ECE572 – Digital Image Processing

Project5 – Color Image Processing

Name: Sang-hyeb(Sam) Lee
NetID: slee91
Student ID: 000330428

98/100

Abstract

The objectives of this project are to examine and apply various image processing techniques on color images. The result shows that we can apply the same image processing techniques to the color images as we applied to the gray-scale images.

TASK0) Short Answer Problems(15pts)

Question 1) Why does CIE standard specify R,G,B as the primary colors?

Answer) Since 1930s, it has been long assumed that retina contains three different types of cone receptors with different absorption spectra. For example, 65% of cones are sensitive to red light, 33% are sensitive to green light, and 2% are sensitive to blue light. Even though that this assumption has proven known to be incorrect. For this reason, CIE specifies R,G,B as the primary colors.

Question 2) Why does Bayer color filter array have 50% green but 25% red and blue?

Human eye is more sensitive to green color; therefore, there are twice as many green element as red or blue to closely mimic the physiology of the human eye.

Question 3) What is chromaticity diagram? Tristimulus? Why can't the three primary colors generate all the visible colors specified in the diagram? Where is brown?

Tristimulus values of a color are the amounts of red, green, and blue in a three-component additive color model needed to form that particular color.

Chromaticity diagram represents all the chromaticities visible to the average person. It shows color composition as a function of x (red) and y (green). For any value of x and y, the corresponding value of z(blue) can be computed using the equation z = 1-(x+y).

Because chromaticity diagram only shows the dominant hue and the saturation, and is independent of the brightness.

You can get the color brown by reducing the intensity of red.

Question 4) Comment on the different usage of RGB, CMY, YIQ, HSI color models.

Color models	Usage
RGB	Devices which use addictive color system: Color monitor and video cameras.
CMY	Devices which use subtractive color system:Color printers and copiers require CMY data input.
YIQ	Used by NTSC color TV system. It is currently in use only for low-power television stations.
HSI	Useful when describing colors in terms which are practical for human interpretation. Often used in color image manipulation software.

Question 5) Read related sections in our textbook and find out what "safe color" means.

Safe color is s subset of colors that can be reproduced faithfully, reasonably independently of viewer hardware capabilities. For example, there are safe web colors or safe browser colors for Internet applications.

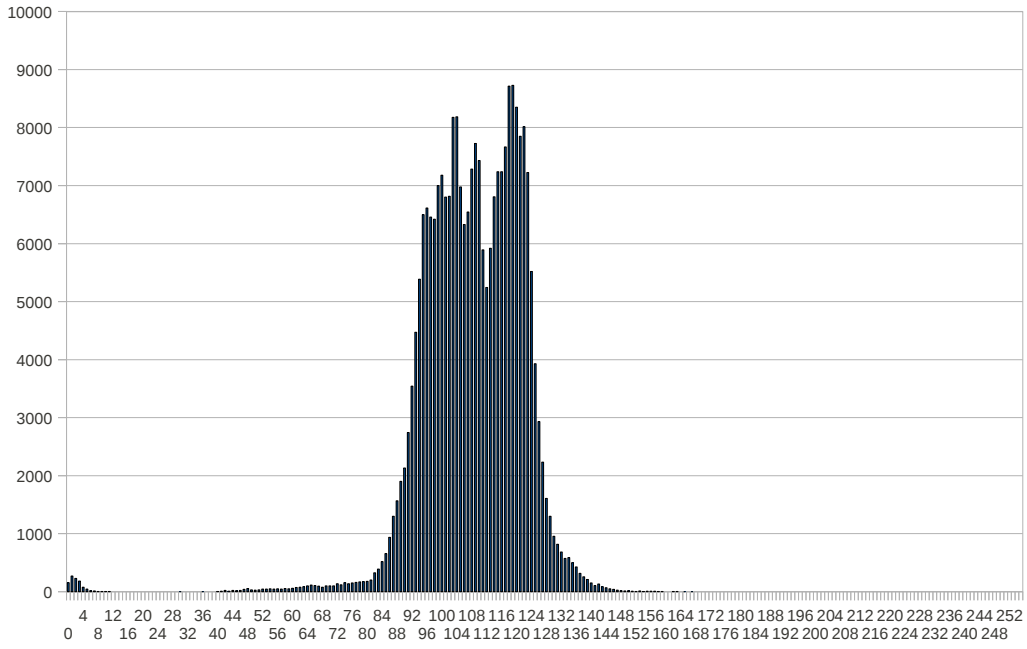
Task1) Pseudo-color processing (30 pts.)

Below is a gray-scale image used in task1.

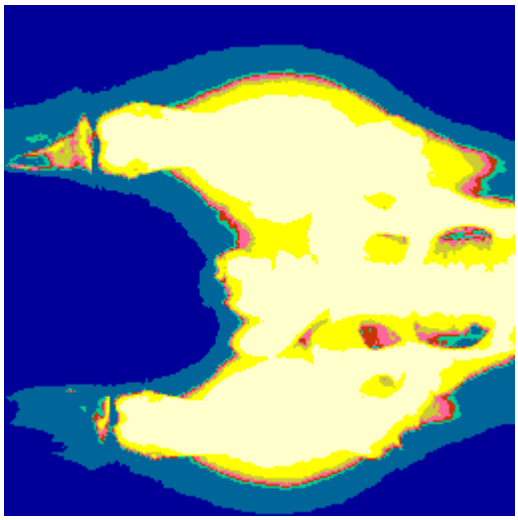
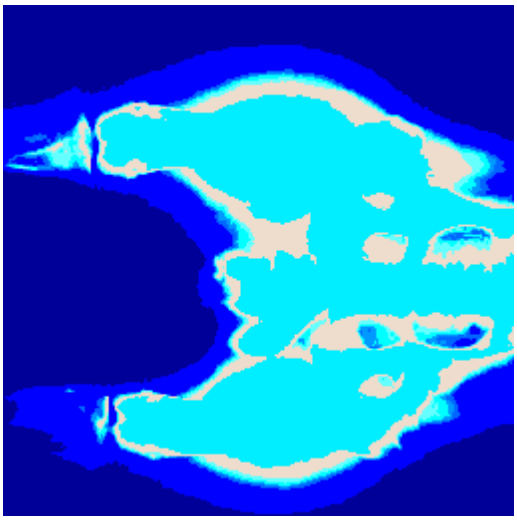
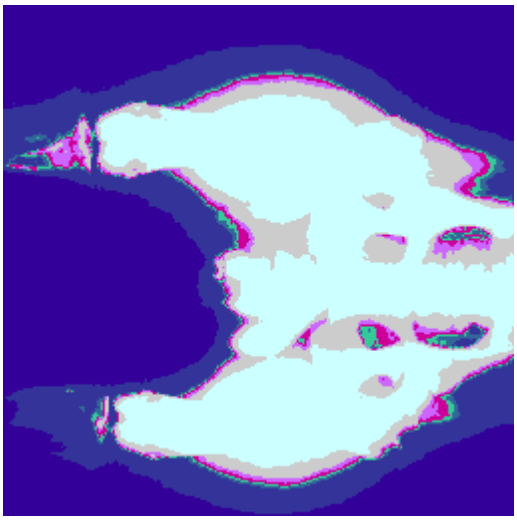
(Original Image: dogbonesquare.pgm)

Question 1) Transform this image to a color image using intensity slicing(choose as many color levels as you think is appropriate)

To choose appropriate color levels for the image, I worked out the histogram of the image as below:



I implemented *Image intensityslicing(Image& inimg)* function in *colorimageprocessing.cpp* file. Here are the images after intensity slicing.

how many color did you use? need more explain
-1

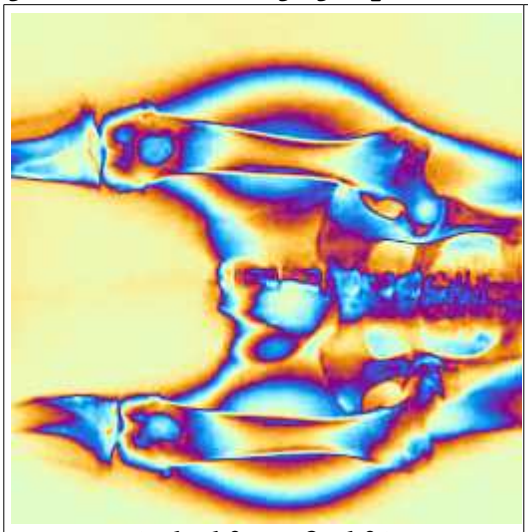
Question 2) Transform the same image to a color image using gray level to color transformation method in the spatial domain.

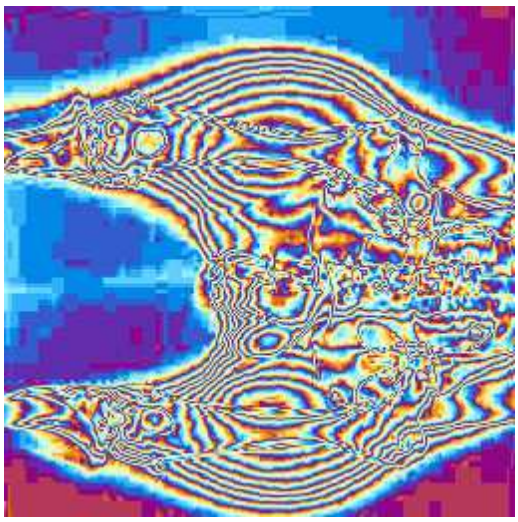
I used the following formula to transform the image to a color image:

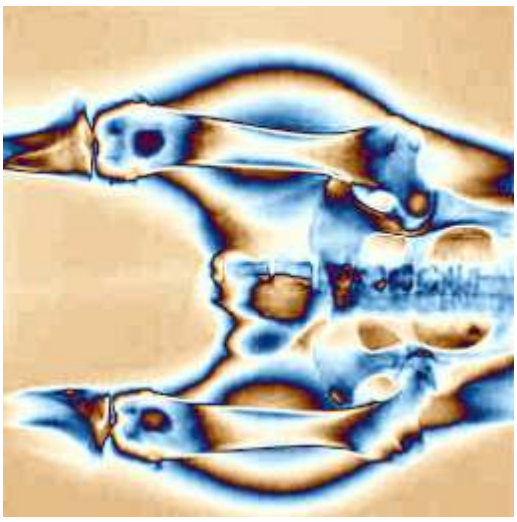
$$\text{red} = L * |(\sin(s * p1 / L))|$$
$$\text{blue} = L * |(\sin(s * p1 / L + p2))|$$
$$\text{green} = L * |(\sin(s * p1 / L + p2 * 2))|$$

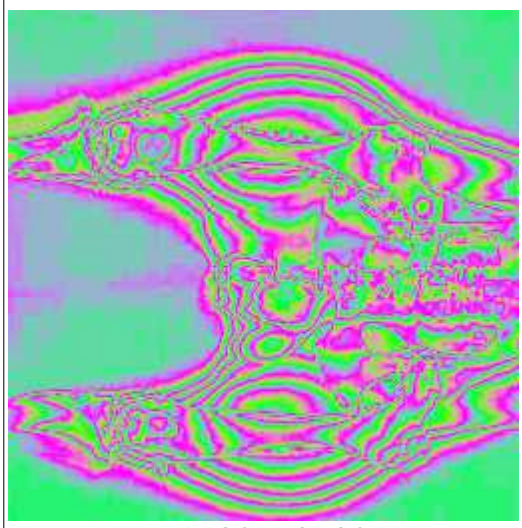
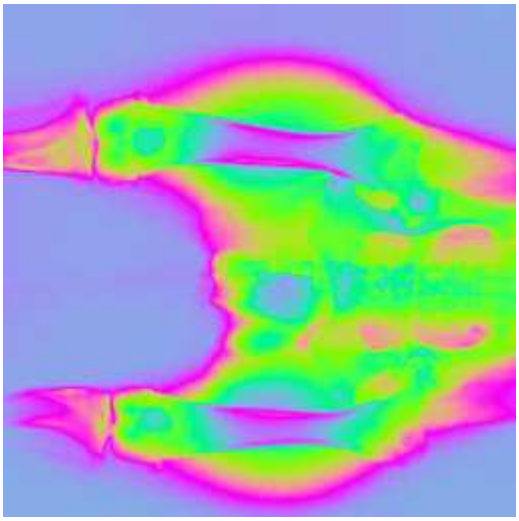
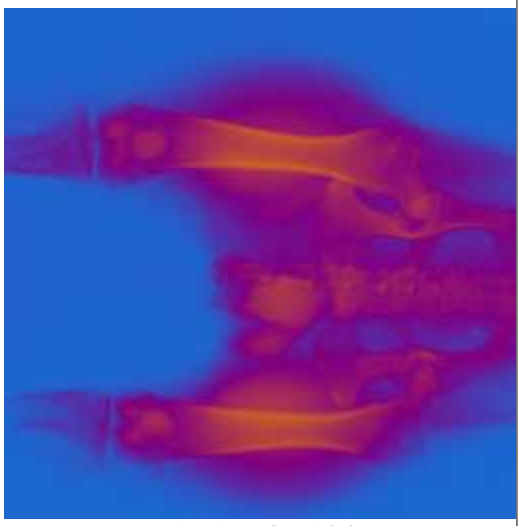
where s is the intensity in the original image.
P1 changes the frequency.
P2 changes the phase.

This transformation method is implemented in *Image greytocolorsp(Image& inimg, float p1, float p2)* in *colorimageprocessing.cpp* file. Here are the images generated using gray level to color transformation method in the spatial domain.


(p1=10, p2=10)


(p1=50, p2=10)


(p1=10,p2=50)

 (p1=30,p2=30)	 (p1=5,p2=5)	 (p1=1,p2=100)

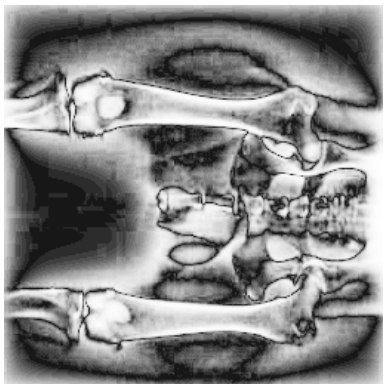
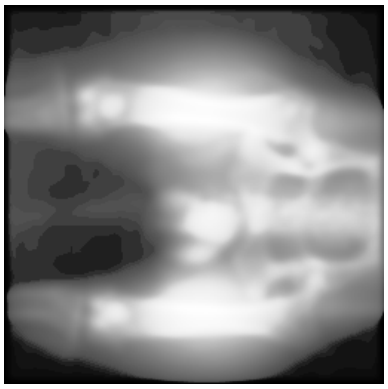
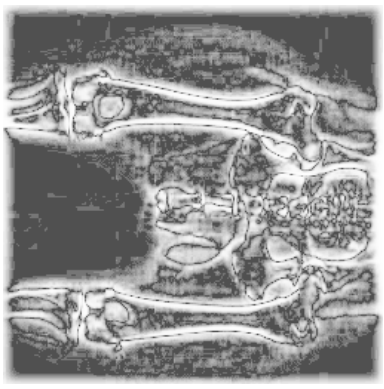
Question 3) Transform the same image to a color image using gray level to color transformation method in the frequency domain.

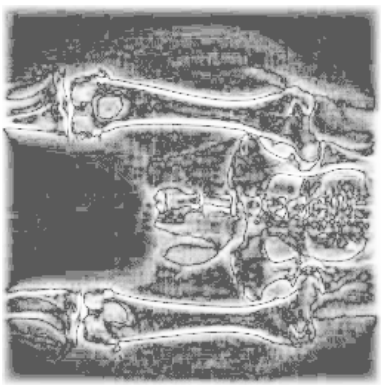
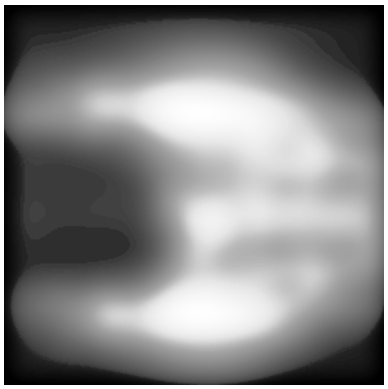
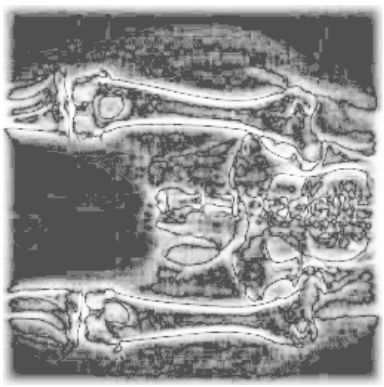
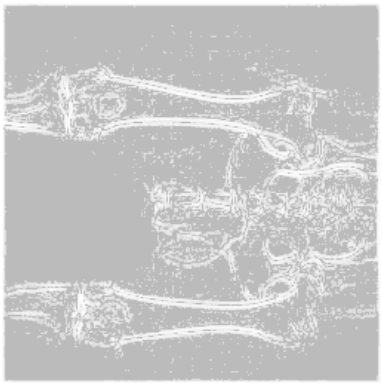
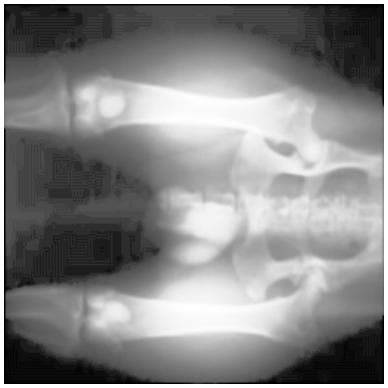
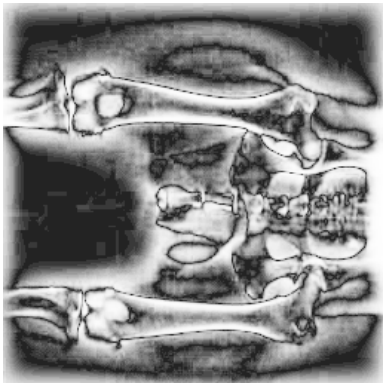
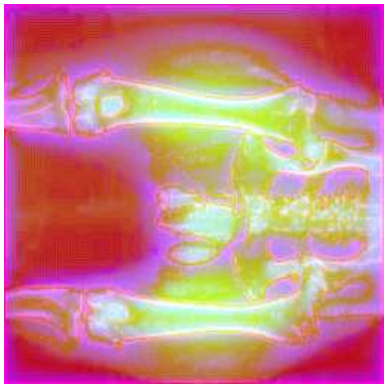
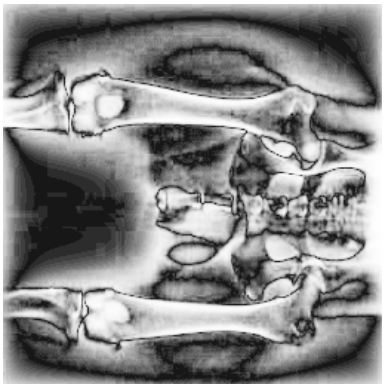
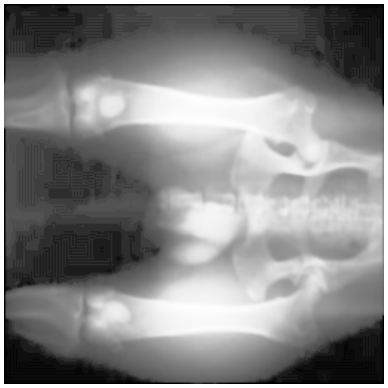
I used the following filters for each RGB component:

```
Red: high pass butterworth filter
Green: high pass gaussian filter
Blue: low pass gaussian filter
```

This transformation method is implemented in *Image greytocolorfreq(Image& inimg, float p1, float p2, float p3)* in *colorimageprocessing.cpp* file. P1 specifies a radius(D_0) for the filter for red, P2 specifies a radius(D_0) for the filter for blue, and P3 specifies a radius for the filter for green.

Here are the images:

RED	**GREEN**	**BLUE**	**FINAL IMAGE**
			 **(P1=10,P2=20,P3=30)**

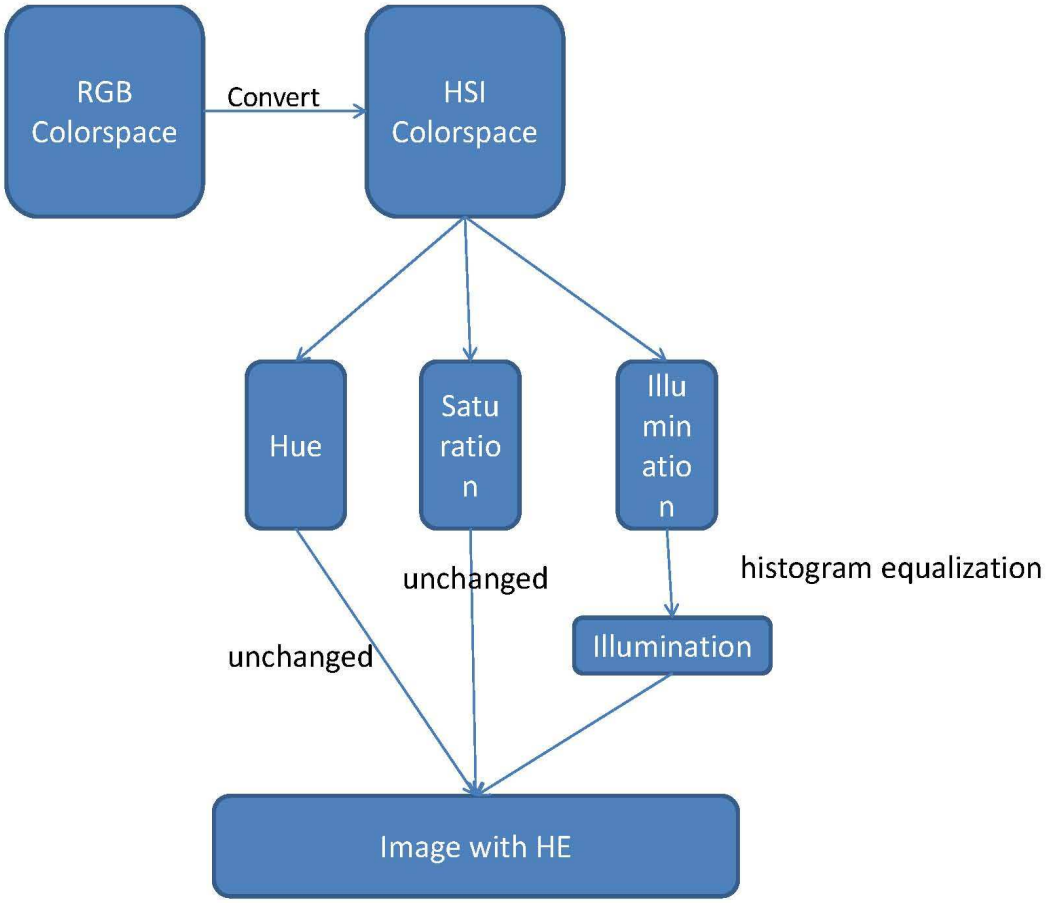
			 (P1=30,P2=20,P3=10)
			 (P1=100,P2=10,P3=100)
			 (P1=10,P2=100,P3=100)

Images in column 1, 2, and 3 shows red, green, and blue components respectively.
Images in column 4 represents the final images with all RGB components.

TASK2) Color image processing (40pts)

Question1) Use histogram equalization to enhance the flower image.

Here is a block diagram explaining the procedures I took to perform histogram equalization of color image.



Basically, I converted the color space from RGB to HSI using RGB2HSI function and then perform histogram equalization on the illumination component. Then I converted back from HSI color space to RGB color space. Here is a new flower.ppm image after histogram equalization.

(original image: flower.ppm)

(image after histogram equalization)

Question2) Use median filter to reduce the impulse noise in this color image. Compare the results of applying the filter on the I component ONLY using the HSI model and that on RGB components using the RGB model. Provide the difference image for comparison.

Method 1) Applying the median filter on the I component of HSI model.

BASIC Procedure)

1) Given a color image with RGB color space, I converts the RGB color space to HSI color space.
2) Apply median filter only on the I component of HSI color space.
3) Converts the HSI color space back to RGB color space.

The Image mediancolorhsi(Image&, int) function in the *colorimageprocessing.cpp* implements this method. Here are the resulting images with this method.

(original image)

(median filter on I with kernelsize=3)

(median filter on I with kernelsize=5)

(median filter on I with kernelsize=7)

Like you can see above, median filter with greater kernel size works better at removing noises; however, the image gets more distortion and losses much detail.

Method 2) Applying the median filter on the RGB model.

BASIC Procedure)

1) Given a color image with RGB color space, split the image down to each RGB components.
2) Apply median filter on each RGB components.
3) Combine all RGB components to get the final image.

The Image mediancolorrgb(Image&, int) function in the *colorimageprocessing.cpp* implements this method. Here are the resulting images with this method.

(original image)

(median filter on RGB with kernelsize=3)

(median filter on RGB with kernelsize=5)

(median filter on RGB with kernelsize=7)

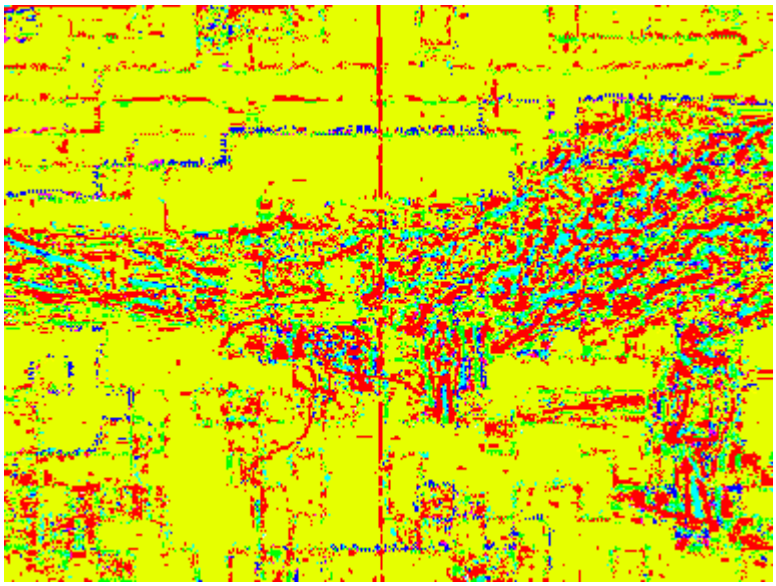
Just like the method1, the median filter removes noise better with the bigger kernel size; however, the image losses more details.

Difference Images)

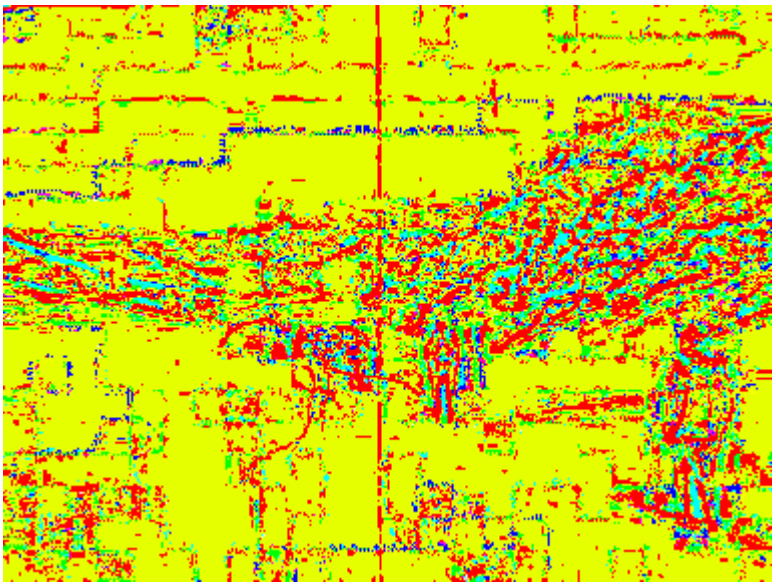
Here are the difference images between images generated by method1 and method2. The images in the first row are just difference images; whereas the images in the second row are difference images after histogram equalization. I applied histogram equalization as the difference images were very dark. Like you can see below, the difference between images generated by method1 and method2 is very small. From this observation, we can assume that it might be more computationally efficienct to work in HSI color space as we only need to modify the intensity component (leaving the hue and saturation components unmodified).

(difference image with kernelsize=3)

(difference image with kernelsize=5)



(difference image with kernelsize=3 after histogram equalization)



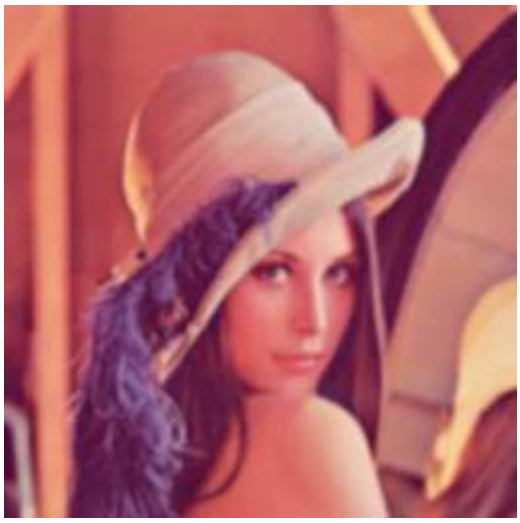
(difference image with kernelsize=5 with histogram equalization)

need more explanation of two result images. In RGB result, chromaticity changed. -1

Question3) Use Wiener filter to deblur this color image (lenablur.ppm). Therefore, if your inverse filter doesn't give you expected solution, you should know why and comment on it in your report)

Here are the original image and the deblurred images:

(original image:lena.pgm)

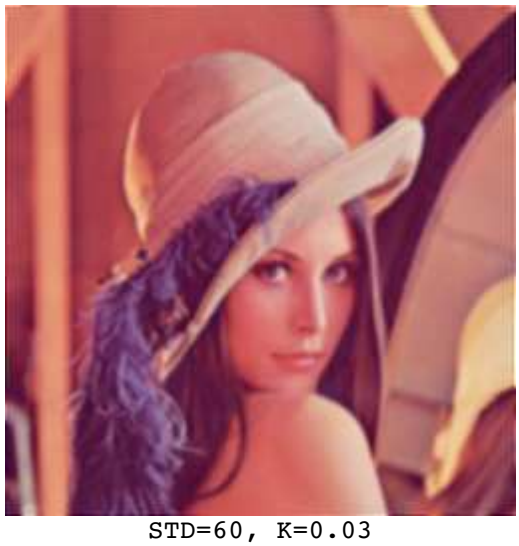


(blurred images: lenablur.ppm)

Method1) applying wiener filter on every RGB components

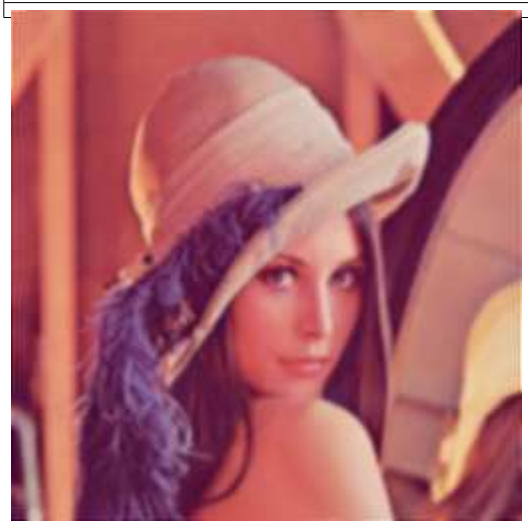
I applied wiener filter on every RGB components, and obtained the following results:

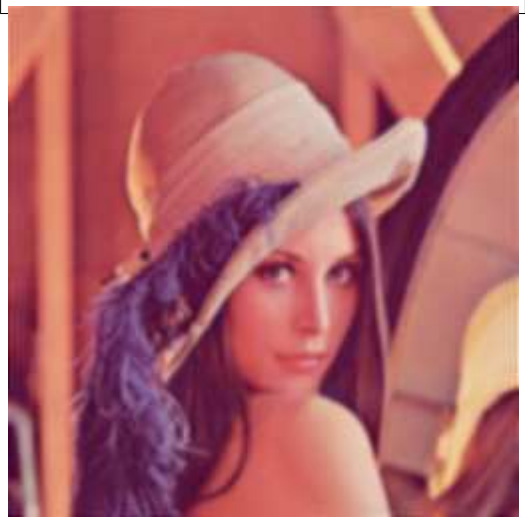
STD=10, K=0.03

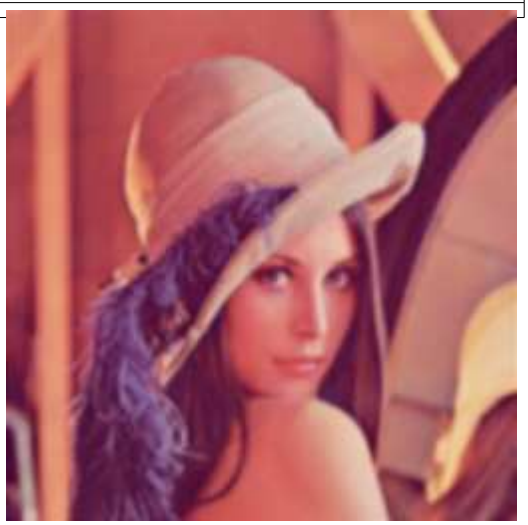


STD=60, K=0.03

STD=70, K=0.03



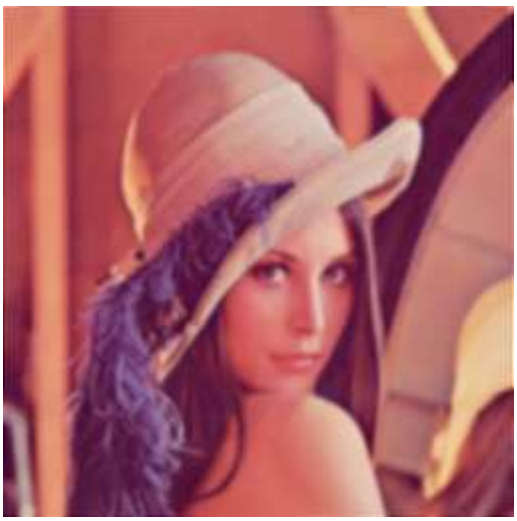




STD=80,K=0.03	STD=100,K=0.03	STD=110,K=0.03

As you can see above, I could not successfully remove the blur using wiener filter. **De-blurring in general is a difficult process since H makes this an ill-posed problem; therefore, a small error in the estimation of H will result in a dramatic error in the output image. De-blurring with color image is even harder since it is possible that different color channels are affected by different noise.**

Method2) applying wiener filter on I components in HSI

 STD=10,K=0.03	 STD=60, K=0.03	 STD=80,K=0.03

Like method2, I could not successfully remove the blur using wiener filter. I suppose that de-blurring with color image is extremely hard process since the noise might have affected each channels in many different ways.

Question4) Read Sec. 6.7.3 and derive the edge image from "peppers.ppm"

To derive the edge image from “peppers.ppm”, I used two different method. The first method is the vector method discussed in section 6.7.3. And the second one is computing the gradient of each RGB component image and formin g acomposite gradient image by adding the corresponding values of the three components at each coordinate.

Method1) Gradient computed in RGB color vector space

Sobel operation is used for working out partial derivatives. Firstly, I worked out the value of **u** and **v** by using the following formula:

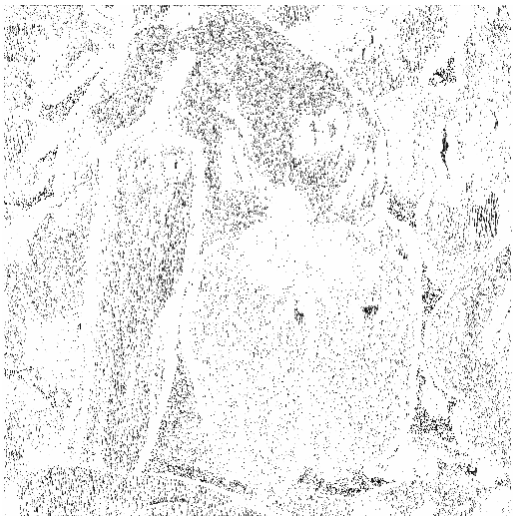
$$\boldsymbol{u}=\frac{dR}{dx}\boldsymbol{u}+\frac{dG}{dx}\boldsymbol{g}+\frac{dB}{dx}\boldsymbol{b}$$
$$\boldsymbol{v}=\frac{dR}{dy}\boldsymbol{u}+\frac{dG}{dy}\boldsymbol{g}+\frac{dB}{dy}\boldsymbol{b}$$

Here are the photos of **u** and **v** images:

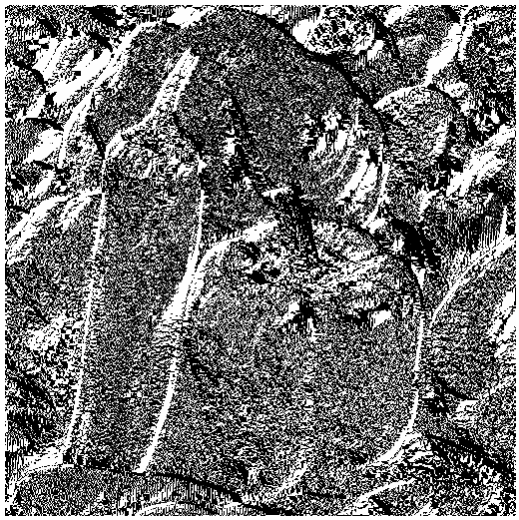
(image **u**)

(image **v**)

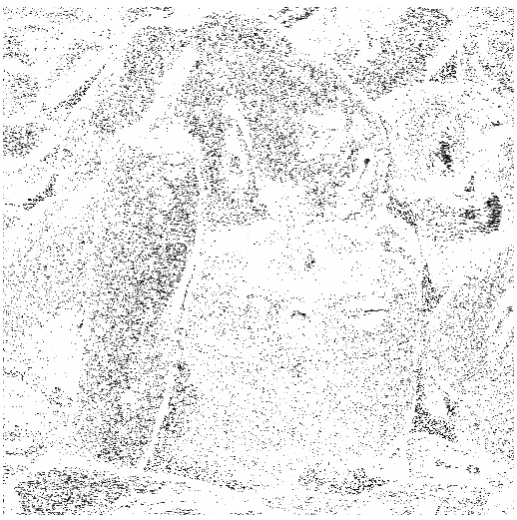
Then, I worked ou the value of g_{xx}, g_{yy}, and g_{xy}. Here are the results:



g_xx

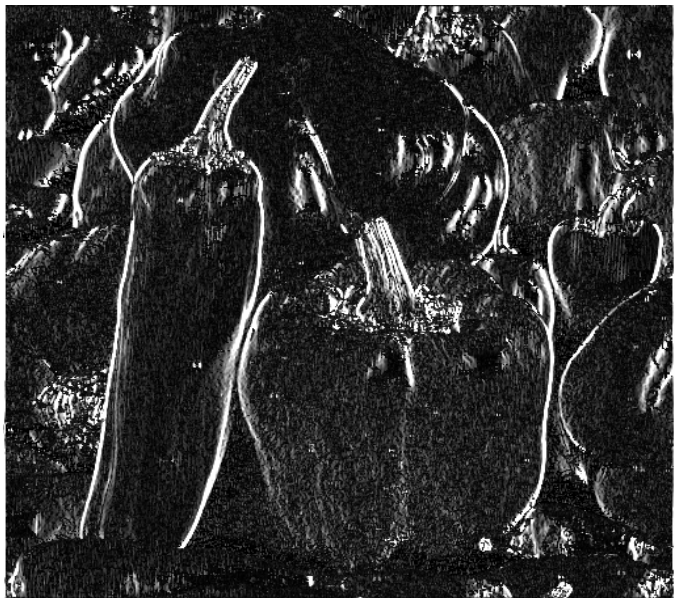


g_xy



g_yy

And finally, I worked out the color edge detection image.

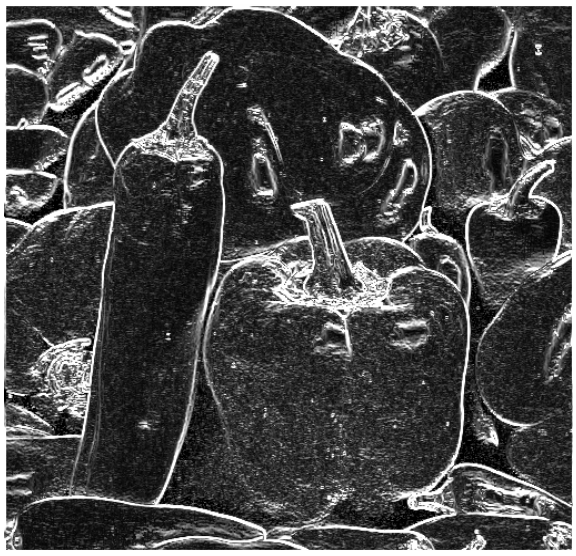


(Gradient computed in RGB color vector space)

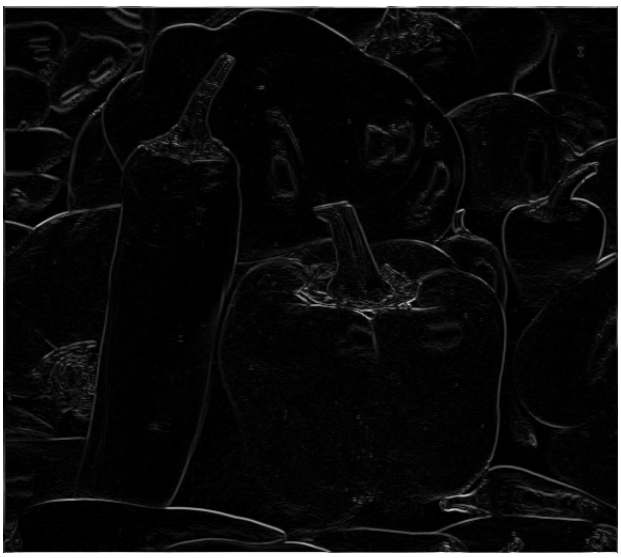
Method 2) Gradient computed on a per-image basis and then added.
In this method, I computed the gradient of each RGB component image and added all the component images. Here are the results:

(Gradients computed on a per-image basis and then added)

Difference image between method1 and method2)
Here is a difference image between method1 and method2.



(difference image)

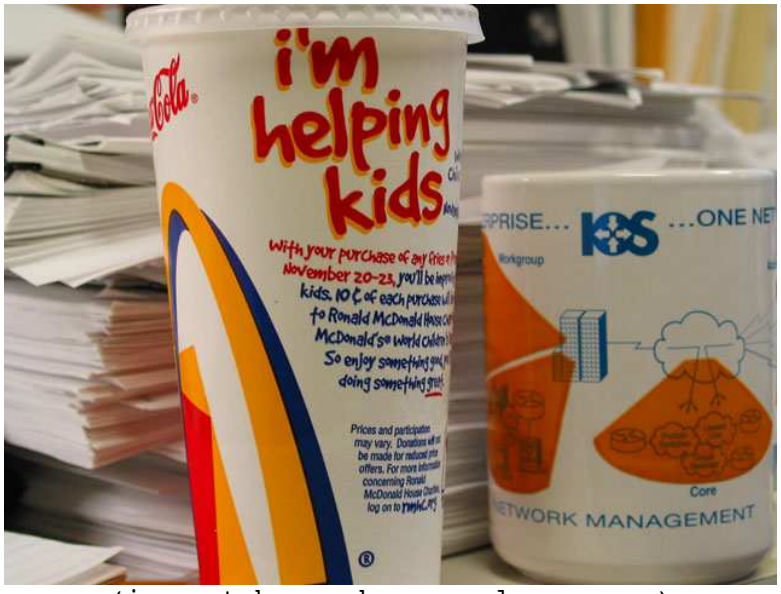


(rescaled difference image)

Task 3) Color correction

*** Correcting the image taken under under-exposure.**

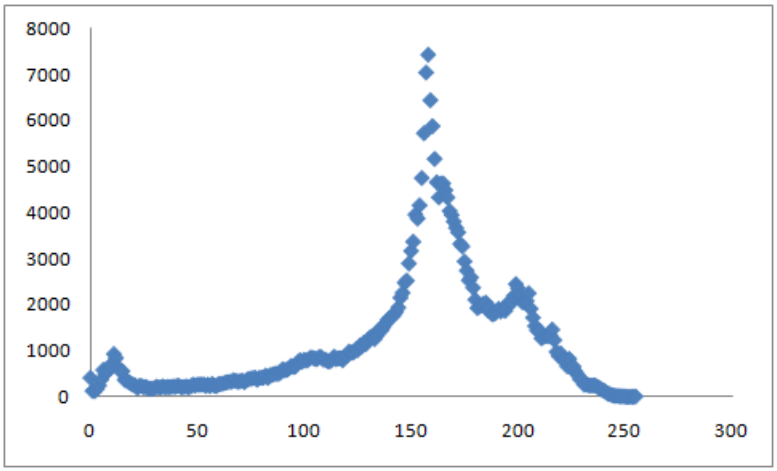
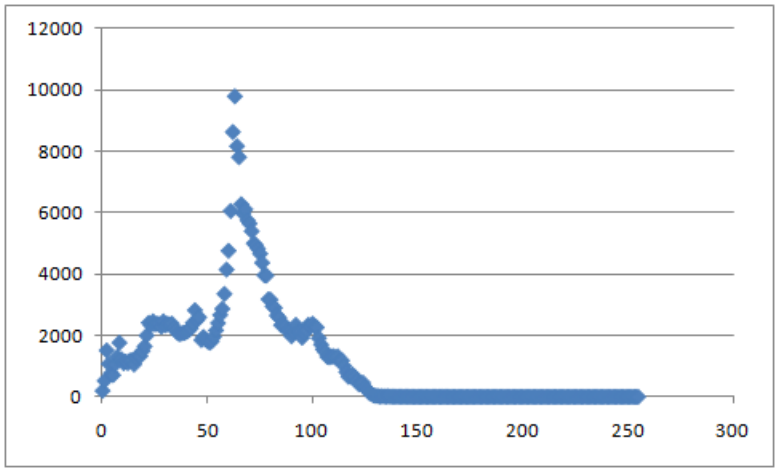
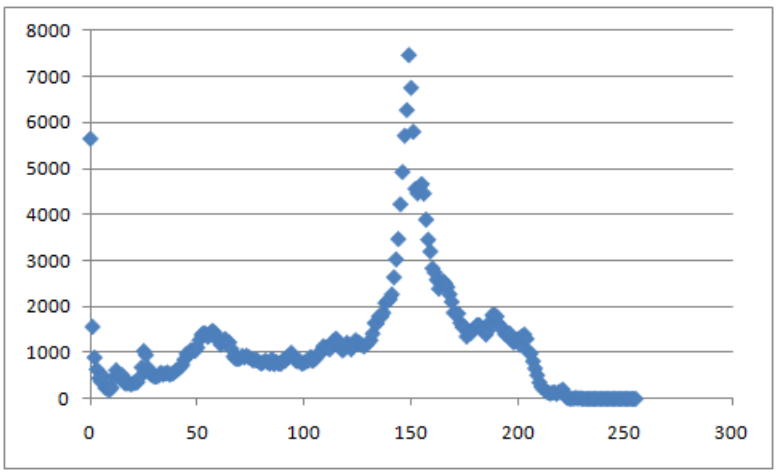
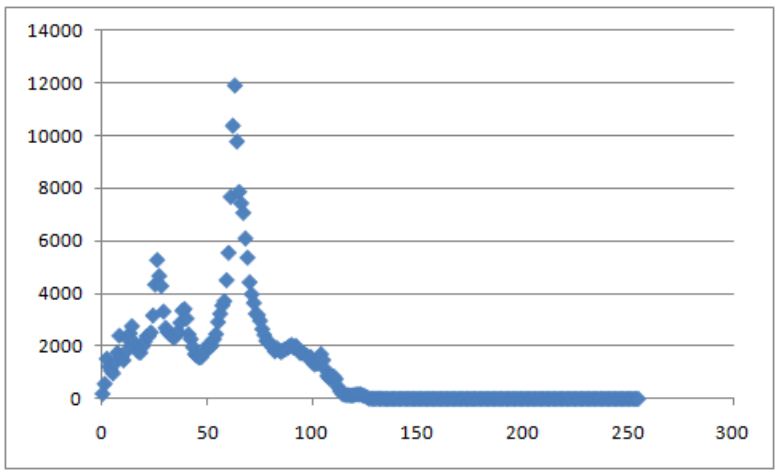
Here are the images taken under normal and under-exposure.

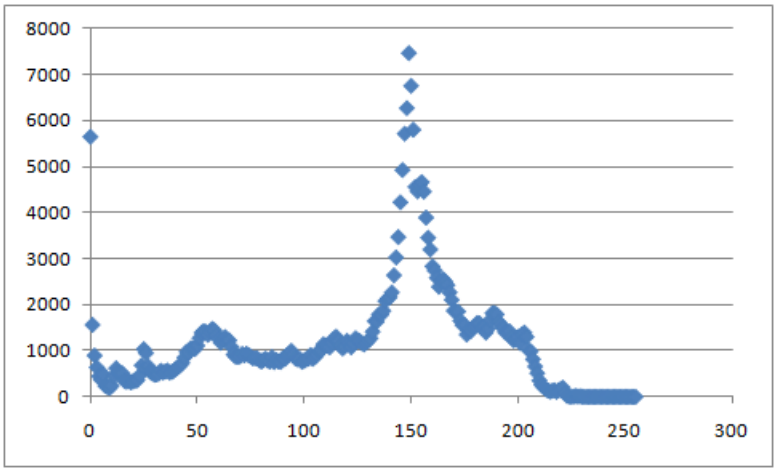


(image taken under normal exposure)

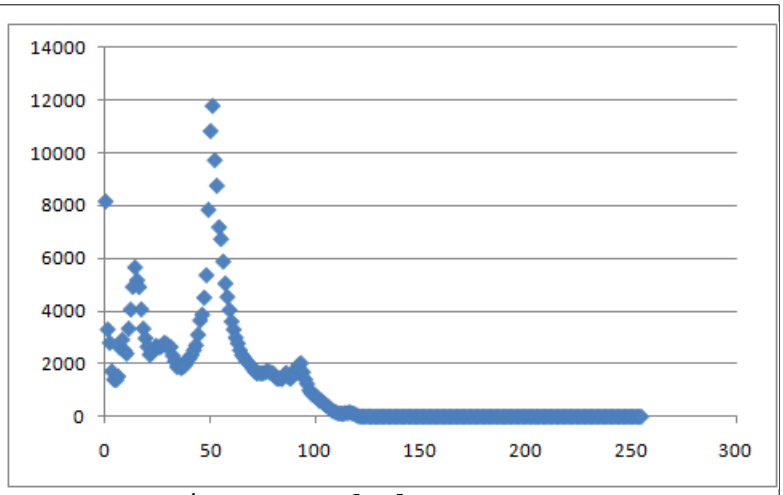
(image taken under under-exposure)

Firstly, I worked out the histogram of RGB components of the both images. Here are the results.

Image taken under normal exposure	Image taken under under-exposure
	
(Histogram of Red component)	(Histogram of Red component)
	
(histogram of green component)	(histogram of green component)



(Histogram of Blue component)



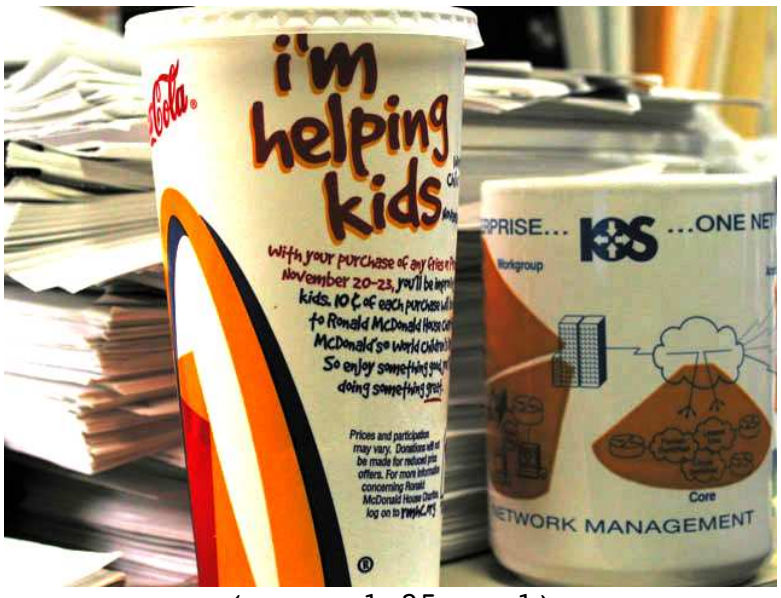
(Histogram of Blue Component)

The first column of the table shows histograms of RGB components of the image taken under normal exposure. The second column of the table shows histograms of RGB components of the image taken under under-exposure. Histograms of the image taken under under-exposure are shifted to the left compared to the histograms of the image taken under normal exposure; therefore, the image looks darker. I applied the below powerlaw function to all RGB components to make the image more brighter.

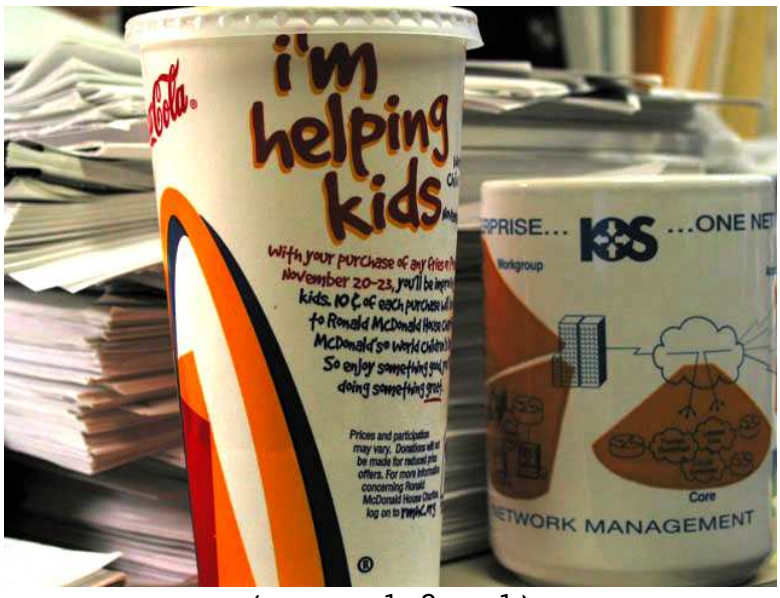
$$S = C * r^{gamma}$$

where r is the original pixel intensity, s is the enhanced pixel intensity.

Here are the results:



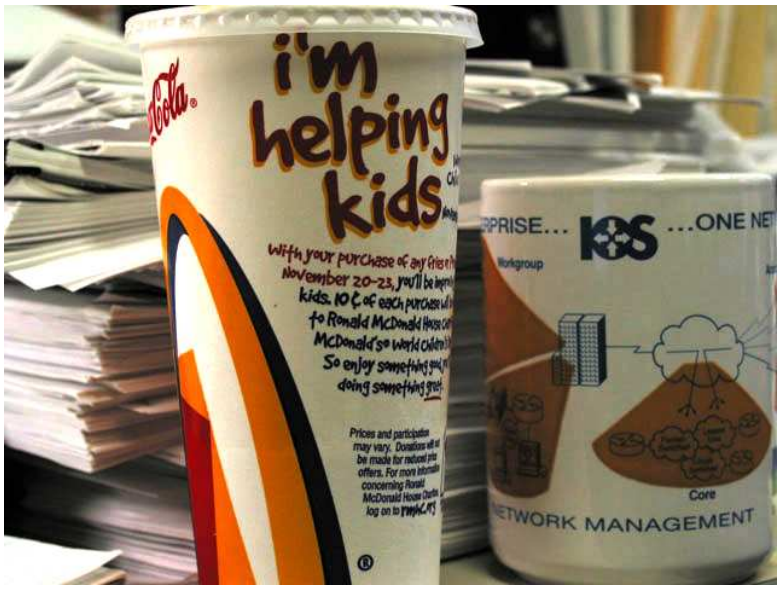
(gamma=1.25, c=1)



(gamma=1.2,c=1)

I also converted the RGB color space of the image to the HSI color space, and applied the same powerlaw function on the intensity component. Here are the results:

(c=1,gamma=1.25)

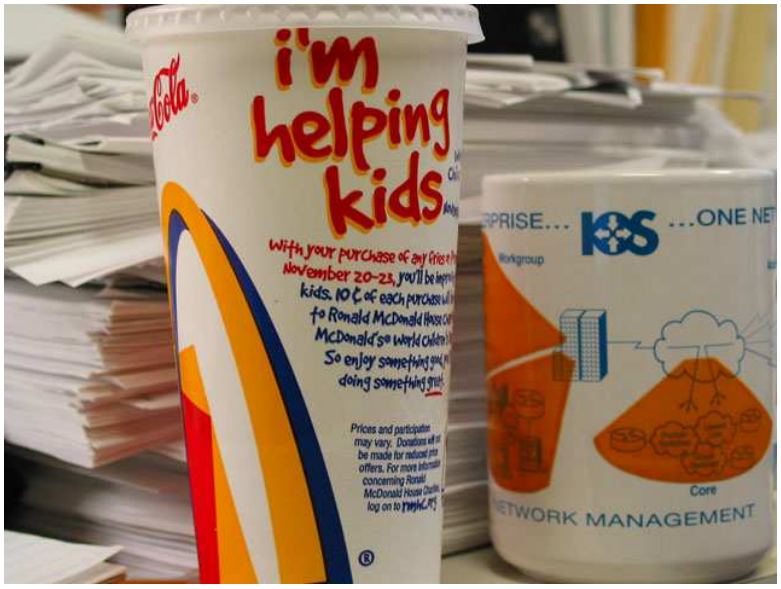


(c=1,gamma=1.2)

As you see above, we can achieve similar effects also by applying powerlaw function on the intensity component of HSI color space.

*** Correcting the image taken under over-exposure.**

Here are the images taken under normal and over-exposure.



(Image taken under normal exposure)

(Image taken under over-exposure)

Just like the image taken under under-exposure, I applied the below powerlaw transformation function on all RGB components to make the image darker:

$$s = c * r^{gamma}$$

where r is the original pixel intensity, s is the enhanced pixel intensity. Here are the results:

(gamma=0.95,c=1)

(gamma=0.95,c=0.9)

(c=0.9,gamma=0.93)

(c=1.3,gamma=0.85)

Here are the images computed by performing powerlaw transformation function on th e intensity component of HSI color space.

(c=1,gamma=0.95)

(c=0.9,gamma=0.95)

(c=0.9,gamma=0.93)

(c=1.3,gamma=0.85)

Conclusion

In this project, I examined various color image processing techniques: pseudo-color processing, color image processing, and color correction. It was very interesting to find out that we can apply the gray-scale image processing techniques such as histogram equalization and median filter to the color images to achieve similar effects.

```
/*********************************************************
 * colorImageProcessing.cpp - color image processing
 *
 *  - intensityslicing : performs intensity slicing
 *  - greytocolorsp : convert grey-level to color using spatial domain method
 *  - greytocolorfreq : conert to grey-level to color using frequency domain method

 *  - mediancolorrgb: perform median filtering on color image using rgb channels
 *  - coloredge_vector: color edge detection by using vector method
 *  - coloredge_perimage: color edge detection by computing per-image basis
 * Author: Sang-hyeb(Sam) Lee (C) slee91@utk.edu
 *
 * Created: 09/02/2011
 *
 * Modified:
 *********************************************************/

#include "Image.h"
#include "Dip.h"
#include "utility.h"
#include <iostream>
#include <cstdlib>
#include <cstring>
#include <cmath>

using namespace std;
/**
 * Square root of an image.
 * @param inimg The input image.
 * @return The square root of the image.
 */
Image sqrt(Image &inimg) {
  int i, j, k;
  Image outimg;

  outimg.createImage(inimg.getRow(), inimg.getCol(), inimg.getType());

  for (i=0; i<inimg.getRow(); i++)
    for (j=0; j<inimg.getCol(); j++)
      for (k=0; k<inimg.getChannel(); k++)
            outimg(i,j,k) = sqrt(inimg(i,j,k));

  return outimg;
}

/**
 * Perform convlution
 * @param inimg input image
 * @param mask mask
 * @return image after convlution with given mask
 */
Image conv(Image &inimg, Image &mask)
{
  Image outimg;
  int i, j, k, m, n;
  int nc1, nr1, nc2, nr2, ntype1, nchan1, nchan2;
  float sum, radiusC, radiusR;

  // get the dimension of the image
  nc1 = inimg.getCol();
  nr1 = inimg.getRow();
  ntype1 = inimg.getType();
  nchan1 = inimg.getChannel();

  // get the dimension of the kernel
  nc2 = mask.getCol();
  nr2 = mask.getRow();
  nchan2 = mask.getChannel();
  if (nchan2 > 1) {
    cout << "convolution: The mask cannot have more than 1 channel.\n";
    exit(3);
  }

  // handle odd and even mask size
  radiusC = (float)nc2/2;
  radiusR = (float)nr2/2;

  // allocate memory for the output image
  outimg.createImage(nr1, nc1, ntype1);

  // perform the convolution (or kernel operation)
  for (k=0; k<nchan1; k++) {
    for (i=0; i<nr1; i++)
      for (j=0; j<nc1; j++) {
        sum = 0;
        for (m=(int)-radiusR; m<=(int)radiusR; m++)
          for (n=(int)-radiusC; n<=(int)radiusC; n++)
            if (i+m>=0 && i+m<nr1 && j+n>=0 && j+n<nc1 &&
                radiusR+m>=0 && radiusR+m<nr2 && radiusC+n>=0 && radiusC+n<nc2)
              sum += inimg(i+m,j+n,k) * mask((int)radiusR+m,(int)radiusC+n);
        outimg(i,j,k) = sum;
      }
  }

  return outimg;
}

/**
 * Sobel edge detector
 *        mask 1                mask 2
 *        -1  0  1             -1  -2  -1
 *        -2  0  2              0   0   0
 *        -1  0  1              1   2   1
 * @param inimg The input image
 * @return The edge image using the Sobel kernels
 */
Image sobel(Image &inimg) {
  Image outimg, outimg1, outimg2, mask1, mask2;

  // create the masks
  mask1.createImage(3,3);
  mask2.createImage(3,3);

  mask1(0,0) = mask1(2,0) = -1;
  mask1(0,2) = mask1(2,2) = 1;
  mask1(1,0) = -2;
  mask1(1,2) = 2;

  mask2 = transpose(mask1);

  // kernel operation
  outimg1 = conv(inimg, mask1);  // vertical edge
  outimg2 = conv(inimg, mask2);  // horizontal edge

  // combine edges in both directions
  Image temp = outimg1 * outimg1 + outimg2 * outimg2;
  outimg = sqrt(temp);

  return outimg;
}

/**
 * Work out gradient image by computing the gradient of RGB component image
 * and then adding them all
```

APPROVED

```
 * @param inimg input image
 * @return gradient image
 */
Image coloredge_perimage(Image &inimg) {
  Image inimg_red,inimg_blue,inimg_green;
  int nr,nc,ntype;
  Image gradient_red,gradient_blue,gradient_green;
  Image outimg;

  nr = inimg.getRow();
  nc = inimg.getCol();
  ntype = inimg.getType();

  inimg_red = inimg.getImage(0); //red
  inimg_green = inimg.getImage(1); //green
  inimg_blue = inimg.getImage(2); //blue

  gradient_red = sobel(inimg_red);
  gradient_blue = sobel(inimg_blue);
  gradient_green = sobel(inimg_green);

  outimg.createImage(nr,nc);
  outimg = gradient_red + gradient_blue + gradient_green;

  writeImage(gradient_red,"gradient_red.pgm");
  writeImage(gradient_green,"gradient_green.pgm");
  writeImage(gradient_blue, "gradient_blue.pgm");

  return outimg;




}

/**
 * Perform color edge detection using vector method
 * @param inimg input image
 * @return edge image
 */
Image coloredge_vector(Image &inimg) {
  Image outimg, outimg1, outimg2, mask1, mask2;
  Image inimg_red,inimg_green,inimg_blue;
  int nr,nc;

  nr = inimg.getRow();
  nc = inimg.getCol();

  // create the masks
  mask1.createImage(3,3);
  mask2.createImage(3,3);

  mask1(0,0) = mask1(2,0) = -1;
  mask1(0,2) = mask1(2,2) = 1;
  mask1(1,0) = -2;
  mask1(1,2) = 2;

  mask2 = transpose(mask1);

  // kernel operation
  outimg1 = conv(inimg, mask1);  // vertical edge
  outimg2 = conv(inimg, mask2);  // horizontal edge

  //break down to rgb components
  inimg_red = inimg.getImage(0);
  inimg_green = inimg.getImage(1);
  inimg_blue = inimg.getImage(2);

  Image gradient_red_x = conv(inimg_red,mask1); //vertical edge
  Image gradient_blue_x = conv(inimg_blue,mask1);
  Image gradient_green_x = conv(inimg_green,mask1);

  Image v = gradient_red_x + gradient_blue_x + gradient_green_x;



  Image gradient_red_y = conv(inimg_red,mask2); //horizontal edge
  Image gradient_blue_y = conv(inimg_blue,mask2);
  Image gradient_green_y = conv(inimg_green,mask2);

  Image u = gradient_red_y + gradient_blue_y + gradient_green_y;
  writeImage(v,"v.pgm");
  writeImage(u,"u.pgm");


   Image g_xx = gradient_red_x * gradient_red_x + gradient_green_x * gradient_green
_x + gradient_blue_x * gradient_blue_x;
  //Image g_xx = u_t ->* u;
  Image g_yy = gradient_red_y * gradient_red_y + gradient_green_y * gradient_green_
y + gradient_blue_y * gradient_blue_y;
  Image g_xy =  gradient_red_y * gradient_red_x + gradient_green_y * gradient_green
_x + gradient_blue_y * gradient_blue_x;

  writeImage(g_xx,"g_xx.pgm");
  writeImage(g_xy,"g_xy.pgm");
  writeImage(g_yy,"g_yy.pgm");

  //work out theta
  Image theta;
  theta.createImage(nr,nc);
  double data;
  double inside;
  double inside_2;

  for(int i=0;i<nr;i++) {
    for(int j=0;j<nc;j++) {
      inside = 2 * g_xy(i,j) / (g_xx(i,j) - g_yy(i,j));
      inside_2 = atan(inside);
      data = (1/2.0) * inside_2;;

      //cout<<i<<","<<j<<"="<<data<<", "<<inside<<","<<inside_2<<endl;;
      theta(i,j)=data;
    }
  }
  writeImage(theta,"theta.pgm");

  Image f_xy;
  f_xy.createImage(nr,nc);
  for(int i=0;i<nr;i++) {
    for(int j=0;j<nc;j++) {
      f_xy(i,j) = sqrt((1/2.0) * ((g_xx(i,j) + g_yy(i,j))
                                + (g_xx(i,j) - g_yy(i,j))*cos(2*theta(i,j)) +
                                (2*g_xy(i,j) * sin(2*theta(i,j)))));
    }
  }


  // combine edges in both directions
  //Image temp = outimg1 * outimg1 + outimg2 * outimg2;
  //outimg = sqrt(temp);

  return f_xy;
}
/**
```

```
 * Perform median filtering color image by converting to HSI color space
 * and performing median filtering on I.
 *
 * @param inimg input image
 * @param kernelsize kernel size
 * @return color image after median filtering
 */
Image mediancolorhsi(Image &inimg, int kernelSize) {
  Image outimg;
  int i, j, k;
  int nr, nc, ntype, nchan;
  Image inimg_hsi;
  Image inimg_intensity;
  Image inimg_intensity_mf;
   // allocate memory
  nr = inimg.getRow();
  nc = inimg.getCol();
  ntype = inimg.getType();
  nchan = inimg.getChannel();
  outimg.createImage(nr, nc, ntype);

  //convert RGB to HSI
  inimg_hsi = RGB2HSI(inimg);

  //perform median filtering on intensity
  inimg_intensity = inimg_hsi.getImage(2);
  inimg_intensity_mf = median(inimg_intensity,kernelSize);

  //put this back
  inimg_hsi.setImage(inimg_intensity_mf,2);

  //convert HSI to RGB
  outimg = HSI2RGB(inimg_hsi);

  return outimg;

}
/**
 * Perform median filtering on color image by performing median filtering
 * on all RGB channels
 * @return color image after median filtering
 */
Image mediancolorrgb(Image &inimg,int kernelSize) {
  Image outimg;
  int i, j, k;
  int nr, nc, ntype, nchan;
  Image inimg_red,inimg_green,inimg_blue;
  Image out_red,out_green,out_blue;

   // allocate memory
  nr = inimg.getRow();
  nc = inimg.getCol();
  ntype = inimg.getType();
  nchan = inimg.getChannel();
  outimg.createImage(nr, nc, ntype);

  //extract each RGB components
  inimg_red = inimg.getRed();
  inimg_green = inimg.getGreen();
  inimg_blue = inimg.getBlue();

  //perform median filtering filter
  out_red = median(inimg_red,kernelSize);
  out_green = median(inimg_green,kernelSize);
  out_blue = median(inimg_blue, kernelSize);

  //put it back
  outimg.setRed(out_red);
  outimg.setGreen(out_green);
  outimg.setBlue(out_blue);

  return outimg;

}

int sort(const void *x, const void *y) {
  return (*(float*)x - *(float*)y);
}

/**
 * Median filter
 * @param inimg Input image
 * @param kernelSize size
 * @return image with median filter effect
 */
Image median(Image& inimg, int kernelSize) {

  int nr, nc, ntype, nchan;
  Image outimg;
  int i, j, k;
  int arraySize = kernelSize * kernelSize;
  float array[arraySize];
  int arrayCount=0;

  int radius = kernelSize /2;

  // allocate memory
  nr = inimg.getRow();
  nc = inimg.getCol();
  ntype = inimg.getType();
  nchan = inimg.getChannel();

  outimg.createImage(nr, nc, ntype);

  //for each position (i,j,k) in the image
  for (i=0; i<nr; i++) {
    for (j=0; j<nc; j++) {
      for (k=0; k<nchan; k++) {

        //apply median filter
        arrayCount=0; //set the number of element array to 0
        for(int xi=-radius;xi<=radius;xi++) {
          for(int xj=-radius;xj<=radius;xj++) {
            if( (xi+i)>=0 && (xi+i)<nr && (xj+j)>=0 && (xj+j)<nc) {
              //put the pixel intensity into the array
              array[arrayCount]=inimg(xi+i,xj+j,k);
              arrayCount++;
            }
          }
        }

        //sort the array of pixel values
        qsort(array,arrayCount,sizeof(float),sort);

        //pick the one in the middle
        outimg(i,j,k) = array[arrayCount/2];
      }
    }
  }

  return outimg;


}
```

```
/**
 * Grey-level to color transformation in frequency domain
 * @param inimg input img
 * @param p1 -- D_0 for red filter
 * @param p2 -- D_0 for blue filter
 * @param p3 -- D_0 for green filter
 * @return grey to color transformation frequency
 */
Image greytocolorfreq(Image& inimg,float p1,float p2,float p3) {
  Image outimg;
  int i, j, k;
  int nr, nc, ntype, nchan;
  int P,Q;
  Image img2; //padded image
  Image mag,phase;
  Image H_red,H_green,H_blue;
  Image out_red,out_green,out_blue;
  Image cro_red,cro_green,cro_blue;
  Image hs_red, hs_green, hs_blue;

  // allocate memory
  nr = inimg.getRow();
  nc = inimg.getCol();
  ntype = inimg.getType();
  nchan = inimg.getChannel();

  //creates a color image
  outimg.createImage(nr,nc,PPMRAW);

  //find the right padding size
  P=2;
  while(true) {
    if(P>=(nr*2)&&P>=(nc*2)) break; //check if condition is met
    P = P *2;
  }
  Q=P;

  //creates a padded image
  img2.createImage(P,Q);
  for(i=0;i<nr;i++)
    for(j=0;j<nc;j++)
      img2(i,j) = inimg(i,j);

  //Fourier transform
  mag.createImage(P,Q);
  phase.createImage(P,Q);
  out_red.createImage(P,Q);
  out_blue.createImage(P,Q);
  out_green.createImage(P,Q);
  fft(img2,mag,phase);

  //create filter for red -- high butterworth pass
  H_red = hpButterworth(p1,1,P,Q);
  //create filter for blue -- high gaussian pass
  H_blue = hpGaussian(p2,P,Q);
  //create filter for green -- low gaussian pass
  H_green = lpGaussian(p3,P,Q);

  //apply filter
  H_red = H_red * mag;
  H_blue = H_blue * mag;
  H_green = H_green * mag;

  //inverse fourier transform
  ifft(out_red, H_red, phase);
  ifft(out_blue, H_blue, phase);
  ifft(out_green, H_green,phase);

  //crop the image
  cro_red = subImage(out_red,0,0,nr-1,nc-1);
  cro_blue = subImage(out_blue,0,0,nr-1,nc-1);
  cro_green = subImage(out_green,0,0,nr-1,nc-1);

  //histogram equalization
  out_red = histeq(cro_red);
  out_blue = histeq(cro_blue);
  out_green = histeq(cro_green);

  //comebine the image
  for(i=0;i<nr;i++) {
    for(j=0;j<nc;j++) {
      outimg(i,j,0) = out_red(i,j,0);   //red
      outimg(i,j,1) = out_green(i,j,0); //green
      outimg(i,j,2) = out_blue(i,j,0);  //blue
    }
  }

  writeImage(out_red,"red.pgm");
  writeImage(out_green,"green.pgm");
  writeImage(out_blue,"blue.pgm");

  return outimg;



}

/**
 * Grey-level to color transformation in spatial domain
 * @param inimg input img
 * @param p1 changes frequency
 * *param p2 shifts the wave
 */
Image greytocolorsp(Image& inimg,float p1,float p2) {
  Image outimg;
  int i, j, k;
  int nr, nc, ntype, nchan;

  // allocate memory
  nr = inimg.getRow();
  nc = inimg.getCol();
  ntype = inimg.getType();
  nchan = inimg.getChannel();

  //creates a color image
  outimg.createImage(nr,nc,PPMRAW);

  //for each pixel in the image
  float px;
  for (i=0; i<nr; i++)
    for (j=0; j<nc; j++) {
      px = inimg(i,j,0);

      //assign each component value.
      //red
      outimg(i,j,0) = L * abs(sin((px*p1/L)));
      //blue
      outimg(i,j,1) = L * abs(sin((px*p1/L)+p2));
      //green
      outimg(i,j,2) = L * abs(sin((px*p1/L)+p2+p2));

    }
```

```
  return outimg;
}



/**
 * Intensity slicing
 * convert grey-scale image using intensity slicing
 * @param inimg input image
 */
Image intensityslicing(Image &inimg) {
  Image outimg;
  int i, j, k;
  int nr, nc, ntype, nchan;

  // allocate memory
  nr = inimg.getRow();
  nc = inimg.getCol();
  ntype = inimg.getType();
  nchan = inimg.getChannel();

  //creates a color image
  outimg.createImage(nr,nc,PPMRAW);

  //for each pixel determine the color
  float px;
  for (i=0; i<nr; i++)
   for (j=0; j<nc; j++) {
     px = inimg(i,j,0);
     if(px<12) {
       //black
       outimg(i,j,0) = 0x00;
       outimg(i,j,1) = 0x00;
       outimg(i,j,2) = 0x00;
     }else if(px<41) {
       outimg(i,j,0) = 0x33;
       outimg(i,j,1) = 0x00;
       outimg(i,j,2) = 0x99;
       //color 2
     }else if(px<93) {
       //color 3
       outimg(i,j,0) = 0x33;
       outimg(i,j,1) = 0x33;
       outimg(i,j,2) = 0x99;
     }else if(px<98) {
       //color 4
       outimg(i,j,0) = 0x33;
       outimg(i,j,1) = 0x66;
       outimg(i,j,2) = 0x99;
     }else if(px<101) {
       //color 5
       outimg(i,j,0) = 0x33;
       outimg(i,j,1) = 0xCC;
       outimg(i,j,2) = 0x99;
     }else if(px<106) {
       //color 6
       outimg(i,j,0) = 0xCC;
       outimg(i,j,1) = 0x00;
       outimg(i,j,2) = 0x99;
     }else if(px<112) {
       //color 7
       outimg(i,j,0) = 0xCC;
       outimg(i,j,1) = 0x66;
       outimg(i,j,2) = 0xFF;
     }else if(px<112) {
       //color 8
       outimg(i,j,0) = 0xFF;
       outimg(i,j,1) = 0x99;
       outimg(i,j,2) = 0x00;
     }else if(px<130) {
       outimg(i,j,0) = 0xCC;
       outimg(i,j,1) = 0xCC;
       outimg(i,j,2) = 0xCC;
     }else {
       outimg(i,j,0) = 0xCC;
       outimg(i,j,1) = 0xFF;
       outimg(i,j,2) = 0xFF;
     }
   }



  return outimg;
}
```

```
/*************************************************************
 * tesths.cpp: test code for histogram equalization
 *
 * Author: Sang-hyeb(Sam) Lee (C) slee91@utk.edu
 *
 * Created: 08/31/11
 *
 *************************************************************/

#include "Image.h"
#include "Dip.h"
#include "utility.h"
#include <iostream>
#include <cstdlib>
#include <cstring>
#include <cstdio>

using namespace std;

#define Usage "testchs inimg outimg \n"

int main(int argc, char **argv)
{
  Image inimg, outimg;    // the original image
  Image hsi;
  Image hsi_intensity;
  Image hsi_intensity_he;

  // check if the number of arguments on the command line is correct
  if (argc < 2) {
    cout << Usage;
    exit(3);
  }

  // read in image
  inimg = readImage(argv[1]);
  // print the histogram of input image
  //printHistogram(inimg,"input.dat");

  // convert to hsi
  hsi = RGB2HSI(inimg);
  hsi_intensity = hsi.getImage(2);
  hsi_intensity_he = histeq(hsi_intensity);

  //put this back
  hsi.setImage(hsi_intensity_he,2);

  outimg = HSI2RGB(hsi);


  //outimg = histeq(inimg);

  // print out function
  writeImage(outimg, argv[2]);
  // print the histogram of output image
  //printHistogram(outimg,"output.dat");
  return 0;
}
```

```
/*************************************************************
 * tesths.cpp: test code for histogram equalization
 *
 * Author: Sang-hyeb(Sam) Lee (C) slee91@utk.edu
 *
 * Created: 08/31/11
 *
 *************************************************************/

#include "Image.h"
#include "Dip.h"
#include "utility.h"
#include <iostream>
#include <cstdlib>
#include <cstring>
#include <cstdio>

using namespace std;

#define Usage "testclr inimg outimg c gamma\n"


int main(int argc, char **argv)
{
  Image inimg, outimg;    // the original image
  Image inimg_red, inimg_green, inimg_blue; //each RGB components
  Image out_red, out_green, out_blue;
  int nr,nc,ntype,nchan;
  float c,gamma;
  Image HSI;

  // check if the number of arguments on the command line is correct
  if (argc < 4) {
    cout << Usage;
    exit(3);
  }

  // read in image
  inimg = readImage(argv[1]);

  c = atof(argv[3]);
  gamma = atof(argv[4]);

  // break it down to each RGB components
  inimg_red = inimg.getImage(0);
  inimg_green = inimg.getImage(1);
  inimg_blue = inimg.getImage(2);

  // print the histogram of input image
  printHistogram(inimg_red,"red.dat");
  printHistogram(inimg_green,"green.dat");
  printHistogram(inimg_blue,"blue.dat");

  out_red = powerlaw(inimg_red,c,gamma);
  out_green = powerlaw(inimg_green,c,gamma);
  out_blue = powerlaw(inimg_blue,c,gamma);

  // allocate memory
  nr = inimg.getRow();
  nc = inimg.getCol();
  ntype = inimg.getType();
  nchan = inimg.getChannel();

  outimg.createImage(nr, nc, ntype);

  //put everything back together
  outimg.setImage(out_red,0);
  outimg.setImage(out_green,1);
  outimg.setImage(out_blue,2);


  //HSI component!!
  //convert from RGB to HSI
  HSI = RGB2HSI(inimg);

  Image HSI_intensity = HSI.getImage(2); //get intensity component
  Image HSI_intensity_out = powerlaw(HSI_intensity,c,gamma); //perform power-law tr
ansformation

  //set the image
  HSI.setImage(HSI_intensity_out,2);

  //write the image
  Image HSI_outimg = HSI2RGB(HSI);

  writeImage(HSI_outimg,"clr_hsi.ppm");
  writeImage(outimg,argv[2]);

  return 0;
}
```

```
/*************************************************************
 * testwiener.cpp: test code for wiener color
 *
 * Author: Sang-hyeb(Sam) Lee (C) slee91@utk.edu
 *
 * Created: 08/31/11
 *
 *************************************************************/

#include "Image.h"
#include "Dip.h"
#include "utility.h"
#include <iostream>
#include <cstdlib>
#include <cstring>
#include <cstdio>

using namespace std;

#define Usage "testcoloredge inimg outimg\n"

int main(int argc, char **argv)
{
  Image inimg, outimg1,outimg2;    // the original image
  Image hsi;
  Image hsi_intensity;
  Image hsi_intensity_he;
  float std;
  float k;
  Image inimg_red,inimg_blue,inimg_green;
  Image outimg_red,outimg_blue,outimg_green;

  // check if the number of arguments on the command line is correct
  if (argc < 3) {
    cout << Usage;
    exit(3);
  }

  // read in image
  inimg = readImage(argv[1]);
  outimg1 = coloredge_vector(inimg);
  // print out function
  writeImage(outimg1, argv[2]);
  outimg2 = coloredge_perimage(inimg);
  writeImage(outimg2, "perimage.pgm");

  Image diff = outimg2-outimg1;
  Image res = rescale(diff);
  writeImage(res,"res.pgm");
  writeImage(diff,"diff.pgm");


  // print the histogram of output image
  //printHistogram(outimg,"output.dat");
  return 0;
}
```

```
// test code for contrast stretching

#include "Image.h"
#include "Dip.h"
#include <iostream>
#include <cstdlib>
#include "utility.h"

using namespace std;

#define Usage "testcs inimg outimg1 outimg2 p1 p2 p3\n"

int main(int argc, char **argv)
{
  Image inimg, outimg1, outimg2;    // the original image
  float p1,p2,p3;

  // check if the number of arguments on the command line is correct
  if (argc < 5) {
    cout << Usage;
    exit(3);
  }

  // read in image
  inimg = readImage(argv[1]);

  string filename(argv[1]);
  string ext=".dat";

  p1 = atof(argv[4]);
  p2 = atof(argv[5]);
  p3 = atof(argv[6]);


  outimg1 = greytocolorsp(inimg,p1,p2);
  outimg2 = greytocolorfreq(inimg,p1,p2,p3);

  //print out the histogram of output image
  //filename = string(argv[2]);
  //printHistogram(outimg,(filename+ext).c_str());
  // output the image
  writeImage(outimg1, argv[2]);
  writeImage(outimg2, argv[3]);

  return 0;
}
```

```
// test code for median color

#include "Image.h"
#include "Dip.h"
#include <iostream>
#include <cstdlib>
#include "utility.h"

using namespace std;

#define Usage "testmediancolor inimg kernelsize\n"

int main(int argc, char **argv)
{
  Image inimg;    // the original image
  int kernelSize;
  Image outimg1,outimg2; //out image

  // check if the number of arguments on the command line is correct
  if (argc < 3) {
    cout << Usage;
    exit(3);
  }

  // read in image
  inimg = readImage(argv[1]);

  kernelSize = atoi(argv[2]);

  //perform median filter color
  outimg1 = mediancolorhsi(inimg, kernelSize);
  outimg2 = mediancolorrgb(inimg, kernelSize);

  writeImage(outimg1,"mediancolorhsi.ppm");
  writeImage(outimg2,"mediancolorrgb.ppm");

  Image dif = outimg1 - outimg2;

  cout<<dif<<endl;

  writeImage(dif,"median_diff.ppm");

  return 0;
}
```

```
/************************************************************
 * testwiener.cpp: test code for wiener color
 *
 * Author: Sang-hyeb(Sam) Lee (C) slee91@utk.edu
 *
 * Created: 08/31/11
 *
 ************************************************************/

#include "Image.h"
#include "Dip.h"
#include "utility.h"
#include <iostream>
#include <cstdlib>
#include <cstring>
#include <cstdio>

using namespace std;

#define Usage "testchs inimg outimg std K \n"

int main(int argc, char **argv)
{
  Image inimg, outimg;    // the original image
  Image hsi;
  Image hsi_intensity;
  Image hsi_intensity_he;
  float std;
  float k;
  Image inimg_red,inimg_blue,inimg_green;
  Image outimg_red,outimg_blue,outimg_green;

  // check if the number of arguments on the command line is correct
  if (argc < 5) {
    cout << Usage;
    exit(3);
  }

  // read in image
  inimg = readImage(argv[1]);
  std = atof(argv[3]); //std
  k = atof(argv[4]); //K

  goto HSI;
  //break down the image
  inimg_red = inimg.getImage(0);
  inimg_green = inimg.getImage(1);
  inimg_blue =  inimg.getImage(2);

  //perform wiener filter
  outimg_red = wiener(inimg_red,k,std);
  outimg_green = wiener(inimg_green,k,std);
  outimg_blue = wiener(inimg_blue,k,std);

  //put this back
  outimg.createImage(inimg.getRow(),inimg.getCol(), inimg.getType());
  outimg.setImage(outimg_red,0);
  outimg.setImage(outimg_green,1);
  outimg.setImage(outimg_blue,2);


  // print the histogram of input image
  //printHistogram(inimg,"input.dat");
 HSI:
  // convert to hsi
  hsi = RGB2HSI(inimg);
  hsi_intensity = hsi.getImage(2);
  hsi_intensity_he = wiener(hsi_intensity,k,std);

  //put this back
  hsi.setImage(hsi_intensity_he,2);

  outimg = HSI2RGB(hsi);


  //outimg = histeq(inimg);

  // print out function
  writeImage(outimg, argv[2]);
  // print the histogram of output image
  //printHistogram(outimg,"output.dat");
  return 0;
}
```