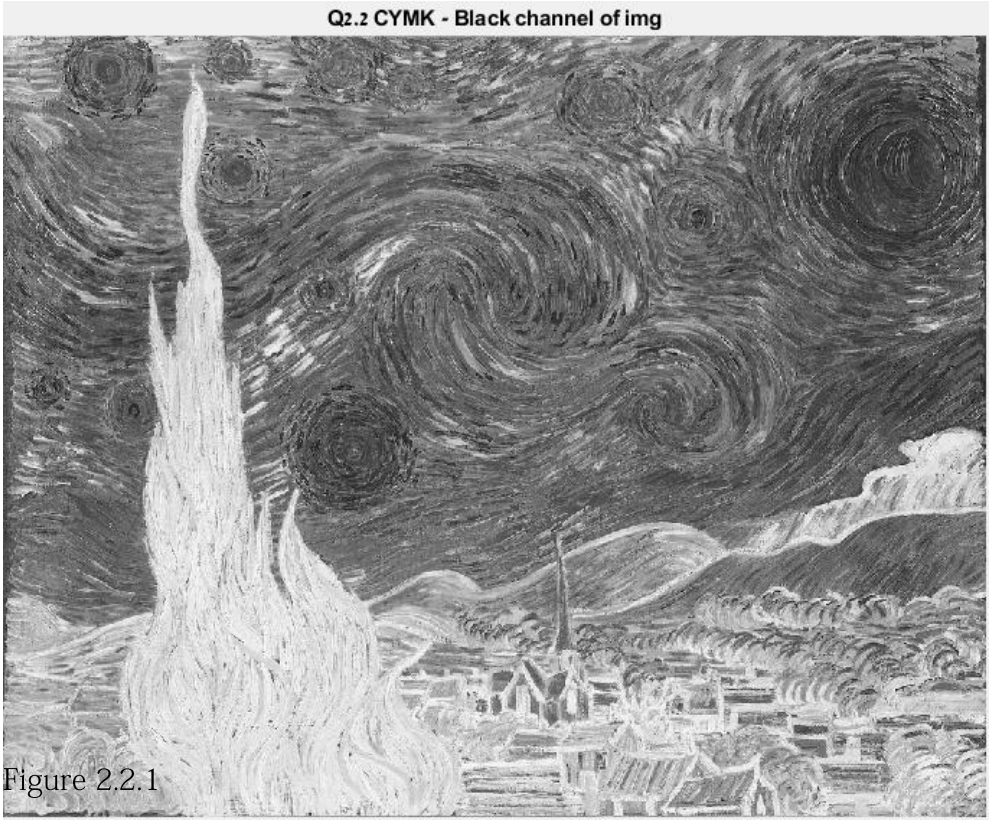


Figure 2.2.1

2.2.2 Cyan Channel

Figure 2.2.2

2.2.3 Magenta Channel

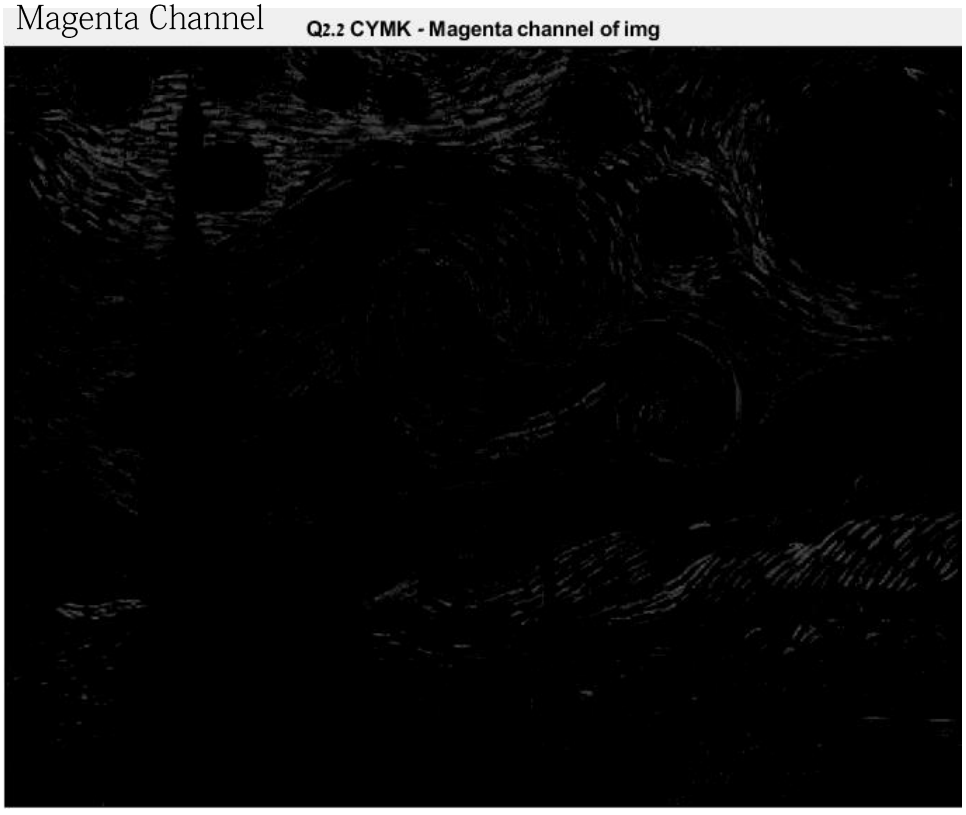


Figure 2.2.3

2.2.4 Yellow Channel

Figure 2.2.4

Explanation: Each channel displays us each color areas correspond to the color (=brighter pixels in the grayscale images).

2.3 Display CYMK Channels

Display the separate channels using the provided *'displayCYMK()'* function.

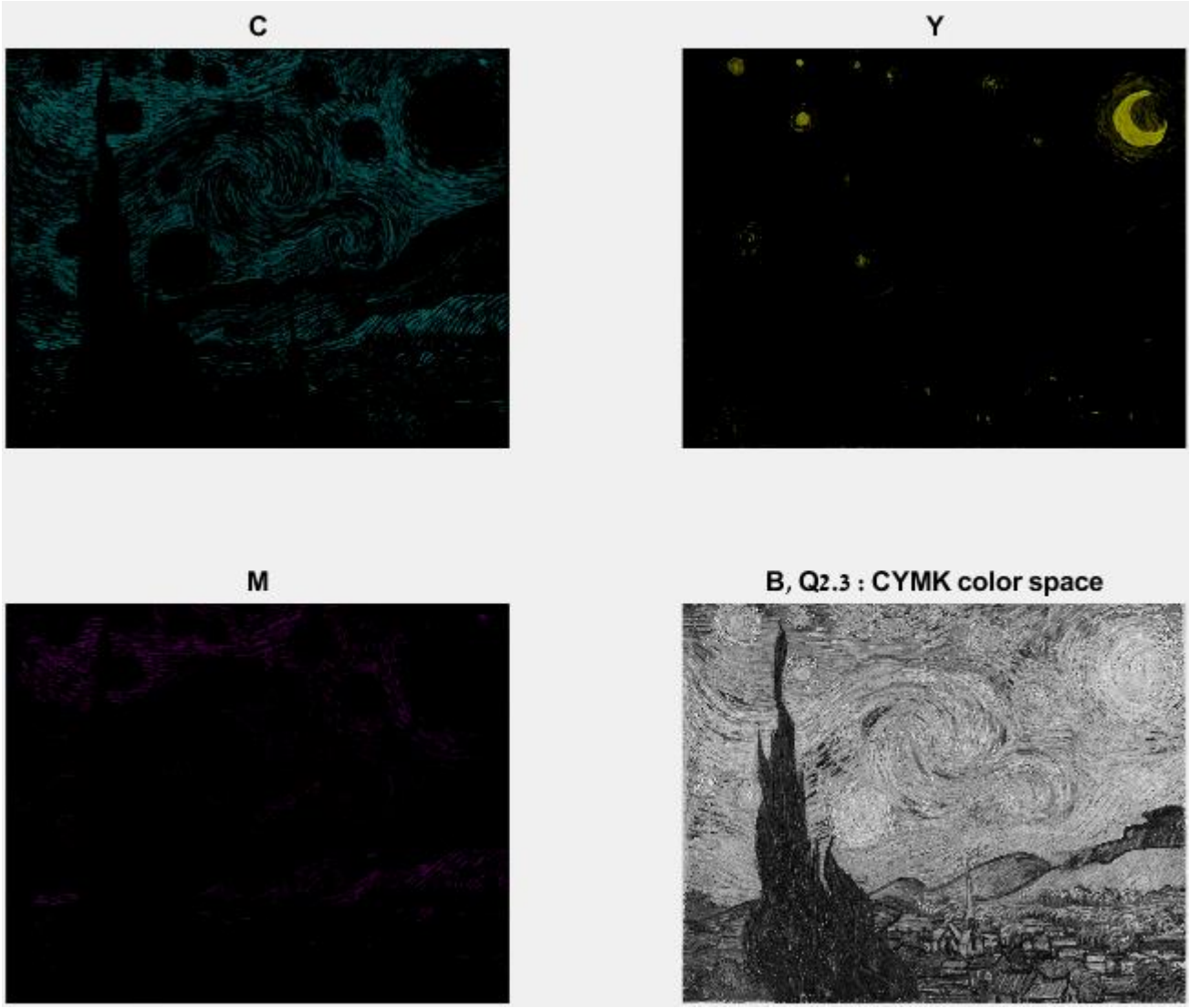


Figure 2.3

2.4 CYMK Manipulation

Repeat step 6 from section 1. Display the results using the provided 'imshowCYMK()' function.

2.4.1 Add a constant to a single channel.

Figure 2.4.1 – *imshowCYMK()*

Explanation : When we convert CYMK to RGB, the only channel that use the Magenta channel data is the Green channel : $G = 255 \times (1-M) \times (1-K)$. In this manipulation we greatly increase the Magenta channel, hence we decrease the green channel – which means each area that has a dominant green color is going to be darker. The results shows us that the indirectly decreasing the green channel effect the whole image and make it darker.

2.4.2 Switch between the channels.

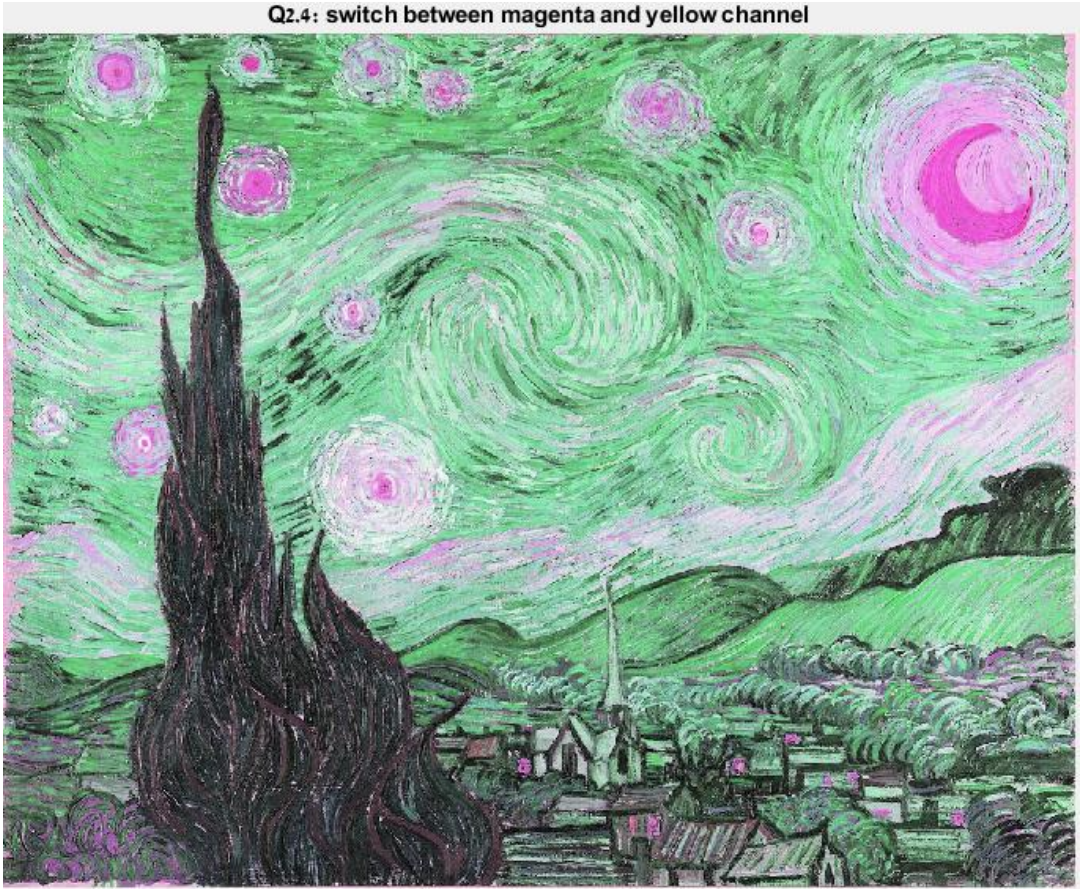


Figure 2.4.2 - *imshowCYMK()*

Explanation : When we convert CYMK to RGB the following channels are affected by this change : $G = 255 \times (1-M) \times (1-K)$

$$B = 255 \times (1-Y) \times (1-K)$$

Basically, this change switching exactly the Green and Blue channel in the RGB color space.

2.4.3 Apply some function on a specific channel

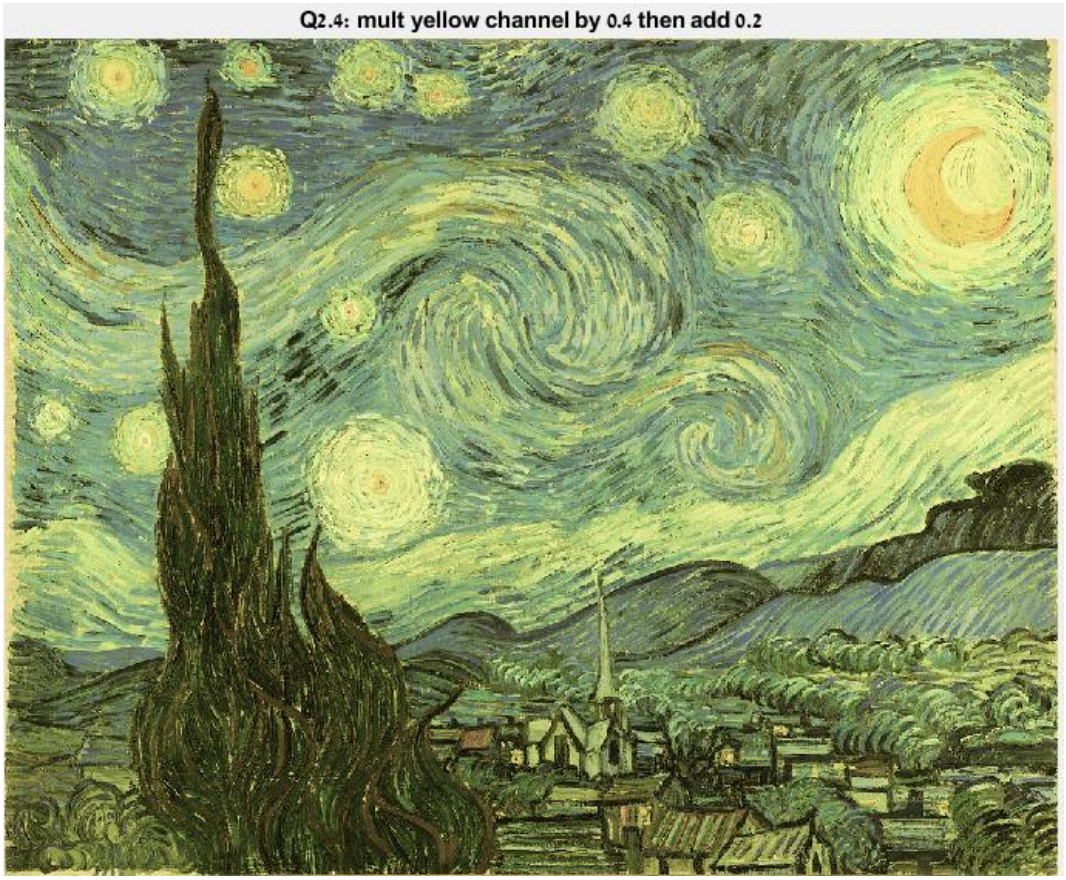


Figure 2.4.3 - *imshowCYMK()*

Explanation: We choose the following function : $F(x) = 0.4x + 0.1$. This function will decrease the values that closer to 1 (brighter pixels) and increase the values that closer to 0 (darker pixels) of the Yellow channels, which effects the Blue channel in the RGB color space ($B = 255 \times (1-Y) \times (1-K)$). In our case, we can see that the areas when the Blue is dominant got a lighter tone, which means that in the original image the Blues has darker tone and values close to 0.

3 HSV colorspace

In this exercise we will examine the difference between the HSV and RGB representations. We will see the effects of changing the values of separate channels.

3.1 HSV

Briefly explain about the HSV color space.

Explanation: Unlike RGB and CMYK, which are defined in relation to primary colors, HSV is defined in a way that is like how humans perceive color. It's based on three values: hue, saturation, and value. This color space describes colors (hue or tint) in terms of their shade (saturation or amount of gray) and their brightness value. **Note** : more on page 29.

3.2 RGB to HSV

Write your own function that converts an RGB image to HSV *'dip rgb2hsv()'*.

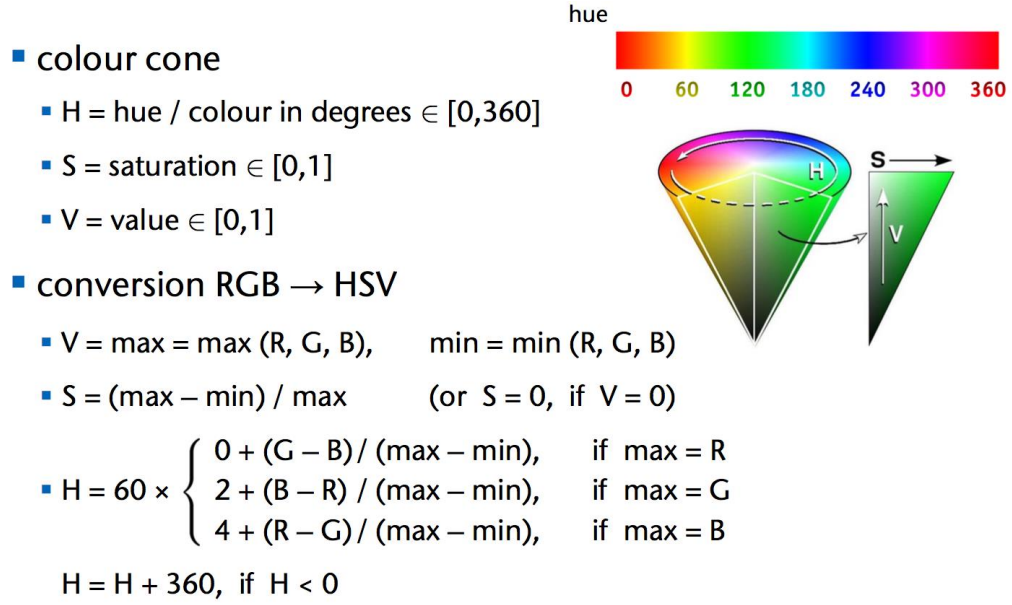


Figure 3.2

Explanation: We wrote our own conversion function based on those formulas. In addition, we normalized the H value to [0,1]. First we are extracting each RGB channels to a matrix. Than we iterate the columns and rows and for each pixel we determine the value of the [H, S, V] index in their separate matrix's.

3.3 Display HSV Channels

Convert the image to HSV and display the separate channels

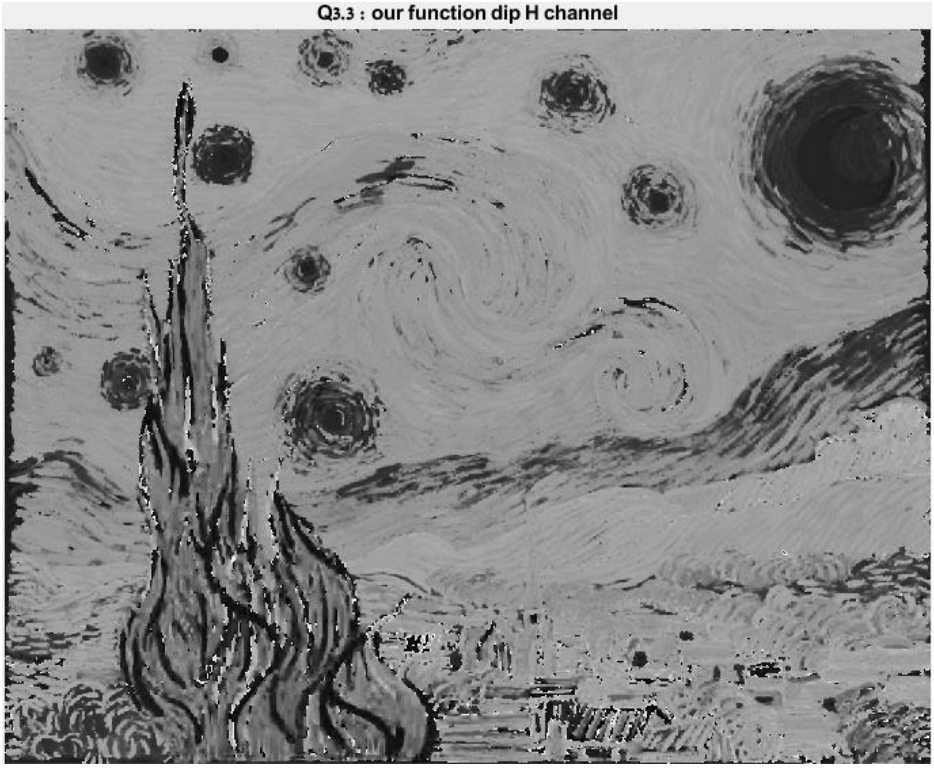


Figure 3.3.1 – H channel

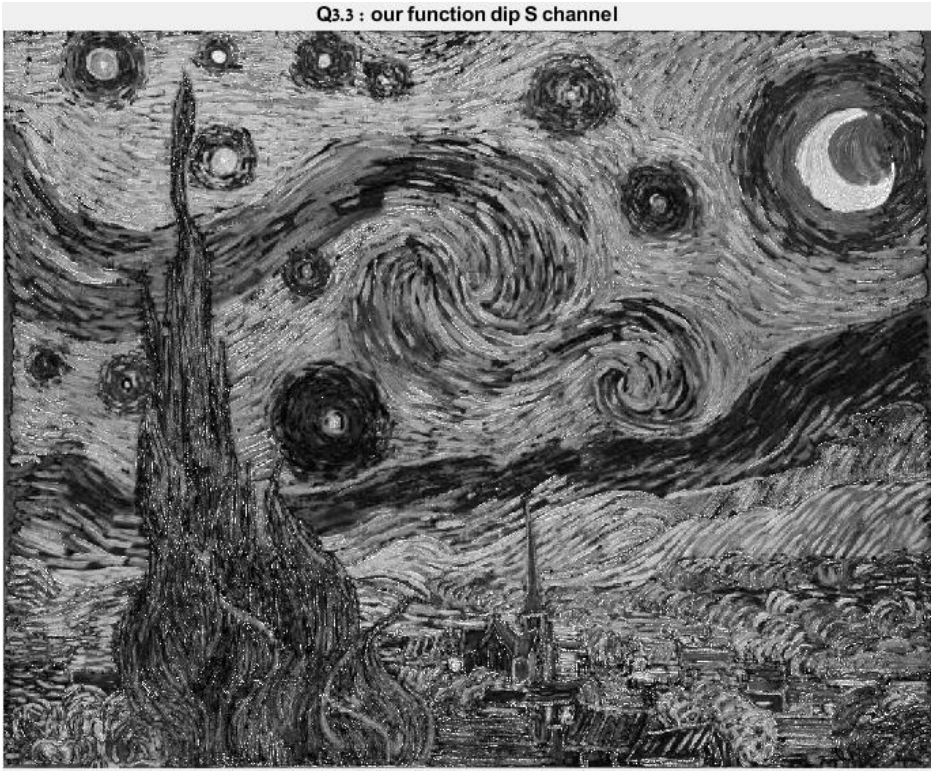


Figure 3.3.2 – S channel

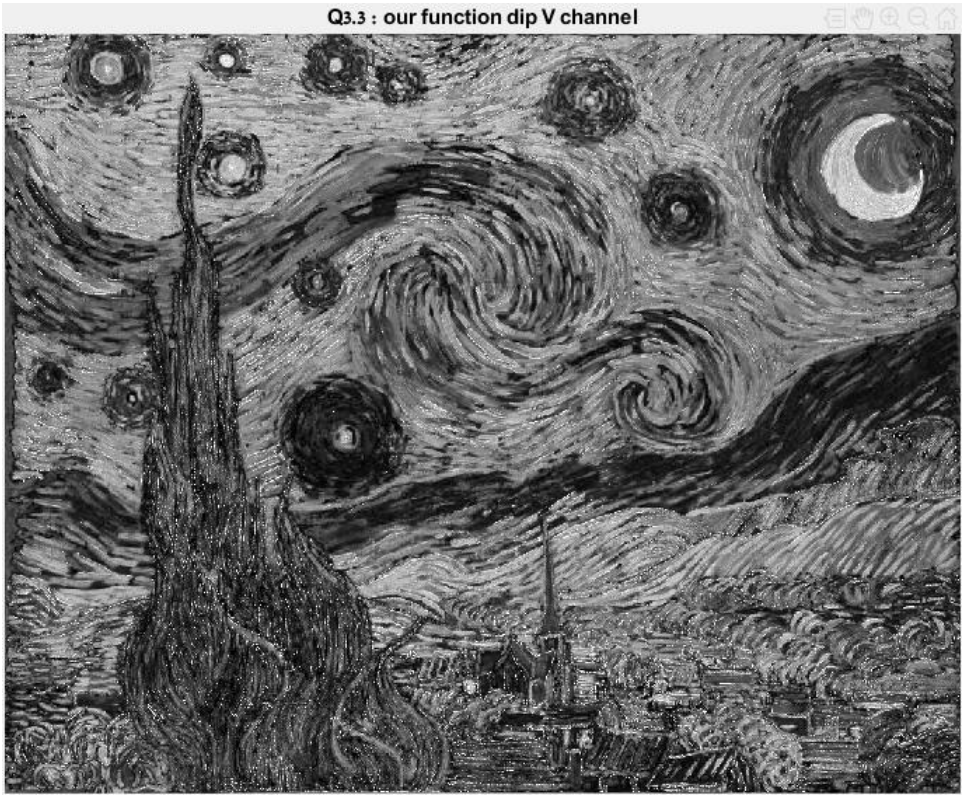


Figure 3.3.3 – V channel

3.4 Comparing to MATLAB function

Compare your results to MATLAB's 'rgb2hsv()' function.

Figure 3.4.1 – MATLAB HSV to RGB

Figure 3.4.2 – Our HSV to RGB

```
% Q3.4 Compare your results to MATLAB's 'rgb2hsv()' function.
Q3_hsv_test_h = round(img_h - img_dip_h,10);
Q3_hsv_test_s = round(img_s - img_dip_s,10);
Q3_hsv_test_v= round(img_v - img_dip_v,10);
mean_h = mean(Q3_hsv_test_h,'all');
mean_s = mean(Q3_hsv_test_s,'all');
mean_v = mean(Q3_hsv_test_v,'all');
if (mean_h ==0 && mean_s ==0 && mean_v ==0)
    disp('Q3.4 : dip_rgb2hsv is accurate');
else
    disp('Q3.4 : dip_rgb2hsv is not accurate');
end
```

Figure 3.4.3 – Accuracy test

Explanation : First we display the 2 results. We can't spot any difference between the build-in function and ours. Second we wrote an accuracy mean test in MATLAB and compare each pixel to the correspond pixel in our result, and successes.

3.5 HSV Manipulation

Repeat step 6 from section 1. Display the results using the provided *'imshowHSV()'* function.

3.5.1 Add a constant to a single channel.

Figure 3.5.1

Explanation : We decrease the Hue channel by 0.35. If we look at the Hue wheel we can see the following :

Hue

Hue is the color portion of the color model, expressed as a number from 0 to 360 degrees:

Color	Angle
Red	0–60
Yellow	60--20
Green	120–180
Cyan	180–240
Blue	240–300
Magenta	300–360

So by subtracting 0.35 in the range of [0,1] (-> 126 angles) we transform the Blue tones to Green tones, exactly as shown in the results.

3.5.2 Switch between the channels.

Figure 3.5.2

Explanation : Switching the S and V channels results each color to be bolder. It can be explained from the following :

Saturation

Saturation is the amount of gray in the color, from 0 to 100 percent. Reducing the saturation toward zero to introduce more gray produces a faded effect.

Sometimes, saturation is expressed in a range from just 0–1, where 0 is gray and 1 is a primary color.

Value (or Brightness)

Value works in conjunction with saturation and describes the brightness or intensity of the color, from 0–100 percent, where 0 is completely black, and 100 is the brightest and reveals the most color.

From this we know that in the original image there was almost non-gray colors, so the Saturation values was close to 1, therefor when making it the V channel it intensifies each color.

3.5.3 Apply some function on a specific channel

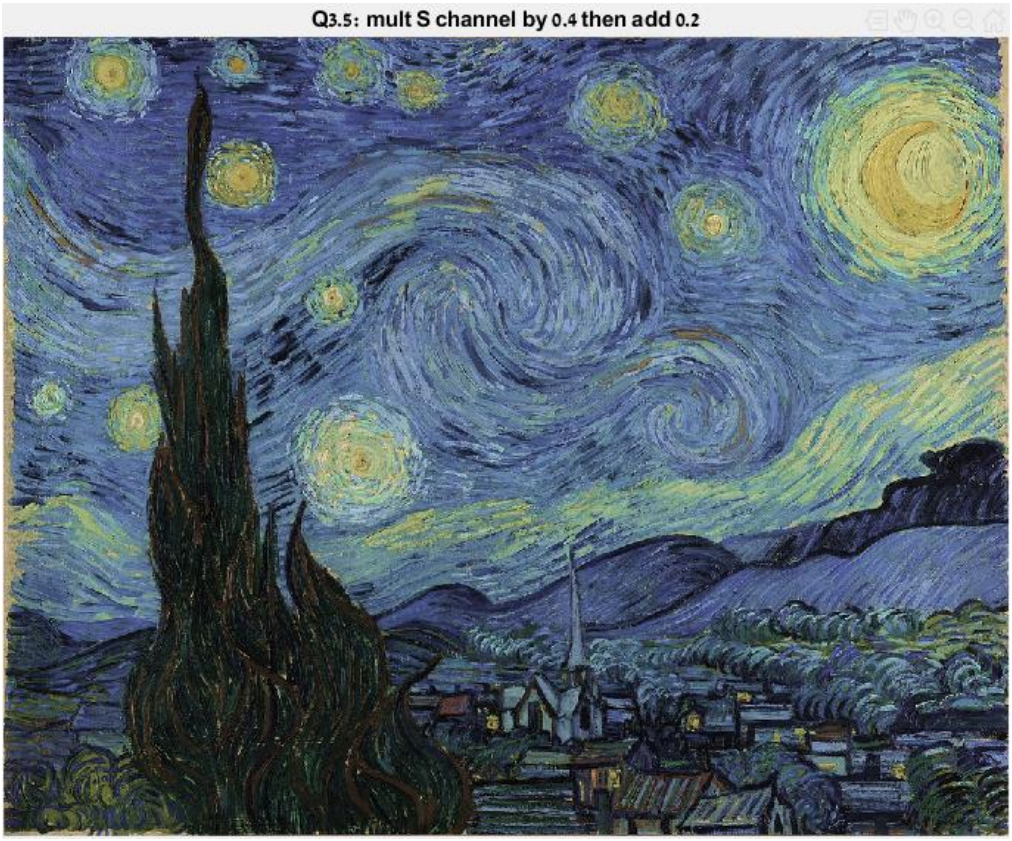


Figure 3.5.3

Explanation: After applying $F(x) = 0.4x + 0.2$ to the Saturation channel, its hard to spot a difference from the original image. Although, we can see that image got a little more faded effect, which suggest that the original S values was closer to 1, then the function decrease their value and introduce more gray in the image.

3.6 Switch RGB Channels -> HSV Channels

Switch the order of the RGB channels and then convert to HSV using *'dip_rgb2hsv()'*. Display the separate channels. Which channels changed? Why?

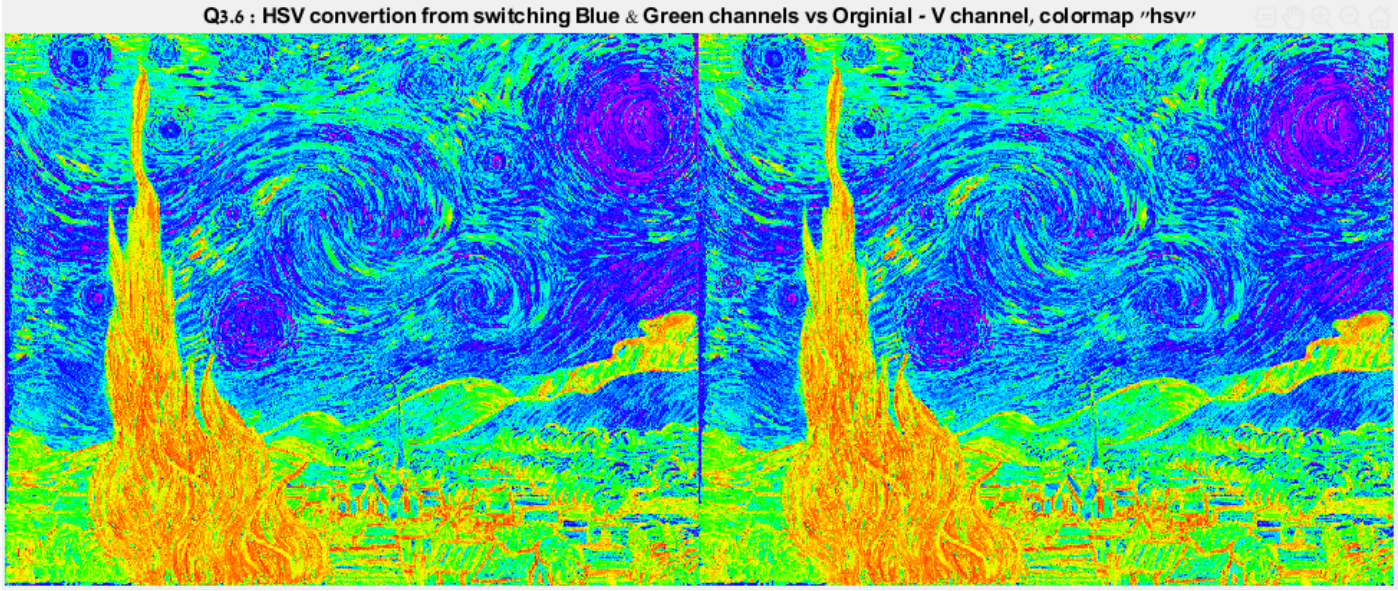


Figure 3.6.1

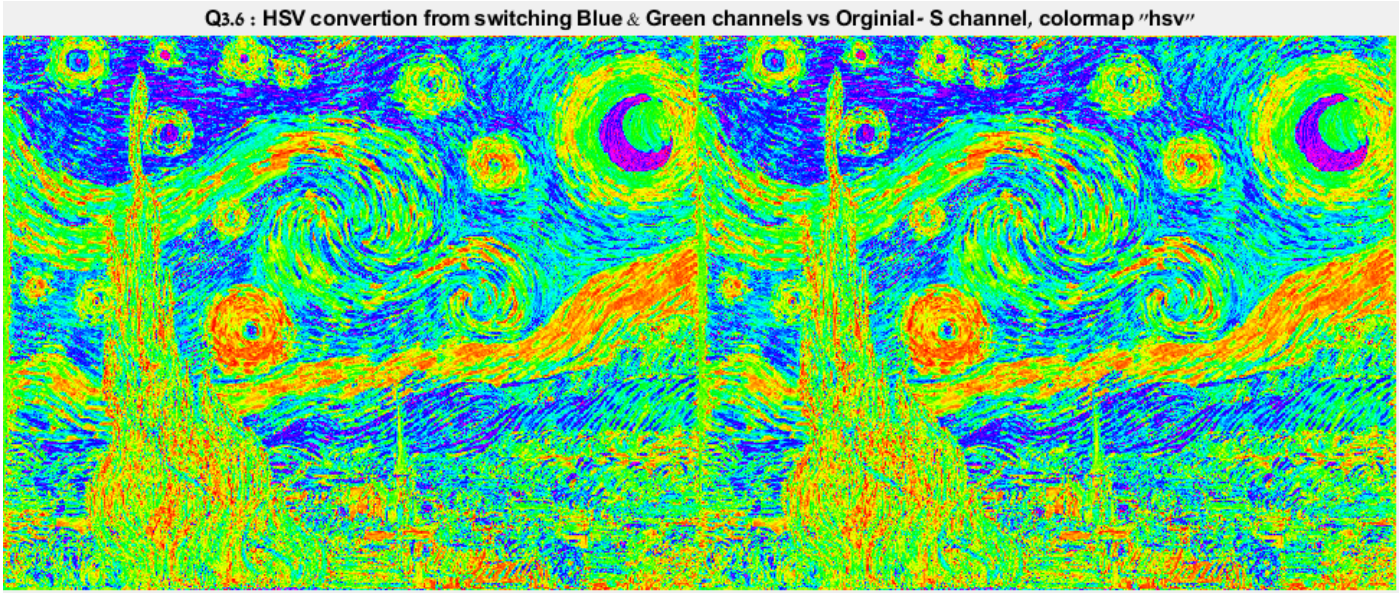


Figure 3.6.2

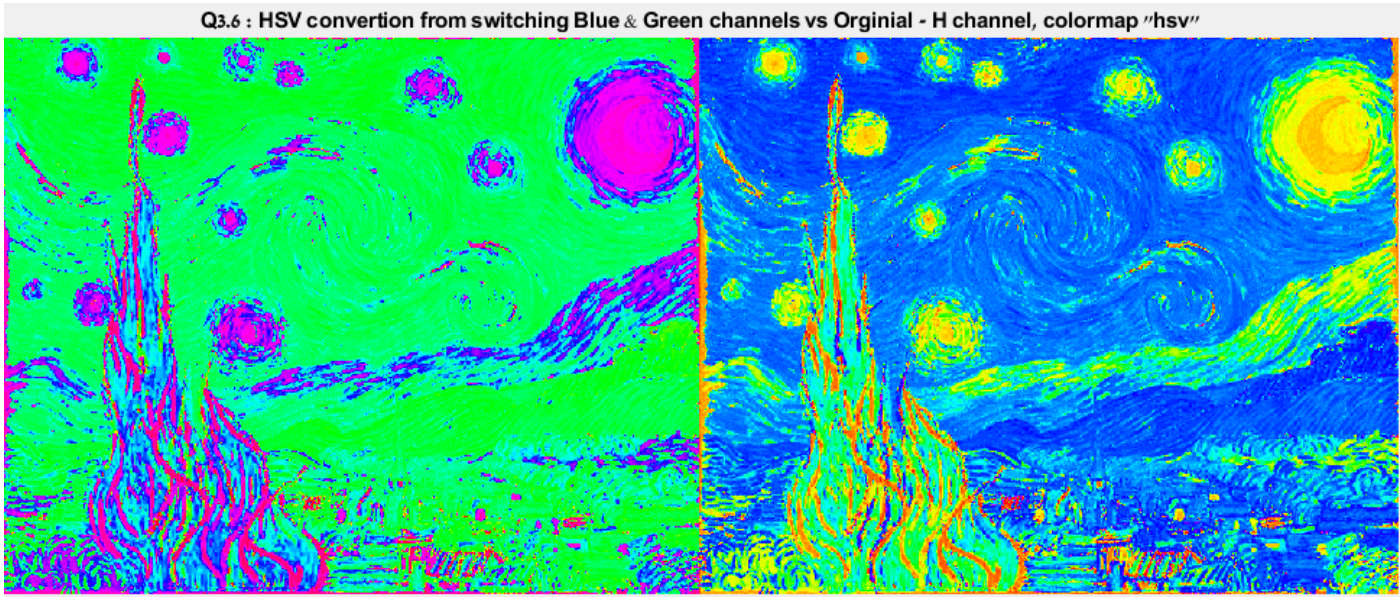


Figure 3.6.3

Explanation : We see that only the H channels has changed from switching the Blue and Green channels in the RGB color space. This is simply because the construction of S & V channels depends only on the max and min values of (R,G,B) at each pixel, which doesn't change if we only change the order of the operation [min (R, G, B) = min (R, B, G)]. In the other hand, the H values are greatly affected by which value is the max value (Red / Green / Blue channel). It's easy to see that the results reflect the switching in channels which also switching which formula to pick for the current pixel color.

4 L*a*b colorspace

In this exercise we will examine the difference between the L*a*b representations. We will see the effects of changing the values of separate channels.

4.1 L*a*b

Briefly explain about the L*a*b color space.

Explanation : The L*a*b* color space expresses color as three numerical values, L* for the lightness and a* and b* for the green – red and blue – yellow color components. It was designed to be perceptually uniform with respect to human color vision, meaning that the same amount of numerical change in these values corresponds to about the same amount of visually perceived change. **Note** : more on page 28.

4.2 RGB to L*a*b

Convert the image to L*a*b using MATLAB's *'rgb2lab()'* function and display the separate channels.

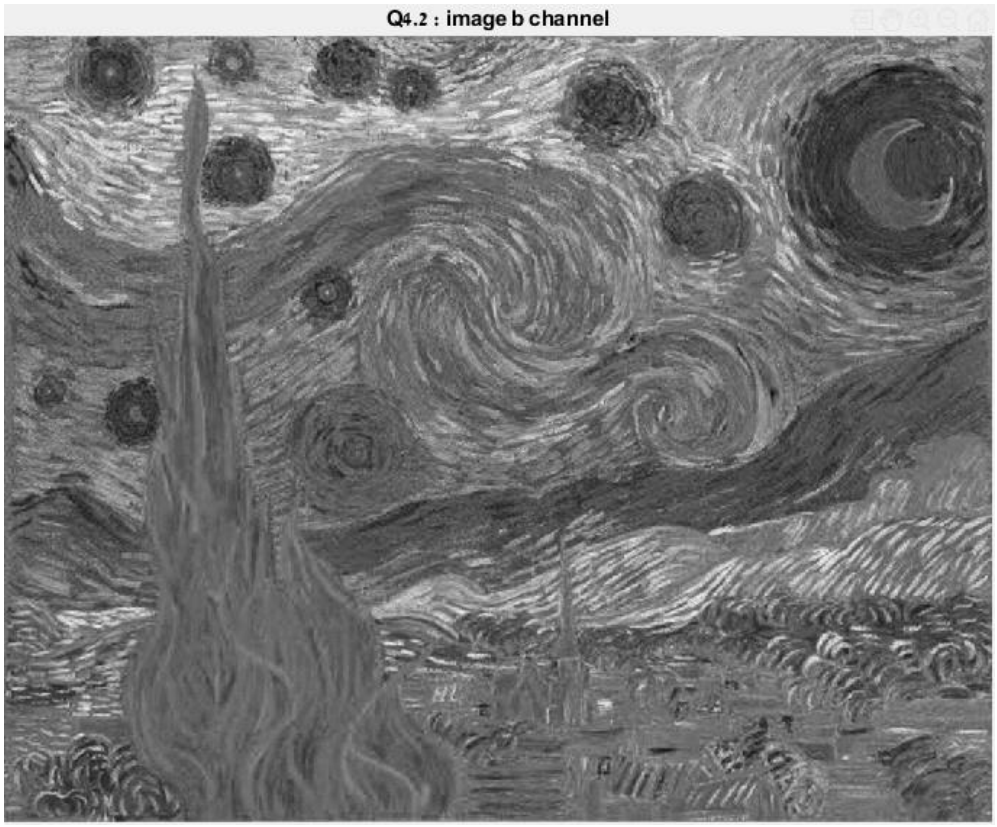


Figure 4.2.1 – b channel

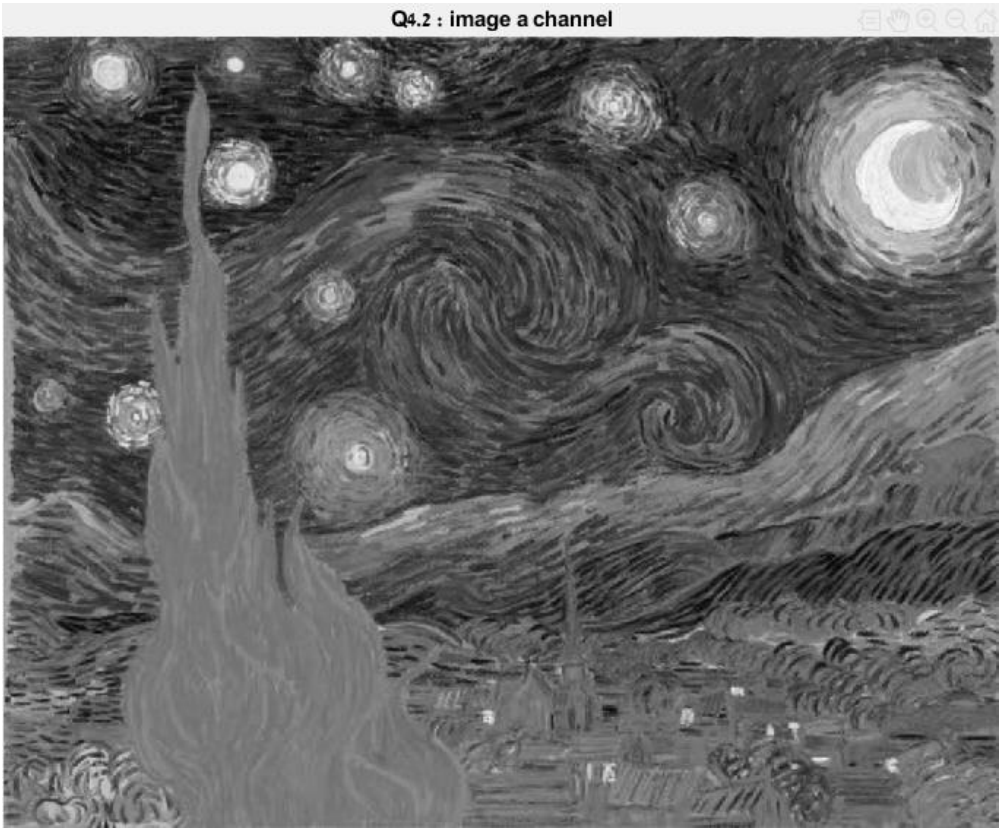


Figure 4.2.2 – a channel

Figure 4.2.3 – L channel

Note: The L channel suggest where are the brighter/darker areas.

4.3 L*a*b Manipulation

Repeat step 6 from section 1. Display the results using the provided *'imshowLab()'* function.

4.3.1 Add a constant to a single channel.

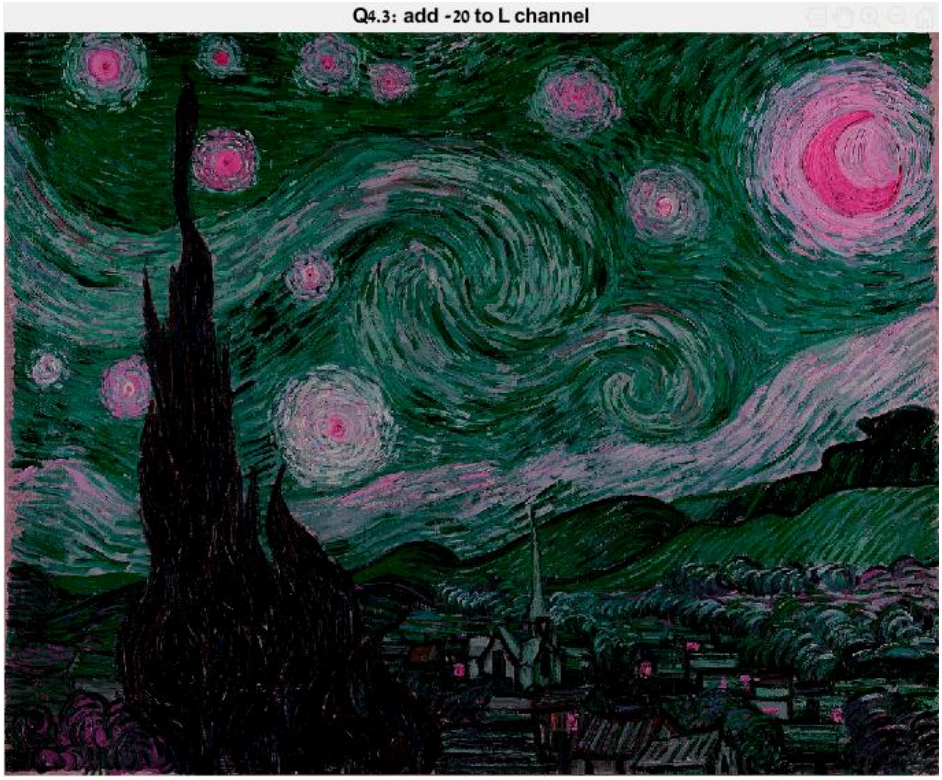


Figure 3.5.1

Explanation: After subtracting 20 from the Lightness channel we can see that each color was preserve but the image got darker tones.

4.3.2 Switch between the channels.

Figure 3.5.2

Explanation: We switch between the a and b channel. Each channel converts to different range of colors in the RGB color space, its hard for us to put a finger about something special here beside that we get a new set of color tones for the image.

4.3.3 Apply some function on a specific channel

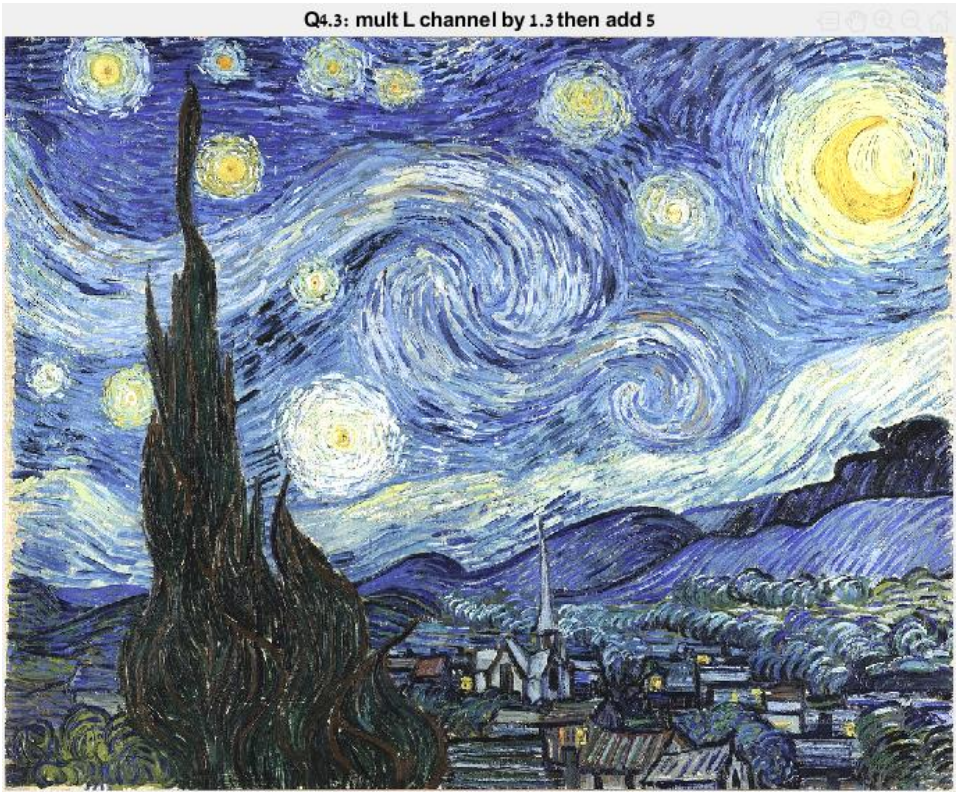


Figure 3.5.3

Explanation: The results are a brighter image while keeping the contrasts of the difference colors.

4.4 Comparing to HSV channels

Compare the channels to the HSV channels from the section 3.3. Explain the relations and differences in the report.

Explanation:

LAB :

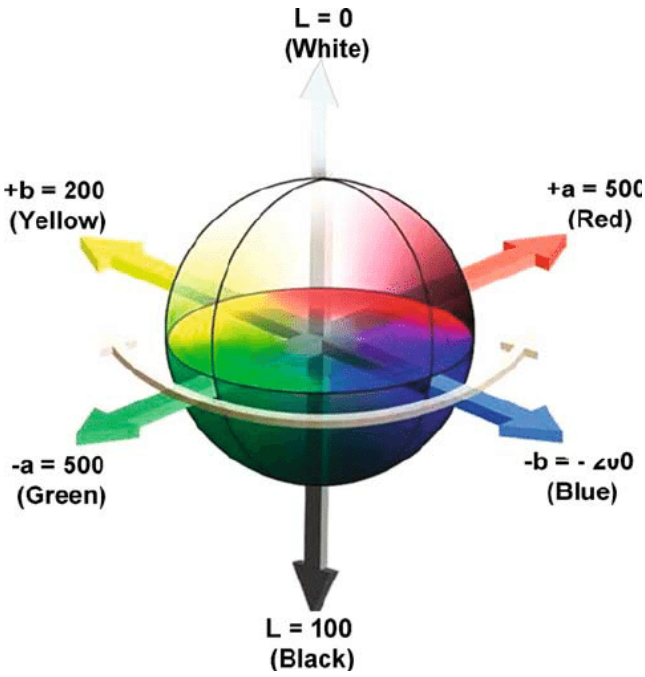
The vertical **L*** axis represents Lightness, ranging from 0-100.

The other (horizontal) axes are now represented by a* and b*.
These are at right angles to each other and cross each other in the center, which is neutral (grey, black or white). They are based on the principal that a colour cannot be both red and green, or blue and yellow.

The **a*** axis is green at one extremity (represented by -a), and red at the other (+a).

The **b*** axis has blue at one end (-b), and yellow (+b) at the other.

The centre of each axis is 0. A value of 0, or very low numbers of both a* and b* will describe a neutral or near neutral. In the case of paper, the whitepoint in terms of a* and b* is usually carried through to the black, being gradually reduced towards '0'.



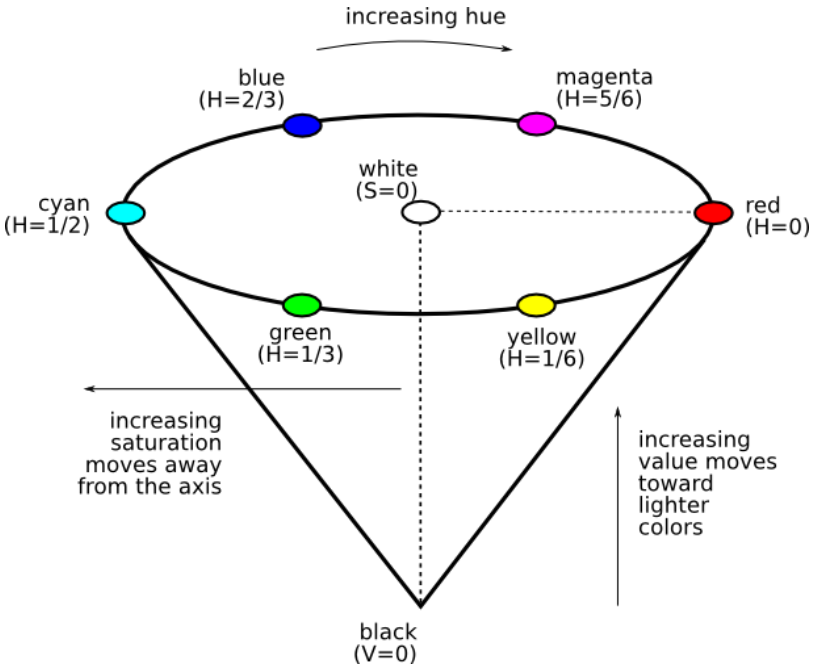
HSV :

Hue is the color portion of the color model, expressed as a number from 0 to 360 degrees:

Color	Angle
Red	0 – 60
Yellow	60- – 20
Green	120 – 180
Cyan	180 – 240
Blue	240 – 300
Magenta	300 – 360

Saturation is the amount of gray in the color, from 0 to 100 percent. Reducing the saturation toward zero to introduce more gray produces a faded effect. Sometimes, saturation is expressed in a range from just 0 – 1, where 0 is gray and 1 is a primary color.

Value (or Brightness) works in conjunction with saturation and describes the brightness or intensity of the color, from 0 – 100 percent, where 0 is completely black, and 100 is the brightest and reveals the most color.



Comparing : Lab color space was designed to be perceptually uniform with respect to human color vision, meaning that the same amount of numerical change in these values corresponds to about the same amount of visually perceived change.
The HSV scale does a better job at visually explaining the concept of light, and it is a very useful one to comprehend, as it is what most sophisticated digital color pickers are based on. Not only do graphic designers need to understand this color construct, but fine artists do as well since digital art and rendering has become such an integral part of art processes.

A cool little addition :

https://media.giphy.com/media/5U5i2VwlJaIgg/giphy.gif

5 Playing with Colors

In this exercise we will try to solve real world problems with your newly acquired knowledge.

5.1 Object circling

Use color spaces and other previously learned subjects to automatically circle the blue cap (of the soda bottle) in the *'cap1/2.png'* images.

5.1.1 Read the *'cap1/2.png'* images in MATLAB using the *'imread()'* function and normalize them to [0 , 1].

Figure 5.1.1.1

Figure 5.1.1.2

Explanation : we normalized the pictures to the range [0,1].

5.1.2 Find the blue cap of the soda bottle in the images and circle it in each image. Explain the algorithm you've used and show the result images together with the binary masks you've found. Example the algorithm steps on a chosen image. You should come up with an algorithm that finds the cap in the images, regardless of what the input image is (of the given images).

Algorithm : First we manually sampled the color of the required object (the cap) for reference point. From this color sampling we got the HSV values (according to definition) that represent the color "blue" of the cap. For this step we sample a area of 80 pixels (we choose 80 because we think it's not too big or small sampling of colors) with the function *'Impixel()'* (it lets you select pixels interactively from the image in the current axes - returns pixel values only and not position).

For now, we will work in HSV color space. We extracted the HSV channels of the original image with the help of our function *'dip_rgb2hsv()'.*

Then, we created a binary image of the original image using a threshold that consist the HSV value range from the required blue color.

Binary image of cap1.png

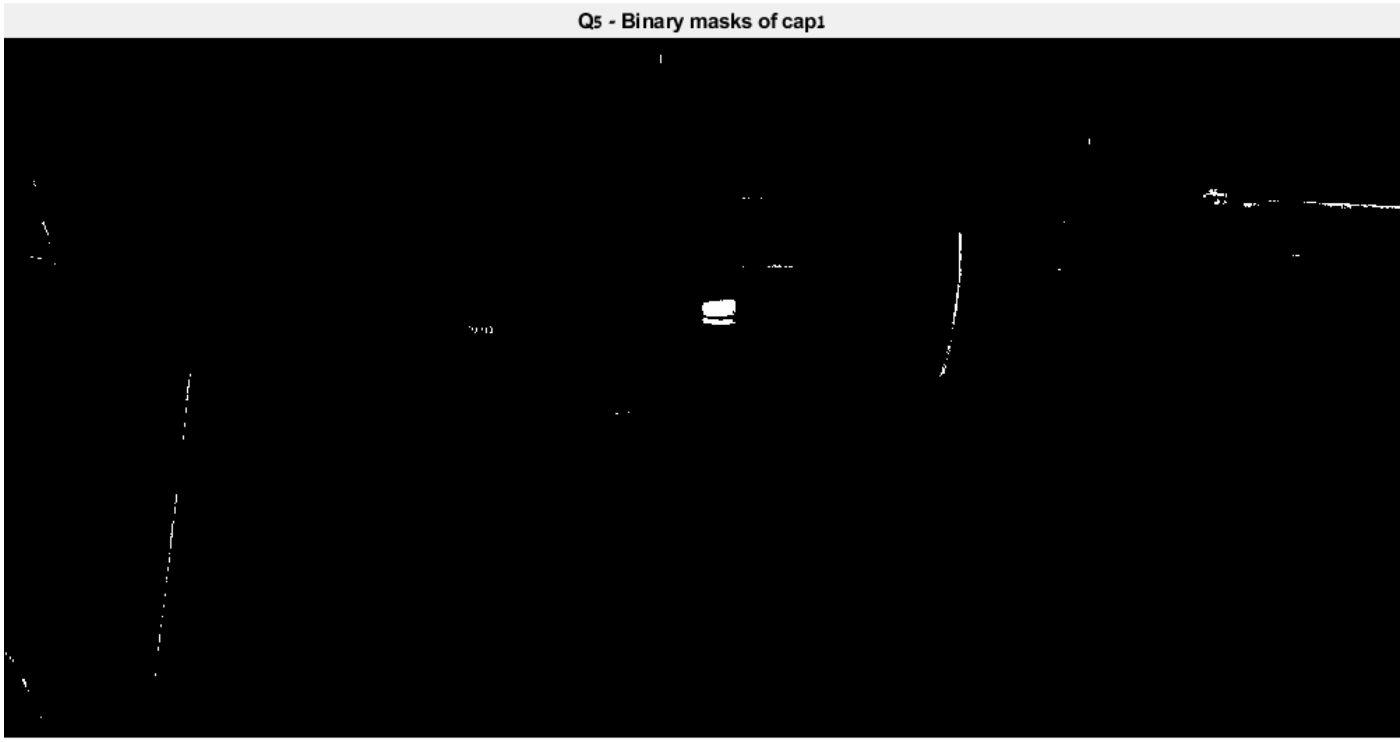


Figure 5.1.2.1

Finally, we scanned the binary image with a window of 10x6 pixels and look for a unit matrix – if we found such matrix it will suggest that we found an area that have a large amount of pixels that fits into the threshold, which means that it's an object that have a dominant cast of the required color (in our case blue) – so it's the cap.

For visualization we inserted a circle in the center of that unit matrix, with radius of 30 pixels.

Note *:* we played with the size of the scanning windows. We found out that if the lighting is good and the image is clean from extreme noises, a 20x8 size windows Can provide satisfactory results. In any case, we assumed the worse and choose a smaller window that will filter best random blue noises.

Cap1.png circling results

Figure 5.1.2.2

And for the second image :

Binary image of cap2.png

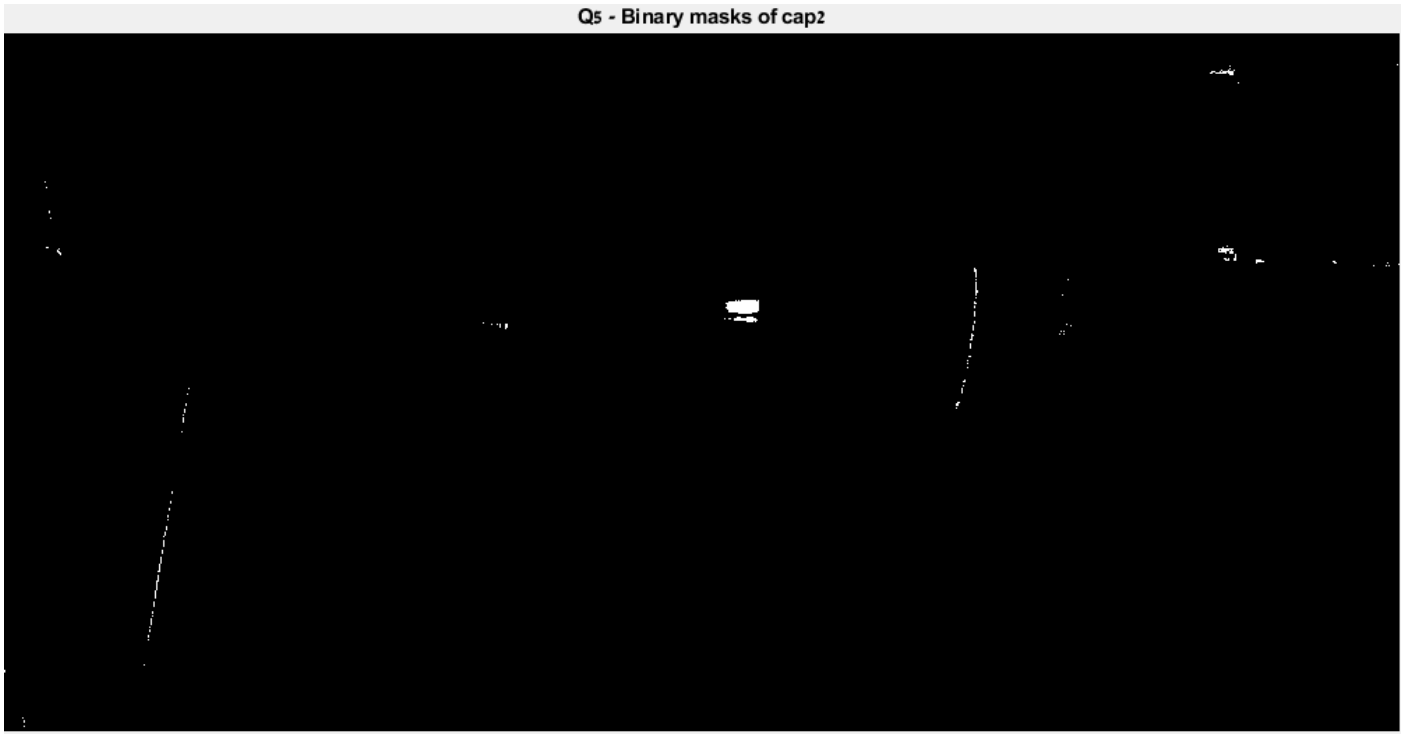


Figure 5.1.2.3

Cap2.png circling results

Figure 5.1.2.4

5.1.3 Test your algorithm on 'cap test.png' image. Don't modify the algorithm, it's alright if it fails. Show and analyze the results.

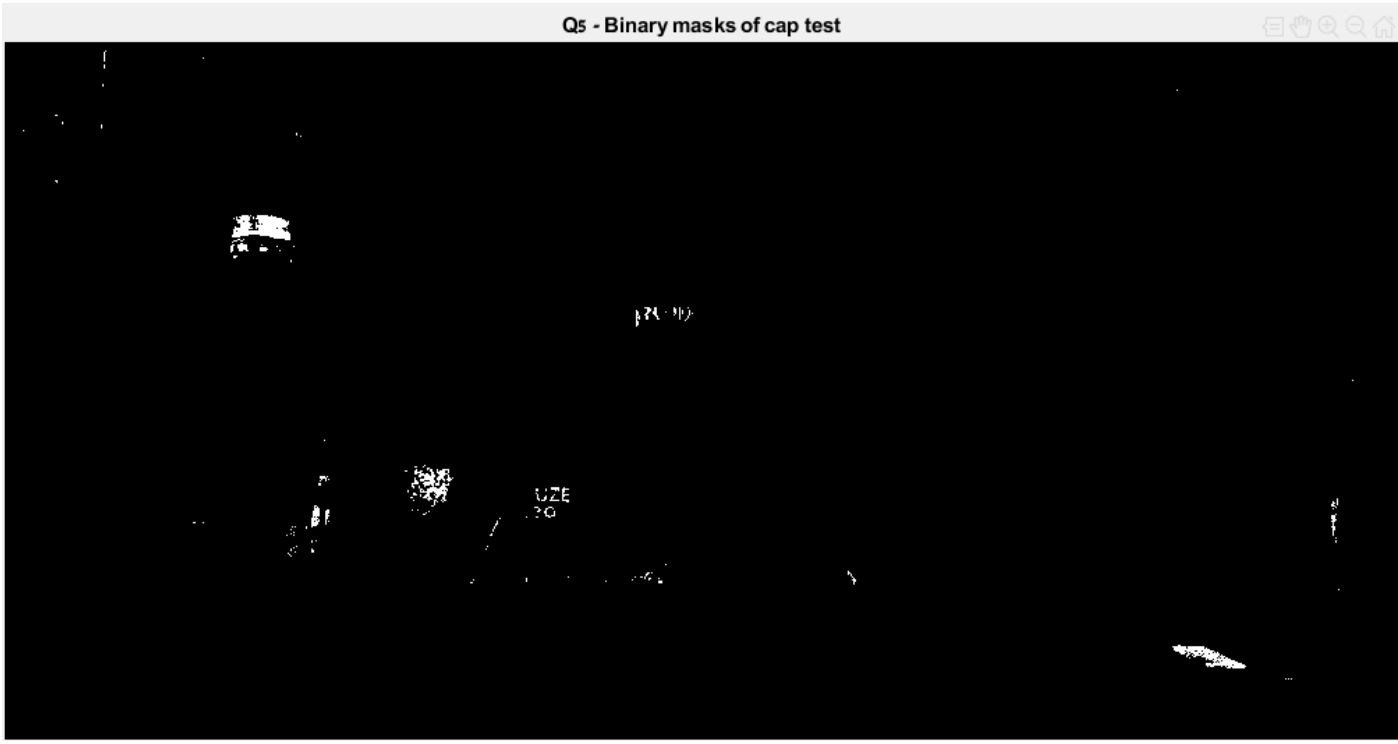


Figure 5.1.3.1 – Binary image of cap_test.png

cap_test.png circling results

Figure 5.1.3.2

Explanation :

We can see in the binary map that there are more potential areas for a blue object (or blue noises). We also can notice in the original image that the cap is photographed at a different angle, so now its color range is influenced by the shadow and so on.

We tested our algorithm on this image with the original blue color range and got A result that is not really sufficient for us.

We can see that the circling is a little off. An explanation for that is our algorithm circle around the area that fits best in our threshold (binary image) – and that happens to be the side of the cap, which is apparently similar to the original cap color values.

In addition, we can see that the center of the cap is white in the binary map, probably the result of bad lighting that effect the color of this area.

5.2 Dim lighting

At night time, when you turn on the light, some colors may look different then they really are (See Figure 2). In this section you will use white balancing to remove yellow light from an image.

5.2.1 Take a picture of a white page in your apartment at night time with the lights on. Make sure to turn of the auto white balance in your camera. If you can't find this option ask google or take a picture with a friend's camera.

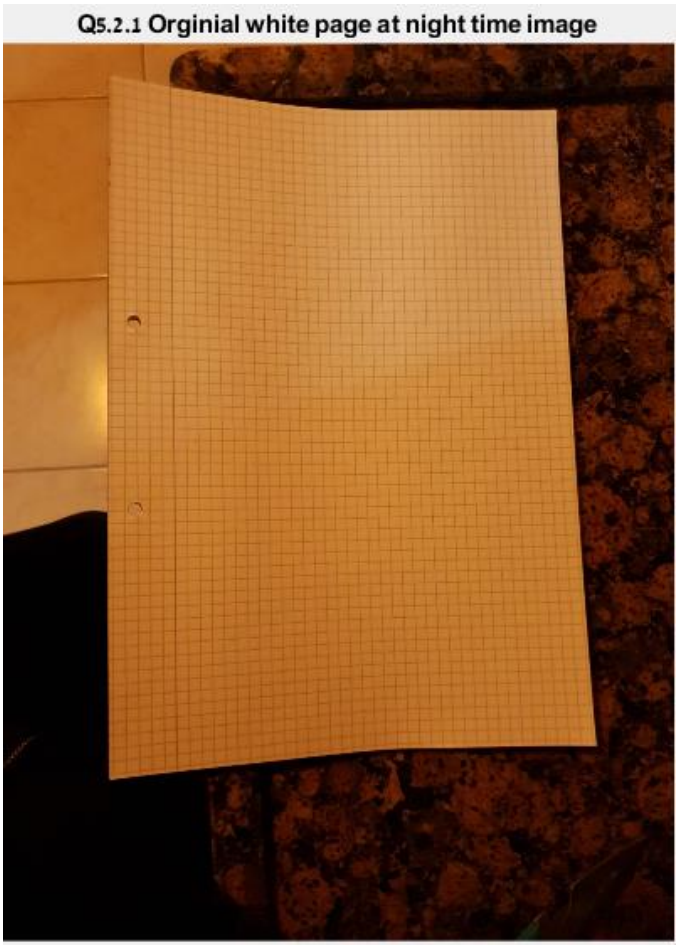


Figure 5.2.1

5.2.2 Read about white balancing and design an algorithm for removal of the yellow light from the picture.

White balance (WB) is the process of removing unrealistic color casts, so that objects which appear white in person are rendered white in your photo. Proper camera white balance must consider the "color temperature" of a light source, which refers to the relative warmth or coolness of white light. Our eyes are very good at judging what is white under different light sources, but digital cameras often have great difficulty with auto white balance (AWB) — and can create unsightly blue, orange, or even green color casts (as shown in our original picture). Understanding digital white balance can help us "improving" out photos under range of lighting conditions.

5.2.3 Explain the algorithm you've used and show the results.

Explanation :
First we extracted the RGB color channels of the original image.

Then, we convert the image to Grayscale to estimate the illuminance of the photo by the mean of the grayscale (Grayscale only contains luminance (brightness) information and no color information; that is why maximum luminance is white and zero luminance is black).

Then we correct the color balance of the image by normalizing each color channel by their mean value, and multiply by the illuminance value that we found.

This process will decrease the effect of specific color if that color has over-cast the image (e.g. yellowish picture -too much yellow to our vision) as shown in the original image.

Note : There are a few methods to estimate the illuminance of a photo, we choose the gray world method for simplicity sake – but there are more accurate methods.

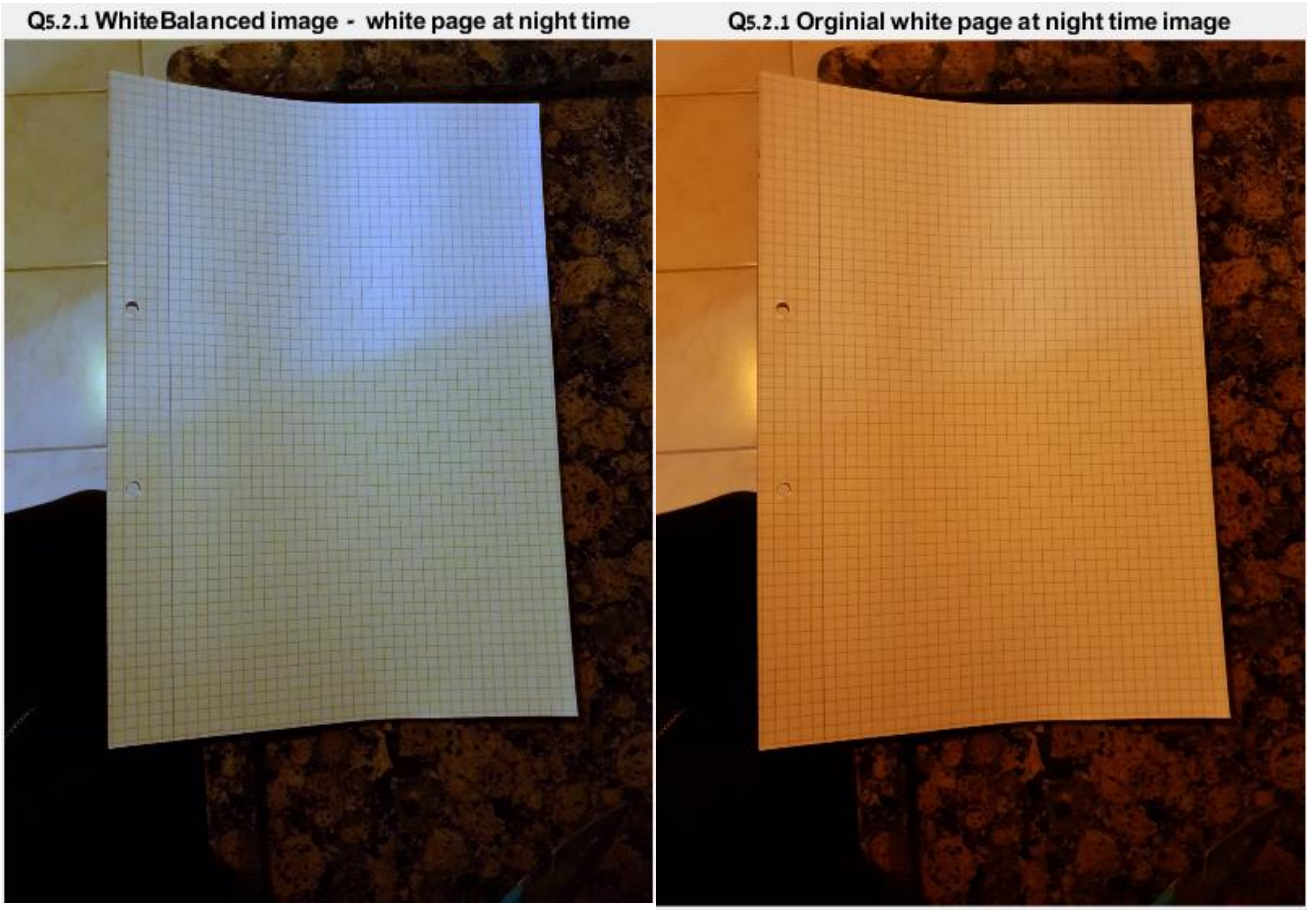


Figure 5.2.2 – result of our white balance algorithm

7 Bonus

Choose an image as you wish and manipulate it in the coolest way you can. Use methods from this exercise and optionally from the previous exercise. Display the initial image together with the modified image in the document. Be creative! One of the images will be chosen by the course standard it's authors will receive one bonus point to the final grade. The staff will judge by the visual result, originality and the code.

➔

Restore background colors

The idea : recolor background colors of a gray scale image while saving some areas that their color is gray-scale (e.g. the white tiger body is white in RGB and in Grayscale).

1. We read an image of white tiger with natural colors in the background

Figure 7.1.1 – Original image

2. Then we converted to gray scale and normalized the image to the range [0, 1]

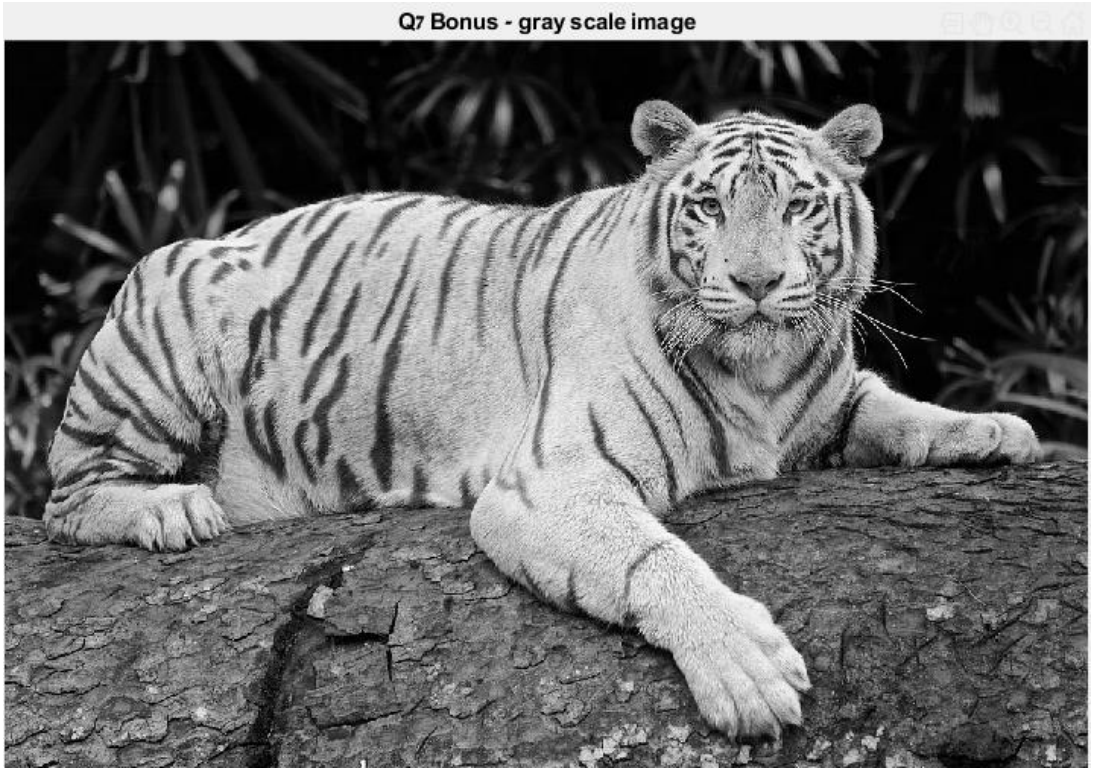


Figure 7.1.2 – Grayscale image

3. We Created a binary map of the white areas (correspond to the white body of the white tiger) – it will help us manipulate only the colors of the background.

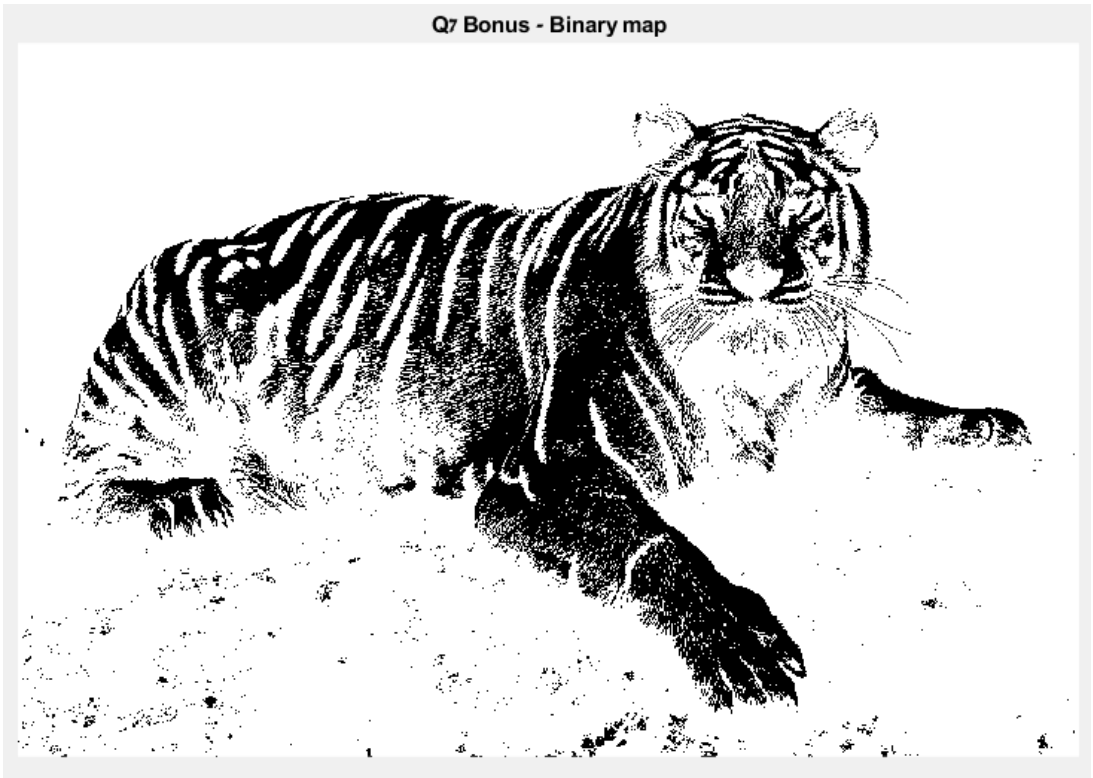


Figure 7.1.3 – Binary map

4. After extracting the HSV channels of the original image we used the binary map and some math manipulation to create a matrix that will replace the H matrix in the original image. In addition, we took the Saturation matrix from the gray scale image.

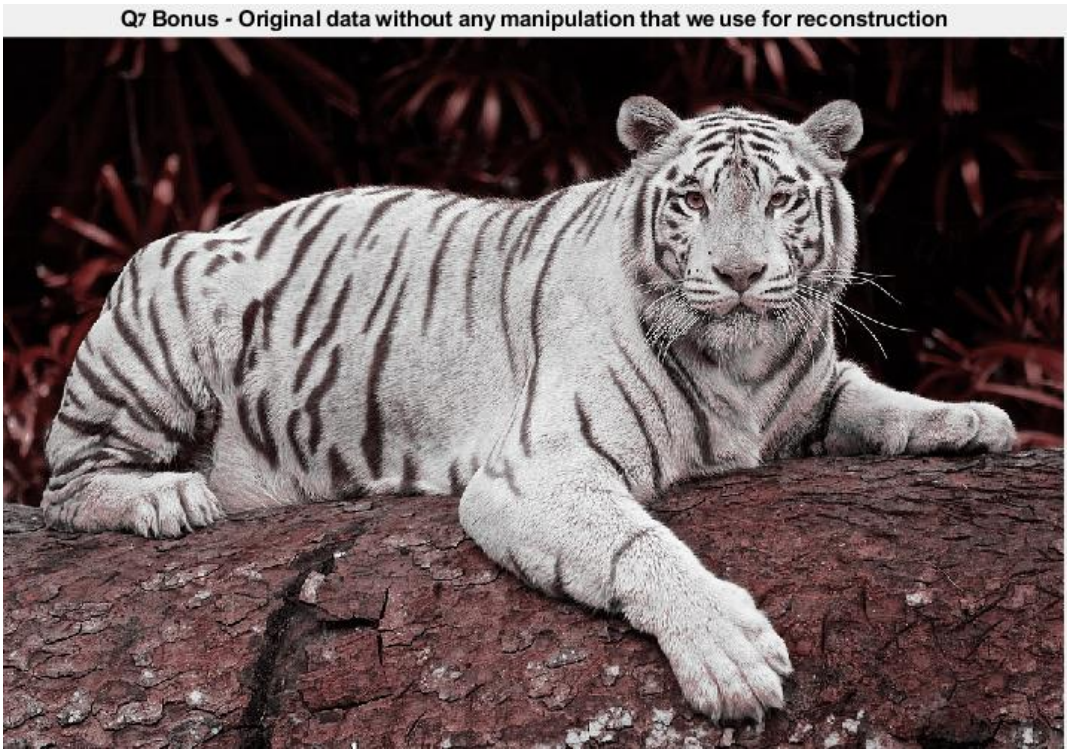


Figure 7.1.4 – Original data used to reconstruct without any manipulation

5. Finally, we showed an RGB reconstructed image from the modified HSV channels.

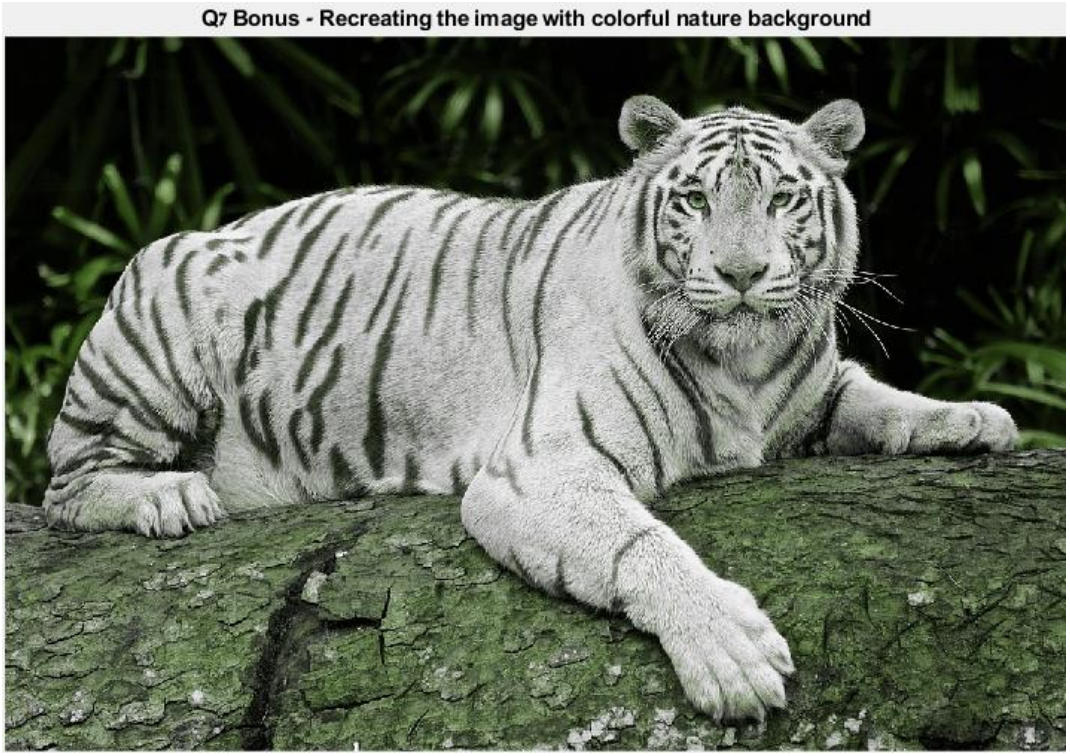


Figure 7.1.5 – The result – reconstruct the image with a colorful background

➔ For another example we used the following images, which turns out beautifully :

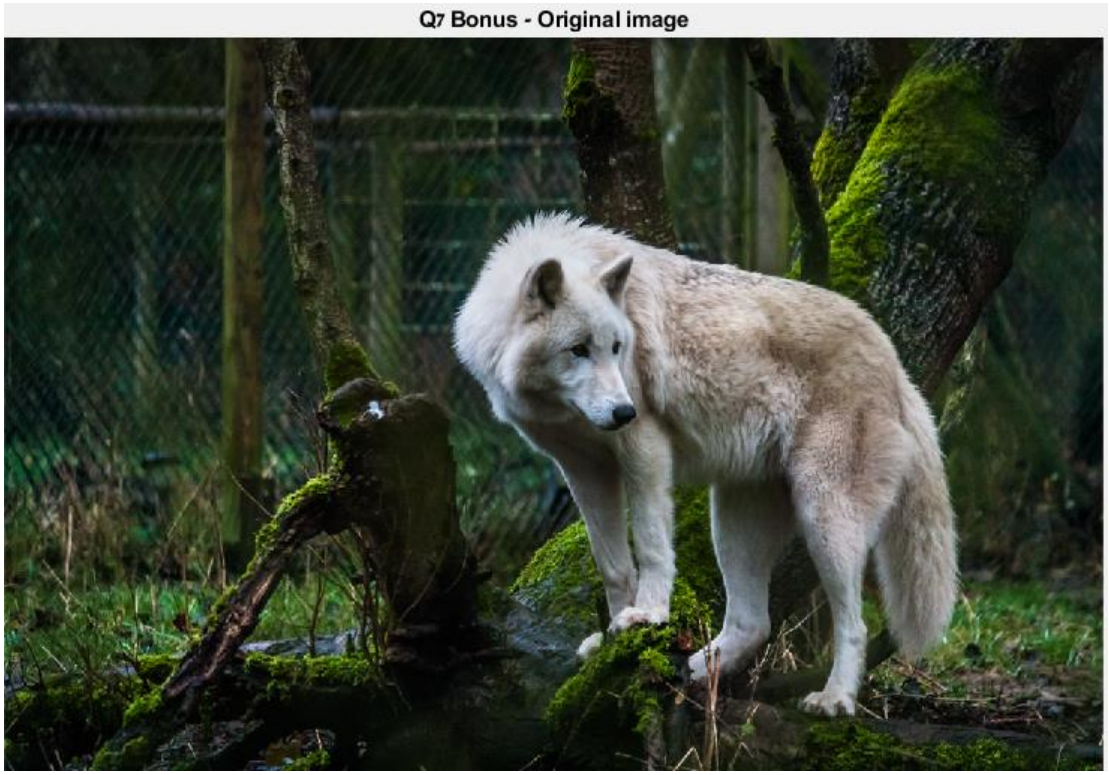


Figure 7.2.1 – Original image

Figure 7.2.2 – Grayscale image

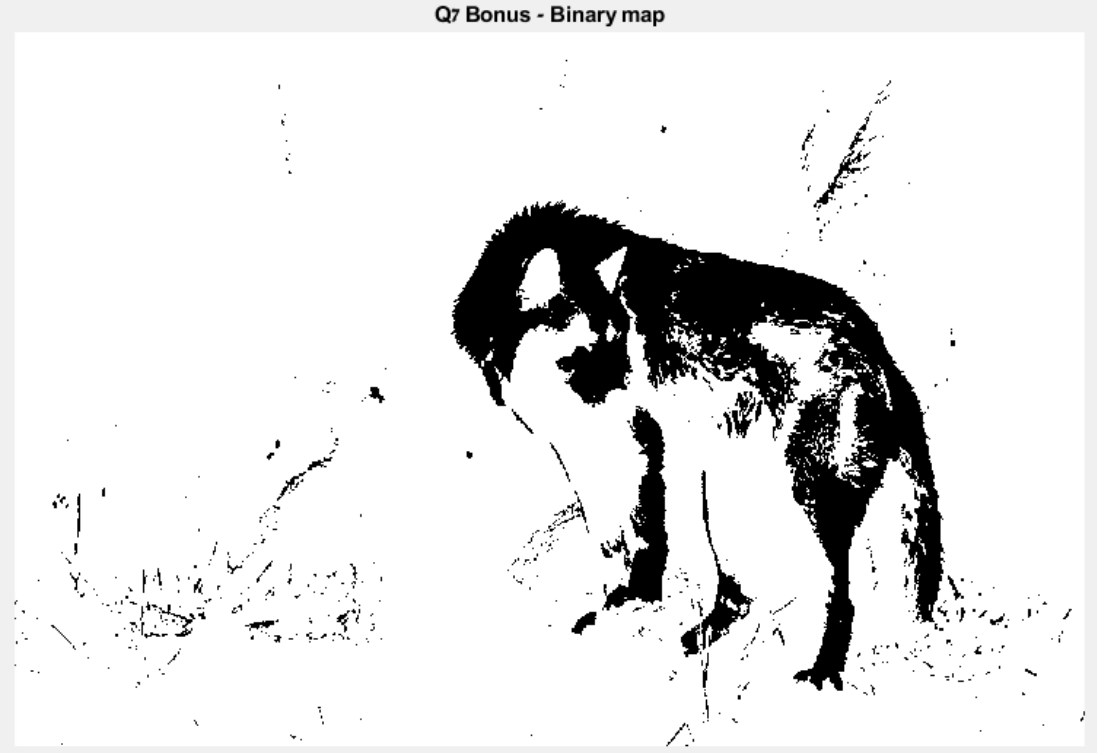


Figure 7.2.3 – Binary map

Figure 7.2.4 – Original data used to reconstruct without any manipulation

Figure 7.2.5 – The result – reconstruct the image with a colorful background