EE 5630 Project 3

Anisa Alshammasi, Varun Bharadwaj, Matthew Sundberg
4/15/2021

Abstract:

Image processing techniques such as image smoothing, color image enhancement are trending fields which help in improving the quality of test images as some images are required to be improved in order to get useful information from these images. Some of the test images don't appear to be natural, it is processed to give a natural look. In this work, multiple image processing tasks containing color model conversions, smoothing, color image enhancement, and green screen techniques are performed on multiple test color images. Test image of dimension 256 by 256 containing 3 color planes was created. That test image was split into R, G and B components and displayed as grayscale. R, G, B components are normalized to [0, 1] and then it is converted to HSI and YCbCr color models and the components are displayed individually in 2x2 tile format. An un-weighted averaging filter of 9 x 9 window size is applied on Bayer pattern test images with the help of zero padding for edges in order to perform color image smoothing. Another test image of dimension 512 × 512 was loaded to perform color image enhancement and is applied on a test image named as Lena image. This image appears to be much orange in color, hence this test image is processed to get more natural and pleasing skin tone. At the end, a green_screen.mat file containing two color images: AFA of dimension of 1312 × 2000 and F16 of dimension of 466 × 720 are loaded in order to perform green screen techniques. The aircraft is displayed in the upper right area of AFA image. The results of applied methods are also observed using different color models in order to check their effects which gives better visualization. All programs run on MATLAB software.

Keywords—image enhancement, smoothing, green screen method, color model conversions.

Introduction:

Image enhancement is used to improve the usefulness of an image for a specific task such as providing a more pleasant image to humans. It estimates the actual image degradation process. This process does not increase the inherent information content in data. It is a subjective process. It contains adjustment of contrast, reduction of noise, edge sharpening, filtering and interpolation. Image enhancement methods can be composed into two classes: frequency domain methods and spatial domain methods. The frequency domain methods process the image as a two-dimensional signal and enhance the image based on its two-dimensional Fourier transform. The low pass filter removes noise from the image while the high pass filter enhances the edge, which is a high frequency signal, and clears the blurred image. Spatial domain based methods are used to perform mean filtering and median filtering (take intermediate pixel value of local neighborhood) to reduce the noise from the given image.

There are multiple ways to perform image enhancement. Smoothing is another type of image enhancement technique which is used to reduce noise within an image. Image smoothing is a method of improving the quality of images. Smoothing is also performed using spatial and frequency filters as discussed before. Many other methods are used to perform these operations in order to enhance the quality of a given image.

In this assignment, multiple image processing tasks have been performed on color images such as color model conversions, smoothing, enhancement, and green screen techniques. RGB, HSI, and YCbCr components of Bayer pattern test images are observed. An un-weighted averaging filter with window size of 9 x 9 is applied on Bayer pattern test image using zero padding to deal with edge effects using three different color models containing RGB, HSI, and YCbCr.

Background:

Each color model has its own importance. They are used collectively and specifically as well depending on their effects. RGB color model is used to capture, view and display color images. The strong correlation between red, green and blue and the incomplete separation of brightness and color information reduces their use in the field of image analysis. Multiple color channels are used to detect registration artefacts provide higher contrast for different visual symbols of natural skin color [1] and [2].

HSV (Hue-Saturation Value) and YCbCr are also color models which are based on separate components of gloss and chroma. The HSV color model uses size of hue and saturation in order to find the color of image, and size of value represents brightness. Similarly, the YCbCr color model differentiates the components such as brightness of RGB (Y), chroma blue (Cb) and chroma red (Cr). The appearance of chrominance components in HSV and YCbCr color models is different which can be used for multiple purposes [3] and [4].

In order to enhance the intensity in order to get high-quality color image enhancement, the hue component is kept unchanged and HSV, HSL and Hue Saturation Intensity (HSI) can be used. HSI is a public color model, and many color applications are commonly based on this model but the transformation from HSI color model to RGB color model gives problems after modifying intensity and saturation in HSI color model. Furthermore, saturation component increases or decreases with change of intensity component. In one work, accurate formulas for color transformation between RGB and HSI color models are proposed in order to reduce the problem directly. Experiments suggest that proposed methods have worked well [5].

Methods:

Test image equivalent to Figure 7.43(h) was created by initially allocating a space of 256x256 containing 3 color planes. Later appropriate values were assigned to each color plane to obtain the test image which is shown in Figure 1 of the results section.

The test image is split into R, G and B components and displayed as grayscale. This was done by displaying different color planes individually.
Using our own conversion routines, we were able to convert RGB to HSI and vice versa.

Converting from RGB to HSI

$$\boxed{H} = \begin{cases} \theta & \text{if } B \le G \\ 360^\circ - \theta & \text{if } B > G \end{cases} \quad \text{for } H \text{ in range } \left[0^\circ, 360^\circ\right]$$

$$\text{where} \quad \theta = \cos^{-1}\left\{ \frac{\frac{1}{2}[(R-G)+(R-B)]}{\sqrt{(R-G)^2 + (R-B)(G-B)}} \right\}$$

$$\boxed{S} = 1 - \frac{3}{R+G+B}[\min(R,G,B)] \quad \text{and} \quad \boxed{I} = \frac{1}{3}(R+G+B)$$

This assumes that R,G,B values are normalized to [0,1] and that θ is measured in degrees from R in the direction of G

Converting from HSI to RGB

	RG sector $0^\circ \le H < 120^\circ$	GB sector $120^\circ \le H < 240^\circ$ set $H = H - 120^\circ$	BR sector $240^\circ \le H < 360^\circ$ set $H = H - 240^\circ$
R	$I\left[1 + \frac{S\cos H}{\cos(60^\circ - H)}\right]$	$I(1-S)$	$3I - (G+B)$
G	$3I - (R+B)$	$I\left[1 + \frac{S\cos H}{\cos(60^\circ - H)}\right]$	$I(1-S)$
B	$I(1-S)$	$3I - (R+G)$	$I\left[1 + \frac{S\cos H}{\cos(60^\circ - H)}\right]$

Converting from RGB to YCbCr

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0.2988 & 0.5869 & 0.1143 \\ -0.1689 & -0.3311 & 0.5000 \\ 0.5000 & -0.4189 & -0.0811 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

This conversion assumes that R,G,B values are normalized to [0,1]

Converting from YCbCr to RGB

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.0 & 0.0 & 1.402 \\ 1.0 & -0.3441 & -0.7141 \\ 1.0 & 1.772 & 0.00015 \end{bmatrix} \begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix}$$

After converting from RGB to HSI and YCbCr color models, we displayed the components individually in a 2x2 tile format.

Verification :

RGB to HSI

Pixel value of test image in RGB color model at location (50,50) is [0, 0, 1]

Using the equations mentioned above

$\theta = cos^{-1}((0.5 * ((0 - 0) + (0 - 1))/((0 - 0)^2 + (0 - 1)(0 - 1))^{0.5}))$

θ = 120 (in degrees)

H = 120

Since B>G: H = 120

Using our conversion routine RGB_HSI.m we got **H = 120**

S = 1 - (3/0+0+1)*min(0,0,1)

S = 1

Using our conversion routine RGB_HSI.m we got **S = 1**

I = (0+0+1)/3

I = 1/3

Using our conversion routine RGB_HSI.m we got **I = 1/3**

HSI to RGB

Pixel value of test image in HSI color model at location (50,50) is [120, 1, 1/3]

Using the equations mentioned above

R = (1/3)*(1 - 1)

R = 0

Using our conversion routine HSI_RGB.m we got **R = 0**

G = (1/3)*(1+$(1 * (cos(120)/cos(60 - 120)))$)

G = 0

Using our conversion routine HSI_RGB.m we got **G = 0**

B = 3*(1/3) - (0+0)

B = 1

Using our conversion routine HSI_RGB.m we got **B = 1**

RGB to YCbCr

Pixel value of test image in RGB color model at location (50,50) is [0, 0, 1]

Using the equations mentioned above

Y = 0.2988*0 + 0.5869*0 + 0.1143*1

Y = 0.1143

Using our conversion routine RGB_YCbCr.m we got **Y = 0.1143**

Using the equations mentioned above

Cb = -0.1689*0 - 0.3311*0 + 0.5000*1

Cb = 0.5000

Using our conversion routine RGB_YCbCr.m we got **Cb = 0.5000**

Using the equations mentioned above

Cr = 0.5000*0 - 0.4189*0 - 0.0811*1

Cr = -0.0811

Using our conversion routine RGB_YCbCr.m we got **Cr = -0.0811**

YCbCr to RGB

Pixel value of test image in YCbCr color model at location (50,50) is [0.1143, 0.5000, -0.0811]

Using the equations mentioned above

R = 1*0.1143 + 0*0.5000 + 1.402*(-0.0811)

R = 0.0005978

Using our conversion routine RGB_YCbCr.m we got **R = 0.000597647**

Using the equations mentioned above

G = 1*0.1143 - 0.3441*0.5 - 0.7141*(-0.0811)

G = 0.00016351

Using our conversion routine RGB_YCbCr.m we got **G = 0.000163529**

Using the equations mentioned above

B = 1*0.1143 + 1.772*0.5 + 0.00015*(-0.0811)

B = 1.000287835

Using our conversion routine RGB_YCbCr.m we got **B = 1.000392157**

Next part of the project we had to create a 256x256 Bayer pattern mosaic. Initially we created a small portion of bayer image (64x64) as shown in the snippet below

```
X = zeros(64,64,3);
%Creating a small portion of bayer image
X(33:64,1:32,2) = 255;
X(1:32,33:64,2) = 255;
X(1:32,1:32,3) = 255;
X(33:64,33:64,1) = 255;
```

We used the ‘repmat’ command to repeat the 64x64 bayer image and create a 256x256 bayer mosaic.

```
%Creating bayer pattern using repmat
Z = repmat(X,4,4);
```

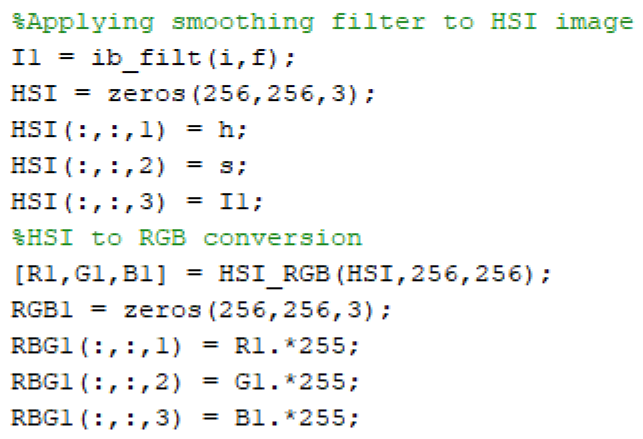
A 9x9 unweighted averaging filter is applied to Bayer mosaic in different color models and the effects are observed . Before applying the filter to an image, we need to zero pad the image to avoid edge effects. This is taken care of by our routine ‘ib_filt’.

$$F = (1/81) \quad * \quad \begin{vmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{vmatrix}$$

In the RGB color model we apply the filter to all the color planes individually and then combine them to obtain a smoothed RGB image.

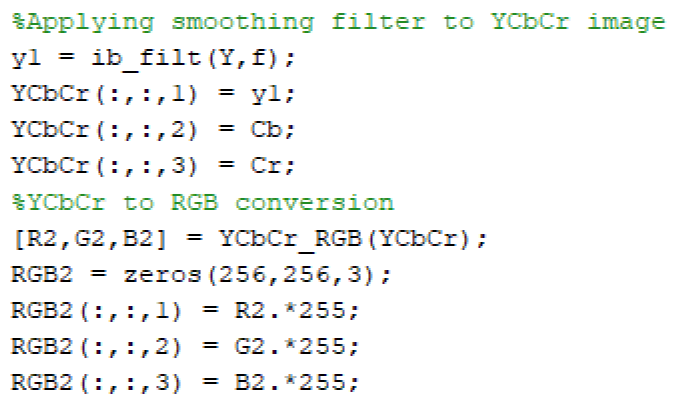
```
%Applying smoothing filter to RGB image
R = ib_filt(Z(:,:,1),f);
G = ib_filt(Z(:,:,2),f);
B = ib_filt(Z(:,:,3),f);
RGB = zeros(256,256,3);
RBG(:,:,1) = R;
RBG(:,:,2) = G;
RBG(:,:,3) = B;
```

In HSI color model we apply the filter only to the I color plane. Later we convert it to RGB for displaying



```
%Applying smoothing filter to HSI image
I1 = ib_filt(i,f);
HSI = zeros(256,256,3);
HSI(:,:,1) = h;
HSI(:,:,2) = s;
HSI(:,:,3) = I1;
%HSI to RGB conversion
[R1,G1,B1] = HSI_RGB(HSI,256,256);
RGB1 = zeros(256,256,3);
RBG1(:,:,1) = R1.*255;
RBG1(:,:,2) = G1.*255;
RBG1(:,:,3) = B1.*255;
```

In YCbCr color model we apply the filter only to the Y color plane. Later we convert it to RGB for displaying

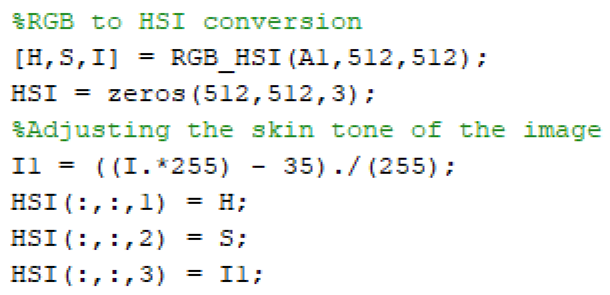


```
%Applying smoothing filter to YCbCr image
y1 = ib_filt(Y,f);
YCbCr(:,:,1) = y1;
YCbCr(:,:,2) = Cb;
YCbCr(:,:,3) = Cr;
%YCbCr to RGB conversion
[R2,G2,B2] = YCbCr_RGB(YCbCr);
RGB2 = zeros(256,256,3);
RGB2(:,:,1) = R2.*255;
RGB2(:,:,2) = G2.*255;
RGB2(:,:,3) = B2.*255;
```

512x512 image of Lena is loaded into MATLAB. The image is split into different components in different color models as we did in the first part of the project. Results section shows the lena image split into R, G ,B components and H, S, I components.

The color image of Lena which is provided to us has an orangish skin tone. To correct the skin tone, we initially converted the image into HSI color model. As seen before, the 'I' component represents the intensity level of colors 'I' component in HSI model can be written in terms of RGB component

I.e I = (R+G+B)/3. 'I' component is the average of 'R', 'G' and 'B' components. Hence if we remove some data from the I color plane, it should reduce the intensity of colors and we should be able to get a natural skin tone for Lena image. The snippet below shows the exact process.



```
%RGB to HSI conversion
[H,S,I] = RGB_HSI(A1,512,512);
HSI = zeros(512,512,3);
%Adjusting the skin tone of the image
I1 = ((I.*255) - 35)./(255);
HSI(:,:,1) = H;
HSI(:,:,2) = S;
HSI(:,:,3) = I1;
```

At the end, another file named as green_screen.mat file is loaded in order to perform green screen techniques. This loaded file contains two color images: 1st one is AFA of dimension 1312 × 2000 while 2nd one is F16 of dimension 466 × 720. The F16 image does not contain original background and contains highly saturated green hue in order to apply a green screen effect. A composite image using green screen techniques was created to make it look like an F-16C is flying over Air Force Academy. Furthermore, the aircraft is required to appear in the upper right area of the AFA image. The F16 image is interpolated to have the same dimension as the background image. Interpolation is performed separately on individual color planes. Now the image is scaled to adjust the size of the jet. We scale the individual color planes separately. Later the image is translated to an appropriate position. These processes are shown below in the snippet.

```
% Interpolating the green screen image to have same dimension as background
B1 = ib_bi_int(F16(:,:,1),1312,2000);
B2 = ib_bi_int(F16(:,:,2),1312,2000);
B3 = ib_bi_int(F16(:,:,3),1312,2000);
% Scaling the green screen image
S1 = ip_scale(uint8(B1),0.15,0.15);
S2 = ip_scale(uint8(B2),0.15,0.15);
S3 = ip_scale(uint8(B3),0.15,0.15);
% Translating the green screen image to approriate position
T1 = ib_translate(S1,0,1701);
T2 = ib_translate(S2,0,1701);
T3 = ib_translate(S3,0,1701);
```

Later a sphere based color splicing is applied to isolate the jet from the green screen and combine it with the background to get the final image. The obtained image is smoothed using a 3x3 averaging filter to have gradual variation in the edges.

```
% Performing sphere splicing
c = [10, 250, 10]; % RGB components
r = 45; % Radius
distance = zeros(1312,2000);
for i=1:1312
    for j=1:2000
        distance(i,j) =  (T(i,j,1) - c(1)).^2 + (T(i,j,2) - c(2)).^2 + (T(i,j,3) - c(3)).^2;
        if distance(i,j) > r^2 % not green
            A(i,j,1) = T(i,j,1);
            A(i,j,2) = T(i,j,2);
            A(i,j,3) = T(i,j,3);
        end
    end
end
```

Results:

Figure 1 shown below shows figure 7.43(h) from the textbook split into its RGB components. The original image is shown in the top left while the three components are labeled and shown in the other three corners. When any of the components are showing white, there exists the color in those specific pixels. Meanwhile, black shown on the components show that there exists none of that specific color in that pixel.

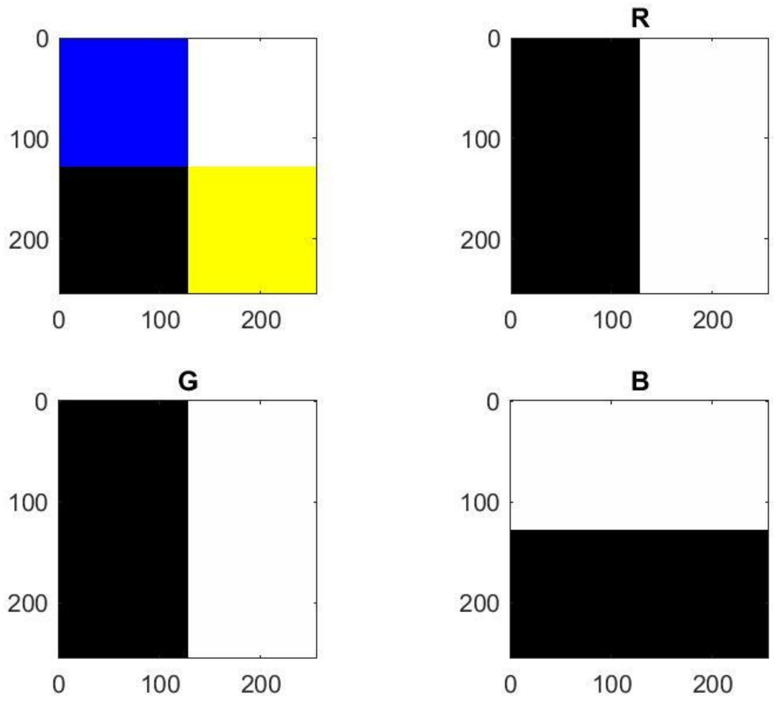


Figure 1: Test image 7.43(h) broken down into its RGB components.

From figure 1 we can see that the components are showing correctly. In the original image, there is only blue in the top left of the four squares. The three components show this properly by having only the blue component B to be white in that corner. Same logic can be applied to the yellow part of the image. Both R and G show this square as white since both red and green make yellow from color theory. In regards to the white and black squares, white consists of all three components while black has none of the components.

Figures 2 & 3 below show the original image in figure 1 but is now split into HSI and YCbCr color models. Figure 2 shows the image broken up into its hue (H), saturation (S), and intensity (I) components. From the figure, we can see that the hue component exists in all four corners since hue is shown as an angle, it is always prevalent. Intensity is showing the correct levels since yellow emits a greater intensity than blue, white shows max intensity and black shows minimum intensity. Saturation's purpose is to bring more color into the particular pixel. This shows correctly in figure 2 with saturation being fully white in the squares while black in the squares where there is no specific color shown.

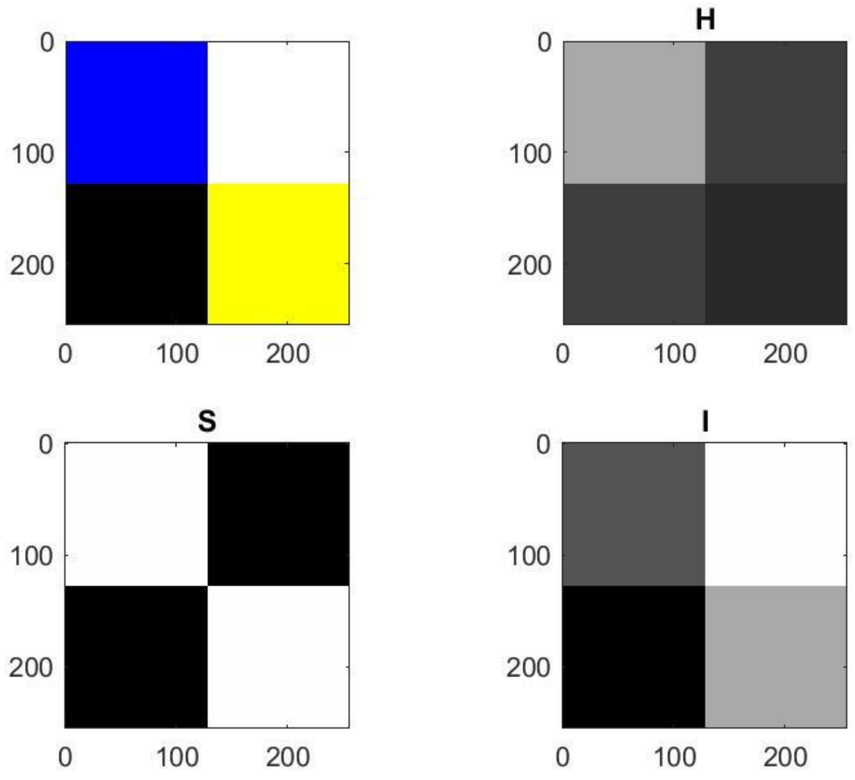


Figure 2: Test image 7.43(h) broken down into its HSI components.

Figure 3 shows the same original image, now it is broken into the YCbCr color model. The part of the initial image shows luminance(Y) and we can see that the white corner shows full luminance and the yellow square is partial. The chroma blue(Cb) shows fully in the blue square but a piece of it contains the chroma red(Cr) component. Yellow on the other hand has no chroma blue component since the YCbCr spectrum has no blue-difference to produce yellow.

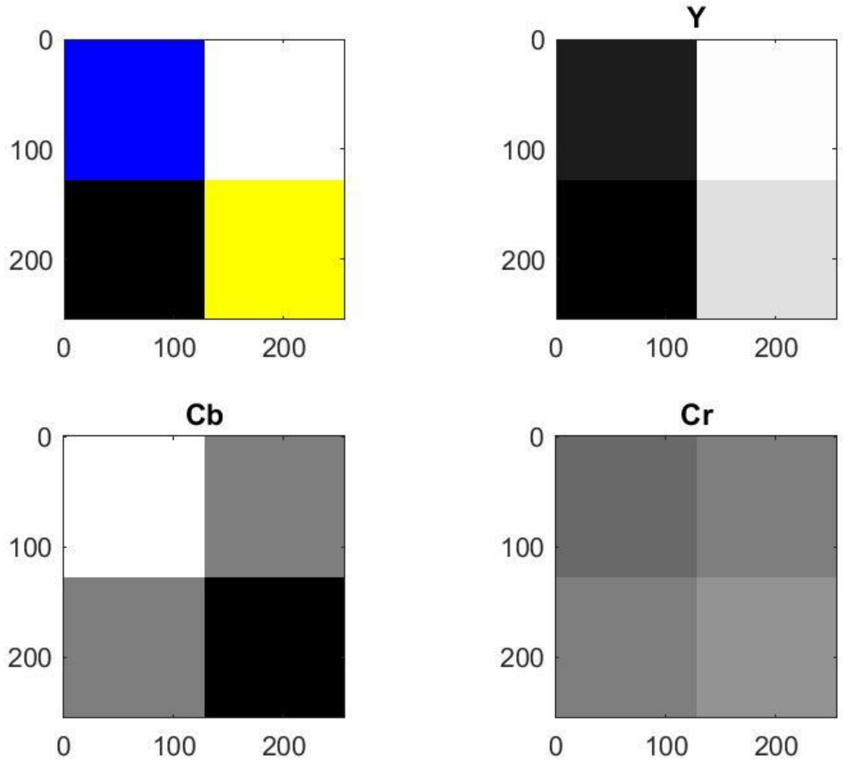


Figure 3: Test image 7.43(h) broken down into its YCbCr components.

Figures 4, 5, and 6 show the Bayer pattern test image split into the different color models shown from figures 1 through 3. Figure 4 is the Bayer pattern broken into its RGB components. Since the Bayer pattern shows a series of squares of 50% green, 25% red, and 25% blue, the distinction between these different sections of the RGB color model are easy to identify.

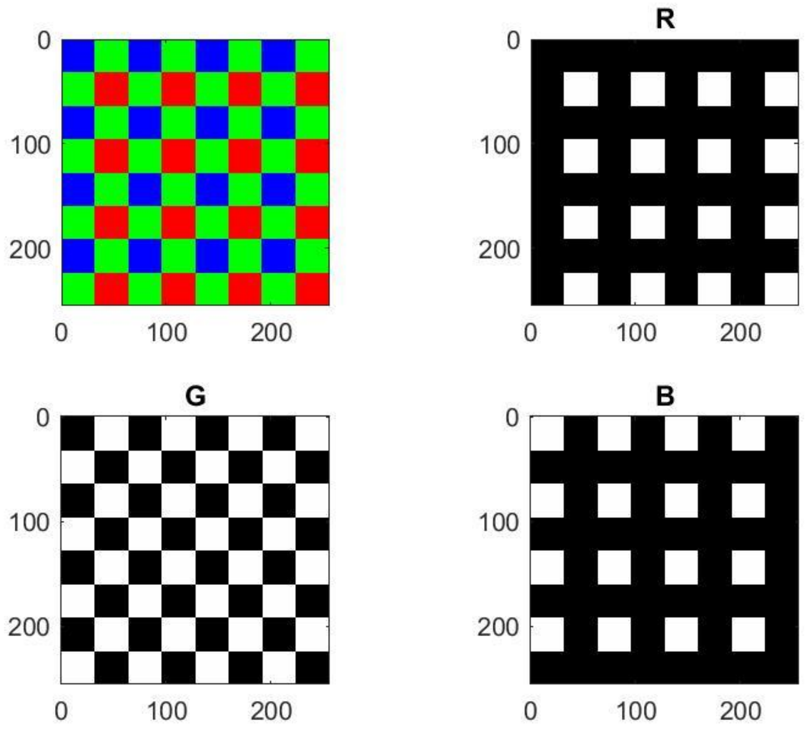


Figure 4: Bayer Pattern test image broken down into its RGB components.

Figure 5 is the same Bayer image but is showing the pieces of the HSI color model. The saturation and intensity are fully one color. Hue shows white for both blue and green squares but is black for all red squares.

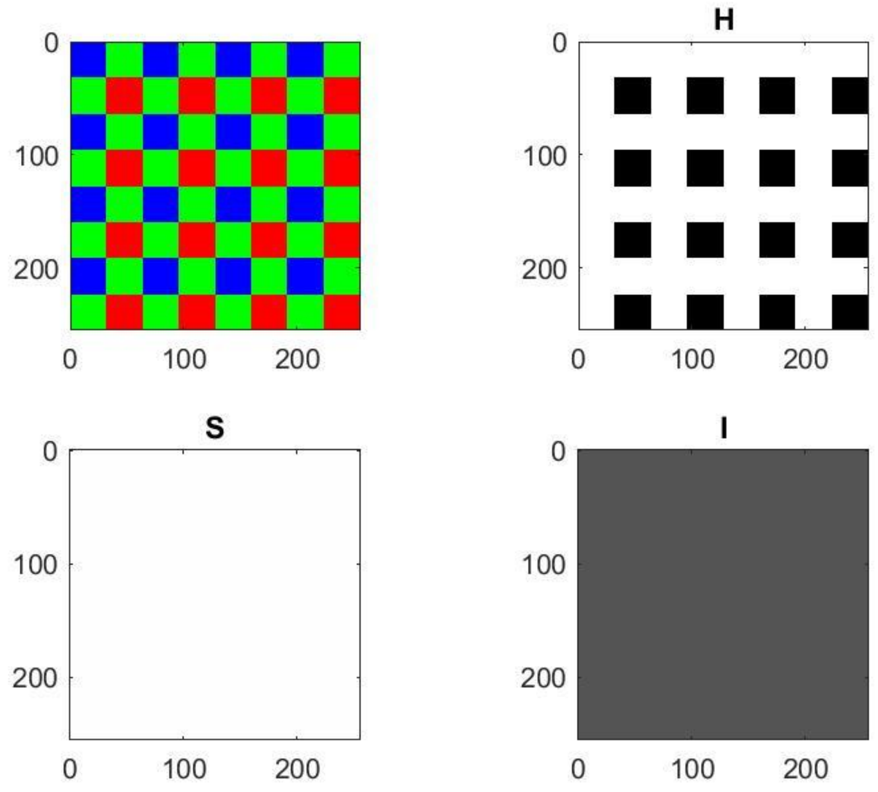


Figure 5: Bayer Pattern test image broken down into its HSI components.

The final color model applied to the Bayer pattern is figure 6. The most important thing of note in this figure are the green squares showing in both chroma blue and red. They are both showing fully black since both Cb and Cr are in reference to green. The luminance is a blend of grays with blue squares from the original image being the darkest.

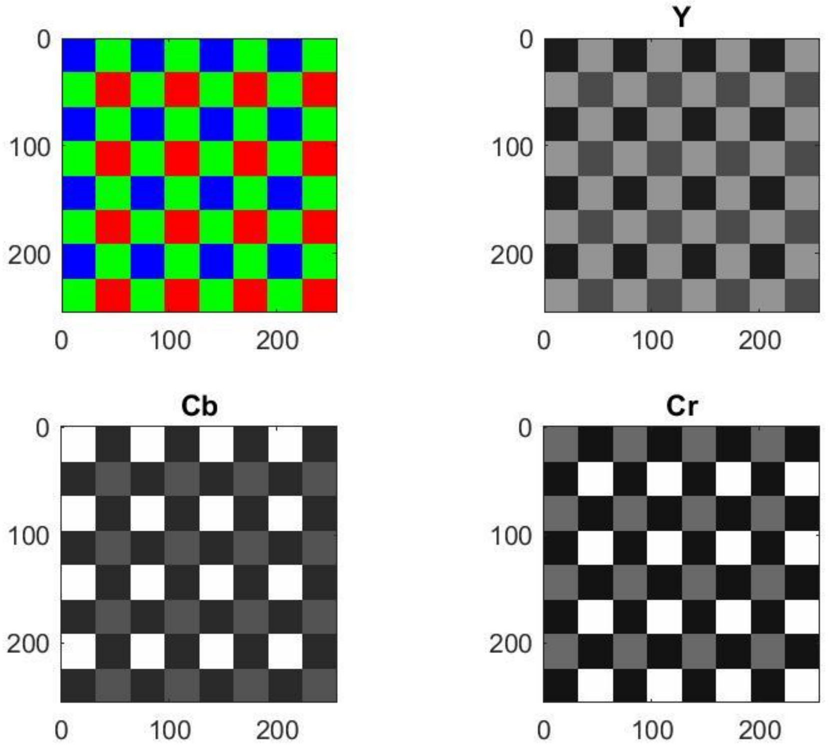
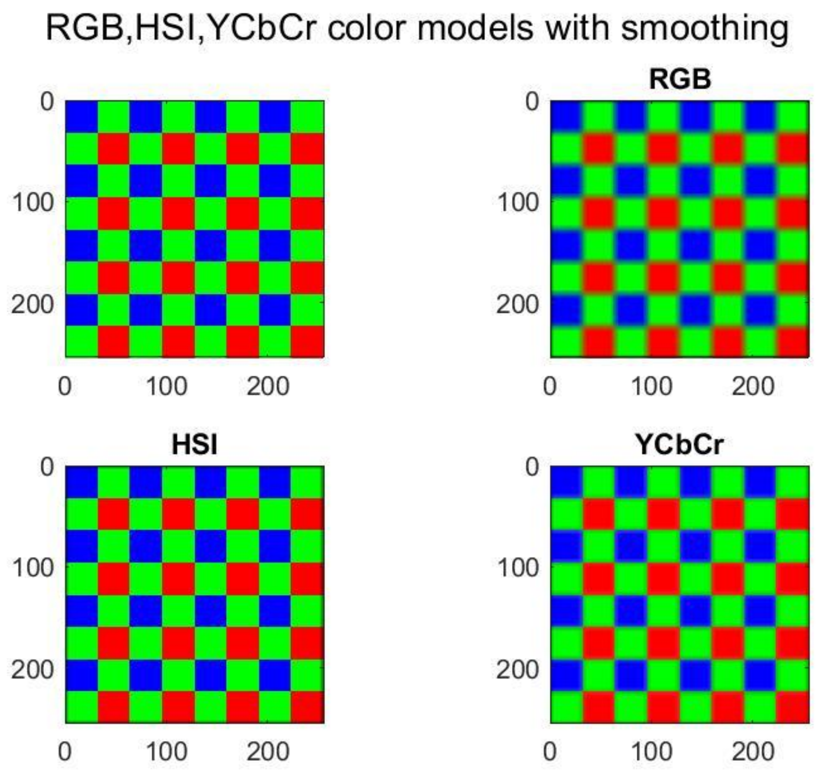


Figure 6: Bayer Pattern test image broken down into its YCbCr components.

Figure 7 shows each of the Bayer pattern images with color smoothing filters applied. We can see that between all of these color models, the RGB model was the most effective. Meanwhile the HSI model provided the least amount of color smoothing. The figure below also shows a close up in the color smoothing for each color model. We can see that the HSI color model has least color smoothing while RGB has many layers of smoothing that blend into the next color. YCbCr color model has a smoothing of that between RGB and HSI color model. Smoothing in the RGB color model reduces the overall sharpness of each color and they appear a little dull. In HSI color model, the extreme edges of the image appear to have zero padding. YCbCr smoothing is not as aggressive as RGB. Sharpness of each color is reduced slightly and the image doesn't appear to blurry compared to RGB.



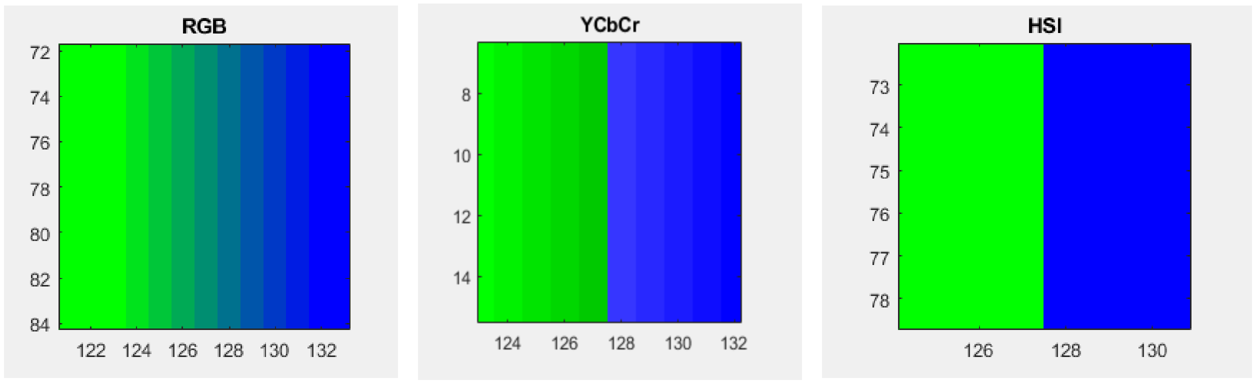


Figure 7: Bayer Pattern test image with each of the three color models after smoothing along with zoomed in change between green and blue.

Now that we've shown these color models when applied to simple test images. This process of breaking color models down to their components will be applied to more complex images. The overall steps to complete will still be the same. Figures 8 & 9 show an image named lena_color broken down into both the RGB and HSI color models. For the RGB model, the red component shows the brightest, due to the original image's orange complexity. We can see the green component is the darkest of the three since the majority of the image has a red complexion and the blue is brightest on the feathers of her hat. Compared to figure 6.38 from the textbook, R, G and B components in our results are lighter.

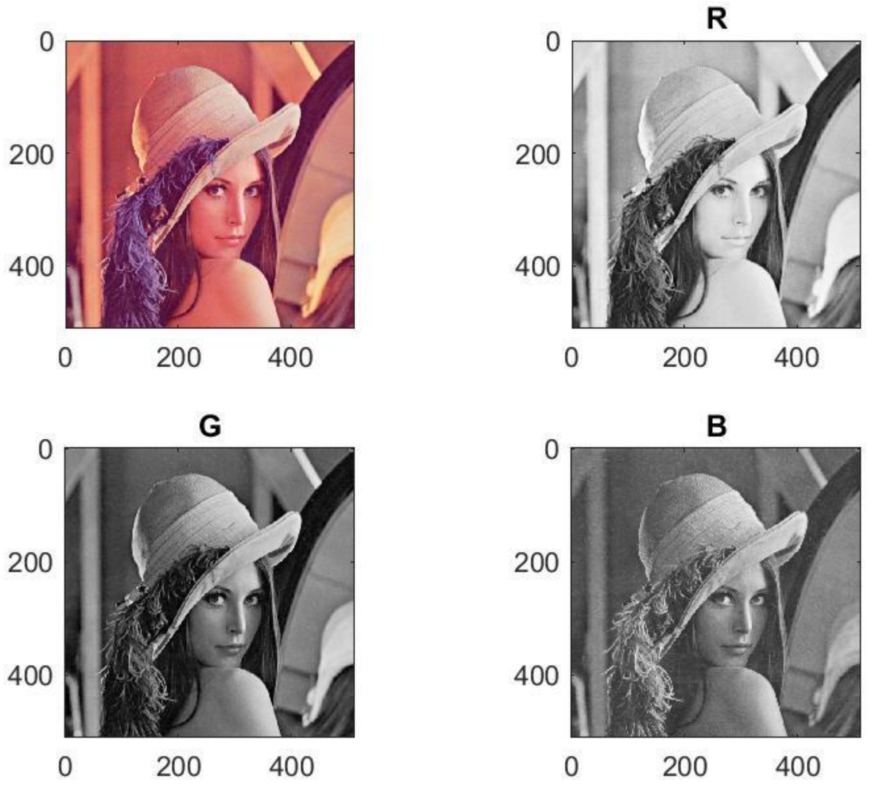


Figure 8: Lena Color image broken down into its RGB components

Figure 9 takes the same original image and splits it into its HSI components. We can see that the hue component emphasizes the shadows in the image. The saturation is much darker than intensity. Since the image isn't very saturated, there aren't any bright colors that pop in the image. There isn't needing to be as much saturation since the intensity is higher. Compared to figure 6.39 from the textbook, H and I components in our results are lighter and the S component is darker.

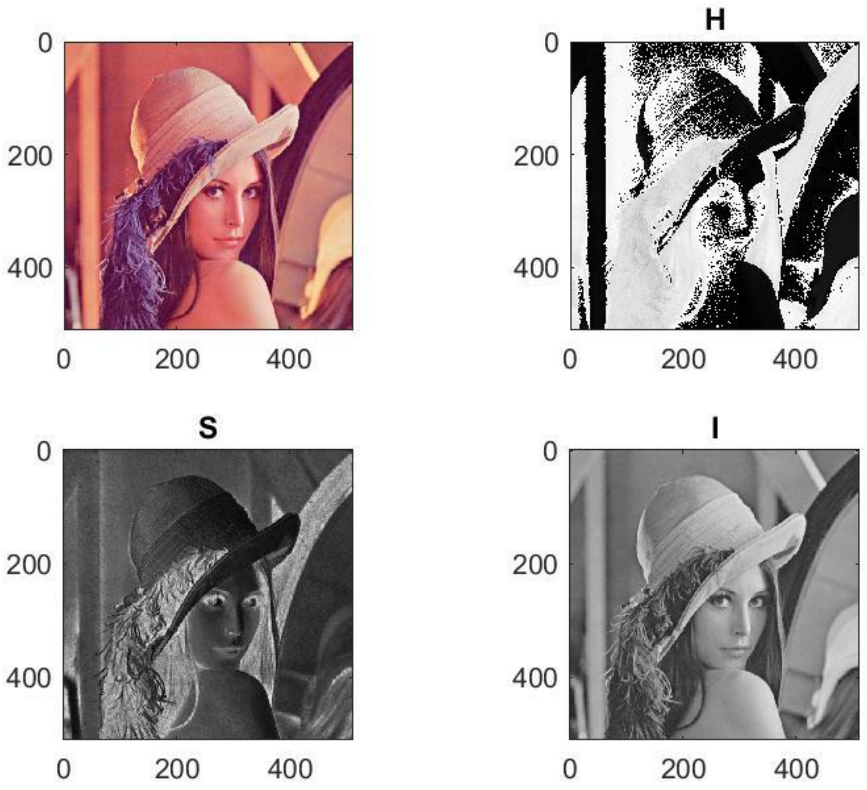


Figure 9: Lena Color image broken down into its HSI components.

The next figure demonstrates the enhancement of color on an image. The image shows more defined features and depth. There is a lot less unrealistic brightness in the image making her skin look less orange. It gives greater emphasis of shadow and color variety. The exact process is discussed in the methods section.

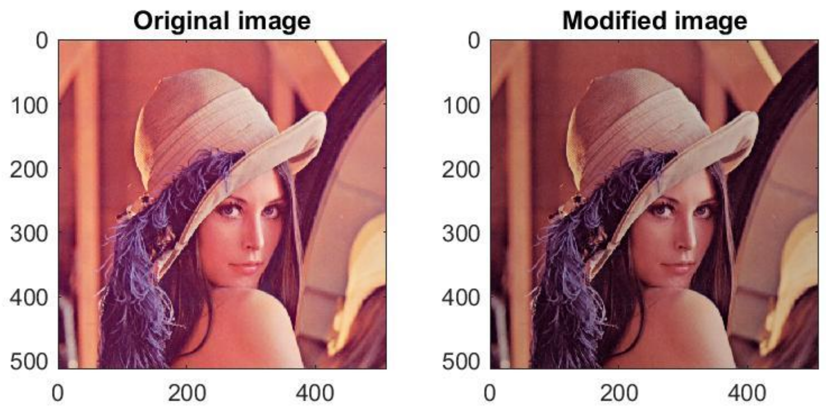


Figure 10: Lena Color image with color enhancement techniques to clear the image.

There are many more practical uses of applying these color models and the manipulation of images. A big example of this are techniques many use for green screen effects. Figure 11 shows both the composite image with a jet in the top right corner (left) and the final composited image of the jet flying over the base (right). We were needing to slightly alter the size of the jet to make the overall composited image appear more realistic. The exact process is described in the methods section.

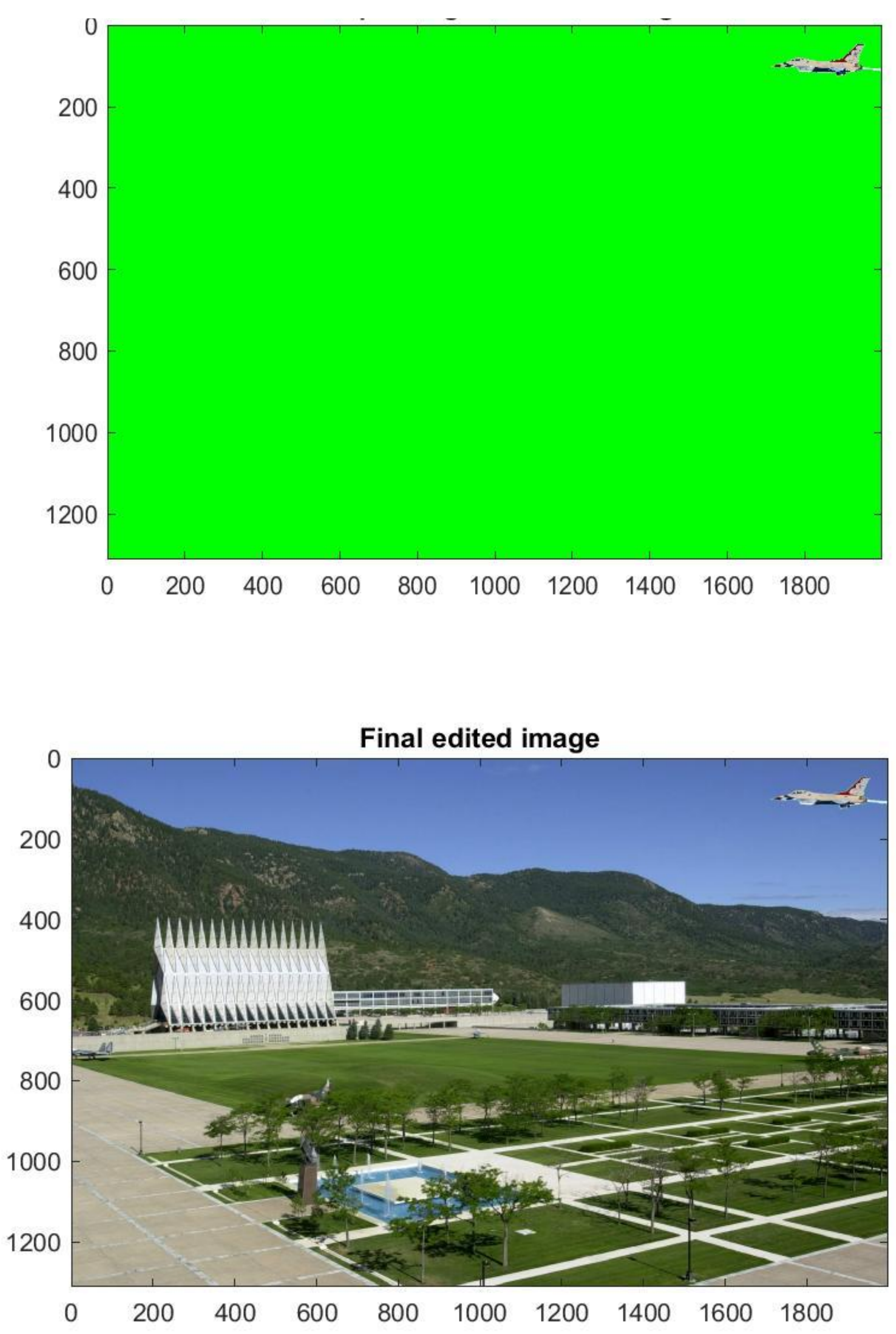


Figure 11: Green screen compositing image and the final composited image.

Conclusion:

Through this project we have applied different color models to the same image to observe the difference between them. With these fundamental color models, we used them to apply different methods of color smoothing, enhancement, as well as some green screen techniques. This project has taught us the different uses of the color models and how much of a difference each one can be.

Appendix:

RGB.m

```
% Creating an RGB image
image=zeros(256,256,3);
image(:,129:256,1)=255;
image(:,129:256,2)=255;
image(1:128,:,3)=255;
[X,Y,Z] = size(image);
V = image;

figure
subplot(2,2,1)
ip_dispc(image)
subplot(2,2,2)
ip_dispsc(image(:,:,1))
title('R')
subplot(2,2,3)
ip_dispsc(image(:,:,2))
title('G')
subplot(2,2,4)
ip_dispsc(image(:,:,3))
title('B')
print('RGB components','-djpeg')

%normalize the values
image(:,:,1) = image(:,:,1)/255; %R
image(:,:,2) = image(:,:,2)/255; %G
image(:,:,3) = image(:,:,3)/255; %B

%RGB to HSI conversion
[H,S,I] = RGB_HSI(image,256,256);
HSI = zeros(256,256,3);
HSI(:,:,1) = H;
HSI(:,:,2) = S;
HSI(:,:,3) = I;

H = H./(360); %normalise the value

figure
subplot(2,2,1)
ip_dispc(V)
subplot(2,2,2)
ip_disp(H.*255)
title('H')
subplot(2,2,3)
ip_disp(S.*255)
```

```
title('S')
subplot(2,2,4)
ip_disp(I.*255)
title('I')
print('HSI components','-djpeg')

% [r,g,b] = HSI_RGB(HSI,256,256);
% figure
% subplot(2,2,1)
% ip_dispc(V)
% subplot(2,2,2)
% ip_disp(r.*255)
% subplot(2,2,3)
% ip_disp(g.*255)
% subplot(2,2,4)
% ip_dispsc(b.*255)

%RGB to YCbCr conversion
[Y,Cb,Cr] = RGB_YCbCr(image);
YCbCr = zeros(256,256,3);
YCbCr(:,:,1) = Y;
YCbCr(:,:,2) = Cb;
YCbCr(:,:,3) = Cr;

Cb = Cb+0.5;
Cr = Cr+0.5;
figure
subplot(2,2,1)
ip_dispc(V)
subplot(2,2,2)
ip_disp(Y.*255)
title('Y')
subplot(2,2,3)
ip_disp(Cb.*255)
title('Cb')
subplot(2,2,4)
ip_disp(Cr.*255)
title('Cr')
print('YCbCr components','-djpeg')

% [r1,g1,b1] = YCbCr_RGB(YCbCr);
% figure
% subplot(2,2,1)
% ip_dispc(V)
% subplot(2,2,2)
% ip_disp(r1.*255)
% subplot(2,2,3)
% ip_disp(g1.*255)
% subplot(2,2,4)
% ip_disp(b1.*255)
```

bayer.m

```
X = zeros(64,64,3);
%Creating a small portion of bayer image
X(33:64,1:32,2) = 255;
X(1:32,33:64,2) = 255;
```

```
X(1:32,1:32,3) = 255;
X(33:64,33:64,1) = 255;
%9x9 unweigthed filter
f = ones(9);
f = f.*(1/81);

%Creating bayer pattern using repmat
Z = repmat(X,4,4);
Z1 = Z./255;

figure
subplot(2,2,1)
ip_dispc(Z)
subplot(2,2,2)
ip_dispsc(Z(:,:,1))
title('R')
subplot(2,2,3)
ip_dispsc(Z(:,:,2))
title('G')
subplot(2,2,4)
ip_dispsc(Z(:,:,3))
title('B')
print('Bayer RGB components','-djpeg')

%RGB to HSI conversion
[H,S,I] = RGB_HSI(Z1,256,256);
h = H; s = S; i = I;
H = H./(2*pi);
figure
subplot(2,2,1)
ip_dispc(Z)
subplot(2,2,2)
ip_disp(H.*255)
title('H')
subplot(2,2,3)
ip_disp(S.*255)
title('S')
subplot(2,2,4)
ip_disp(I.*255)
title('I')
print('Bayer HSI components','-djpeg')

%Applying smoothing filter to RGB image
R = ib_filt(Z(:,:,1),f);
G = ib_filt(Z(:,:,2),f);
B = ib_filt(Z(:,:,3),f);
RGB = zeros(256,256,3);
RBG(:,:,1) = R;
RBG(:,:,2) = G;
RBG(:,:,3) = B;

%Applying smoothing filter to HSI image
I1 = ib_filt(i,f);
HSI = zeros(256,256,3);
HSI(:,:,1) = h;
HSI(:,:,2) = s;
HSI(:,:,3) = I1;
%HSI to RGB conversion
[R1,G1,B1] = HSI_RGB(HSI,256,256);
RGB1 = zeros(256,256,3);
RBG1(:,:,1) = R1.*255;
RBG1(:,:,2) = G1.*255;
RBG1(:,:,3) = B1.*255;

%RGB to YbCr conversion
[Y,Cb,Cr] = RGB_YCbCr(Z1);
YCbCr = zeros(256,256,3);
%Applying smoothing filter to YCbCr image
y1 = ib_filt(Y,f);
YCbCr(:,:,1) = y1;
YCbCr(:,:,2) = Cb;
YCbCr(:,:,3) = Cr;

%Adjusting the range to [0 1]
Cb = Cb+0.5;
Cr = Cr+0.5;
figure
subplot(2,2,1)
ip_dispc(Z)
subplot(2,2,2)
ip_disp(Y.*255)
title('Y')
subplot(2,2,3)
ip_disp(Cb.*255)
title('Cb')
subplot(2,2,4)
ip_disp(Cr.*255)
title('Cr')
print('Bayer YCbCr components','-djpeg')

%YCbCr to RGB conversion
[R2,G2,B2] = YCbCr_RGB(YCbCr);
RGB2 = zeros(256,256,3);
RGB2(:,:,1) = R2.*255;
RGB2(:,:,2) = G2.*255;
RGB2(:,:,3) = B2.*255;

figure
subplot(2,2,1)
ip_dispc(Z)
subplot(2,2,2)
ip_dispc(RBG)
title('RGB')
subplot(2,2,3)
ip_dispc(RBG1)
title('HSI')
subplot(2,2,4)
ip_dispc(RGB2)
title('YCbCr')
suptitle('RGB,HSI,YCbCr color models with smoothing')
print('RGB,HSI,YCbCr color models with smoothing','-djpeg')
```

lena.m

```
Z = imread('lena_color_512.tif');
Z = double(Z);
figure
subplot(2,2,1)
ip_dispc(Z)
subplot(2,2,2)
ip_dispsc(Z(:,:,1))
title('R')
subplot(2,2,3)
ip_dispsc(Z(:,:,2))
title('G')
subplot(2,2,4)
ip_dispsc(Z(:,:,3))
title('B')
print('Lena RGB components','-djpeg')

A1 = Z;

%normalize the values
A1(:,:,1) = A1(:,:,1)/255; %R
A1(:,:,2) = A1(:,:,2)/255; %G
A1(:,:,3) = A1(:,:,3)/255; %B

%RGB to HSI conversion
[H,S,I] = RGB_HSI(A1,512,512);
HSI = zeros(512,512,3);
%Adjusting the skin tone of the image
I1 = ((I.*255) - 35)./(255);
HSI(:,:,1) = H;
HSI(:,:,2) = S;
HSI(:,:,3) = I1;
H = H./(360);

figure
subplot(2,2,1)
ip_dispc(Z)
subplot(2,2,2)
ip_disp(H.*255)
title('H')
subplot(2,2,3)
ip_disp(S.*255)
title('S')
subplot(2,2,4)
ip_disp(I.*255)
title('I')
print('Lena HSI components','-djpeg')

%HSI to RGB conversion
[R,G,B] = HSI_RGB(HSI,512,512);
rgb = zeros(512,512,3);
rgb(:,:,1) = R.*255;
rgb(:,:,2) = G.*255;
rgb(:,:,3) = B.*255;

figure
```

```
subplot(1,2,1)
ip_dispc(Z)
title('Original image')
subplot(1,2,2)
ip_dispc(rgb)
title('Modified image')
print('Skin tone modified image of Lena','-djpeg')
```

gtest.m

```
load green_screen.mat
A = AFA;
B = double(F16./255);
% Smoothing filter
F4 = ones(3);
F4 = F4*(1/9);

% Interpolating the green screen image to have same dimension as background
B1 = ib_bi_int(F16(:,:,1),1312,2000);
B2 = ib_bi_int(F16(:,:,2),1312,2000);
B3 = ib_bi_int(F16(:,:,3),1312,2000);
% Scaling the green screen image
S1 = ip_scale(uint8(B1),0.15,0.15);
S2 = ip_scale(uint8(B2),0.15,0.15);
S3 = ip_scale(uint8(B3),0.15,0.15);
% Translating the green screen image to appropriate position
T1 = ib_translate(S1,0,1701);
T2 = ib_translate(S2,0,1701);
T3 = ib_translate(S3,0,1701);
T = zeros(1312,2000,3);
T(:,:,1) = T1;
T(:,:,2) = T2;
T(:,:,3) = T3;
T(1:1312,1:1702,2) = 255;
T(196:1312,:,2) = 255;

% Performing sphere splicing
c = [10, 250, 10]; % RGB components
r = 45; % Radius
distance = zeros(1312,2000);
for i=1:1312
   for j=1:2000
      distance(i,j) =  (T(i,j,1) - c(1)).^2 + (T(i,j,2) - c(2)).^2 + (T(i,j,3) - c(3)).^2;
      if distance(i,j) > r^2 % not green
         A(i,j,1) = T(i,j,1);
         A(i,j,2) = T(i,j,2);
         A(i,j,3) = T(i,j,3);
      end
   end
end

% Applying smoothing filter
f1 = ib_filt(A(:,:,1),F4);
f2 = ib_filt(A(:,:,2),F4);
f3 = ib_filt(A(:,:,3),F4);

RGB(:,:,1) = f1;
RGB(:,:,2) = f2;
RGB(:,:,3) = f3;

figure
ip_dispc(T)
title('Composite green screen image')
print('Composite green screen image','-djpeg')

figure
ip_dispc(RGB)
title('Final edited image')
print('Final edited image','-djpeg')
```

RGB_HSI.m

```
function [H,S,I] = RGB_HSI(image,X,Y)

H = zeros(X,Y);
S = zeros(X,Y);
I = zeros(X,Y);
R = image(:,:,1);
G = image(:,:,2);
B = image(:,:,3);
for i=1:X
    for j=1:Y
        a = 0.5*((R(i,j)-G(i,j))+(R(i,j)-B(i,j)));
        b = ((R(i,j)-G(i,j)).^2 + ((R(i,j)-B(i,j))*(G(i,j)-B(i,j)))).^0.5;
        if a == 0 && b == 0
            Ab = 0;
        else
            Ab = (a/b);
        end
        theta = acosd(Ab);
        if B(i,j)<=G(i,j)
            H(i,j) = theta;
        elseif B(i,j)>G(i,j)
            H(i,j) = 360 - theta;
        end

        c = min(R(i,j),G(i,j));
        c = min(B(i,j),c);
        S(i,j) = 1 - 3*(c/(R(i,j)+G(i,j)+B(i,j)));

        I(i,j) = (R(i,j)+G(i,j)+B(i,j))/3;
    end
end
```

RGB_YCbCr.m

```
function [Y,Cb,Cr] = RGB_YCbCr(image)

R = image(:,:,1);
G = image(:,:,2);
B = image(:,:,3);

Y = 0.2988*R + 0.5869*G + 0.1143*B;
Cb = -0.1689*R - 0.3311*G + 0.5000*B;
```

```
Cr = 0.5000*R - 0.4189*G - 0.0811*B;
```

HSI_RGB.m

```
function [R,G,B] = HSI_RGB(A,X,Y)
R = zeros(X,Y);
G = zeros(X,Y);
B = zeros(X,Y);
H = A(:,:,1);
S = A(:,:,2);
I = A(:,:,3);

for i=1:X
   for j=1:Y
      if (0<=H(i,j))&&(H(i,j)<120)
         R(i,j) = I(i,j)*(1+(S(i,j)*cosd(H(i,j))/cosd(60-H(i,j))));
         B(i,j) = I(i,j)*(1 - S(i,j));
         G(i,j) = 3*I(i,j) - (R(i,j) + B(i,j));

      elseif (120<=H(i,j))&&(H(i,j)<240)
         H(i,j) = H(i,j) -120;
         R(i,j) = I(i,j)*(1 - S(i,j));
         G(i,j) = I(i,j)*(1+(S(i,j)*cosd(H(i,j))/cosd(60-H(i,j))));
         B(i,j) = 3*I(i,j) - (R(i,j) + G(i,j));

      elseif (240<=H(i,j))&&(H(i,j)<360)
         H(i,j) = H(i,j) - 240;
         G(i,j) = I(i,j)*(1 - S(i,j));
         B(i,j) = I(i,j)*(1+(S(i,j)*cosd(H(i,j))/cosd(60-H(i,j))));
         R(i,j) = 3*I(i,j) - (G(i,j) + B(i,j));
      end
   end
end
```

YCbCr_RGB.m

```
function [R,G,B] = YCbCr_RGB(A)

Y = A(:,:,1);
Cb = A(:,:,2);
Cr = A(:,:,3);

R = Y + 1.402*Cr;
G = Y - 0.3441*Cb -0.7141*Cr;
B = Y + 1.772*Cb + 0.00015*Cr;
```

ib_bi_int.m

```
function C = ib_bi_int (IN, xnew, ynew)

[xog,yog] = size(IN); % Size of the input image

Ax = (xnew-1)/(xog-1);

B = zeros(xnew,yog); % Intermediate matrix

% Interpolating between rows for each column
for i=1:yog
   B(1,i) = IN(1,i); % setting the endpoint
   B(xnew,i) = IN(xog,i); % setting the endpoint

   for j=2:xnew-1
      Xd = (j-1)/Ax; % X' where j starts from 0
      Xd = Xd + 1; % Changing index to start from 1
      Xdf = floor(Xd);
      dx = Xd - Xdf;
      B(j,i) = IN(Xdf,i)*(1-dx) + IN(Xdf+1,i)*dx;
   end
end

Ay = (ynew-1)/(yog-1);

C = zeros(xnew,ynew); % Intermediate matrix

% Interpolating between columns for each row
for i=1:xnew
   C(i,1) = B(i,1); % setting the endpoint
   C(i,ynew) = B(i,yog); % setting the endpoint

   for j=2:ynew-1
      Yd = (j-1)/Ay; % Y' where j starts from 0
      Yd = Yd + 1; % Changing index to start from 1
      Ydf = floor(Yd);
      dy = Yd-Ydf;
      C(i,j) = B(i,Ydf)*(1-dy) + B(i,Ydf+1)*dy;
   end
end

C = round(C); % rounding the final value


% figure
% ip_disp(IN); % display
% title("Original Image");
% print('Original Image','-djpeg');
%
% figure
% ip_disp(C); % display
% title("1200 x 1200 Interpolated Image");
% print('1200 x 1200 Interpolated Image','-djpeg');

% print('Disk','-djpeg');
% print('100 x 100 Interpolated Image of Disk','-djpeg');
```

ib_filt.m

```
%Filter mask
function O = ib_filt(I,F)
%  I = [1 2 3 3; 4 5 6 3; 7 8 9 3; 2 3 4 5];
%  F = [1 1 1; 1 1 1; 1 1 1];
%  F = F*(1/9);
[Ix,Iy] = size(I); % Image size
[Fx,Fy] = size(F); % Filter size

zpad = (Fx - 1);
zIx = zpad + Ix;
```

```
zIy = zpad + Iy;

zI = zeros(zIx,zIy);
X = zpad/2;
zI(1+X:zIx-X,1+X:zIy-X)=I; % Zero padding the image
O = zeros(Ix,Iy); % Output matrix

for i=1:Ix
   for j=1:Iy
      for k=1:Fx
         for l=1:Fy
            O(i,j) = O(i,j) + (zI(i+k-1,j+l-1)*F(k,l));
         end
      end
   end
end

% figure()
% % ip_disp(I);
% imshow(I)
% % title("iptest256a");
% % print('iptest256a','-djpeg');
% %
% figure()
% % ip_disp(abs(O));
% imshow(O)
% title("iptest256a _ filtered");
% print('iptest256a _ filtered','-djpeg');
```

ib_translate.m

```
function B = ib_translate (IN, tx, ty)
% IN = imread('Proj_002_image2.tif');
% load lena
% IN = A512;
% tx = 150.75;
% ty = -67.3;
[xog,yog] = size(IN); % size of the input image
B = zeros(xog,yog); % matrix to store final image

for i=1:xog
   for j=1:yog
      Xd = i - tx; % transformed X coordinates
      Yd = j - ty; % transformed Y coordinates
      if(Yd>1 && Yd<yog && Xd>1 && Xd<xog) % Discarding values exceeding the input image dimension

         %Performing bilinear interpolation
         Ydf = floor(Yd);
         Xdf = floor(Xd);
         dx = Xd - Xdf;
         dy = Yd - Ydf;
         B(i,j) = (1-dy)*[(1-dx)*IN(Xdf,Ydf)+dx*IN(Xdf+1,Ydf)] +
(dy)*[(1-dx)*IN(Xdf,Ydf+1)+dx*IN(Xdf+1,Ydf+1)];
      end
   end
end
```

```
B = round(B); % rounding the final value

% figure
% ip_disp(IN); % display
% title("Original Image");
% print('Original Image','-djpeg');
%
% figure
% ip_disp(B); % display
% title("Translated Image");
% print('Translated Image','-djpeg');
```

ip_scale.m

```
function OUT = ip_scale(IN, cx, cy)
%
% This function performs image scaling using the affine transformation
% matrix.  It uses the inverse mapping approach and bilinear interpolation
% on a 2-D image to produce a scaled image of the same size
% as the original.  Since the size stays the same, some cropping of the
% original image may occur.
%
% Usage: OUT=ip_scale(IN, cx, cy)
%            where IN is a 2-D image 8-bit grayscale,
%            cx and cy are scale factors.
%
% The matrix generated is returned as OUT.
%
% Copyright (c) 2011-2018 Cameron H. G. Wright
%
if nargin~=3
   error('Image matrix, cx and cy required as inputs');
end;

if (isvector(IN) || ndims(IN) > 2)
   error('Input must be a grayscale image matrix');
end;

if ~(isa(IN,'uint8'))
   error('Image matrix must be uint8');
end;

[xold, yold] = size(IN); % get size of image
IN=double(IN); % so math is correct on the pixel values

B=zeros(xold,yold);  % preallocate matrix same size as original
vw=zeros(1,3); % preallocate pixel vector

% create the affine transformation matrix for rotation
A=[ cx     0     0;
    0      cy    0;
    0      0     1];

% find the inverse matrix
Ai=inv(A);

% step through every pixel location in the output image (i,j) and find the
% associated location in the input image (v,w), then perform bilinear
% interpolation as needed to assign the gray level for the output pixel
% location
for i=0:xold-1
   for j=0:yold-1
      % matrix math assumes origin is (0,0) not (1,1)
      vw=Ai*[i; j; 1]; % vw will be [v; w; 1]
      % next 2 lines for debugging only
%        s=sprintf('(x,y) = (%g,%g)  (v,w) = (%g,%g)',i,j,vw(1),vw(2));
%        disp(s);
      % extract v and w from vector for easier to use name
      % add 1 to get back to MATLAB indexing
      v=vw(1)+1;
      w=vw(2)+1;
      % skip if out of range location
      if ~((v < 1) || (v > xold) ||(w < 1) || (w > yold))
         vint=fix(v);
         dv=v-vint;
         wint=fix(w);
         dw=w-wint;
         % perform bilinear interpolation
         % special cases for endpoints of image A
         if (vint < xold) && (wint < yold)
            temp1=(1-dv)*IN(vint,wint)+(dv)*IN(vint+1,wint);
            temp2=(1-dv)*IN(vint,wint+1)+(dv)*IN(vint+1,wint+1);
            B(i+1,j+1)=(1-dw)*temp1+(dw)*temp2;
         elseif (vint == xold) && (wint < yold)
            temp1=IN(vint,wint);
            temp2=IN(vint,wint+1);
            B(i+1,j+1)=(1-dw)*temp1+(dw)*temp2;
         elseif (vint < xold) && (wint == yold)
            temp1=(1-dv)*IN(vint,wint)+(dv)*IN(vint+1,wint);
            B(i+1,j+1)=temp1;
         elseif (vint == xold) && (wint == yold)
            B(i+1,j+1)=IN(vint,wint);
         end
      end
   end
end


B=round(B);  % round to nearest integer
B=uint8(B);  % convert to 8-bit unsigned integer

% optional figures depending upon how the function was called
if nargout == 0 % then show results as figures
   close all
   figure(1)
   subplot(1,2,1)
   ip_disp(IN)
   title('original image')
   subplot(1,2,2)
   ip_disp(B)
   title('scaled image')
else % provide output matrix
   OUT=B;
```

```
end
```

ip_disp.m

```
function ip_disp(A)
% simplified routine to display 8-bit gray scale images
% Syntax: ip_disp(A)
% where A is an image matrix, typically uint8
%
% Copyright 2004-2008 Cameron H. G. Wright

L=256;
L1=L-1;
A=double(A); % just in case
[X Y]=size(A);
% switch X and Y for proper image display
% (0,0) is in UL corner with X rows and Y columns
image(0:Y-1,0:X-1,A)
axis ij
axis equal
axis tight
colormap(gray(L))
end
```

ip_dispsc.m

```
function ip_dispc(A)
% simplified routine to display 24-bit color images
% Syntax: ip_dispc(A)
% where A is an NxMx3 image matrix, typically uint8
% Forces A to uint8 for display purposes, so pixel
% values need to be on 0-255 range.
%
% Copyright 2004-2011 Cameron H. G. Wright

L=256;
L1=L-1;
A=uint8(A); % just in case
[X Y Z]=size(A);
% switch X and Y for proper image display
% (0,0) is in UL corner with X rows and Y columns
image(0:Y-1,0:X-1,A)
axis ij
axis equal
axis tight
end
```

ip_dispc.m

```
function ip_dispsc(A)
% simplified routine to display 8-bit gray scale images
% scales image to full range of display
% Syntax: ip_dispsc(A)
% where A is an image matrix, typically uint8


L=256;
L1=L-1;
A=double(A); % just in case
```

```
minA=min(min(A));
A=A-minA; % shift
maxA=max(max(A));
if maxA~=0
   A=A*L1/maxA; % compress or expand
end
[X Y]=size(A);
% switch X and Y for image coordinates
image(0:Y-1,0:X-1,A)
axis ij
axis equal
axis tight
colormap(gray(L))
caxis([0 L1])
```

References:

[1] Hu, L., Bai, X., Yang, A., Zhang, K., Zhang, C., & Liu, B. (2020, February). Crop extraction based on ultra-simple neural network modeling in the normalized rgb and CIE L* a* b* color spaces. In *MIPPR 2019: Pattern Recognition and Computer Vision* (Vol. 11430, p. 114301V). International Society for Optics and Photonics.

[2] Mousavi, S. M. H., Lyashenko, V., & Prasath, S. (2019). Analysis of a robust edge detection system in different color spaces using color and depth images. *Компьютерная оптика*, *43*(4).

[3] Ji, J., Fang, S., Shi, Z., Xia, Q., & Li, Y. (2020). An efficient nonlinear polynomial color characterization method based on interrelations of color spaces. *Color Research & Application*, *45*(6), 1023-1039.

[4] Nasrin, S., Alom, M. Z., Taha, T. M., & Asari, V. K. (2020, March). PColorNet: investigating the impact of different color spaces for pathological image classification. In *Medical Imaging 2020: Digital Pathology* (Vol. 11320, p. 113201A). International Society for Optics and Photonics.

[5] Chien, C. L., & Tseng, D. C. (2011). Color image enhancement with exact HSI color model. *international journal of innovative computing, information and control*, *7*(12), 6691-6710.

[6] Wright, Cameron H.G.. “*Advanced Image Processing*” EE5630. University of Wyoming, March. 2021, Lecture, Notes, Sample code.