ECE637 Lab report 1
Image Filter

Name:Chengzhang Zhong

Section 3. FIR Low Pass Filter

Part 1. Analytical expression for $H(e^{j\mu}, e^{jv})$

Since we know $|m| \leq 4$ and $|n| \leq 4$ in this question, the formula should looks like $H(e^{ju}, e^{jv}) = \sum_{k=-4}^{4} \sum_{l=-4}^{4} h(k,l) e^{-j(ku+lv)}$,

Thus, we have

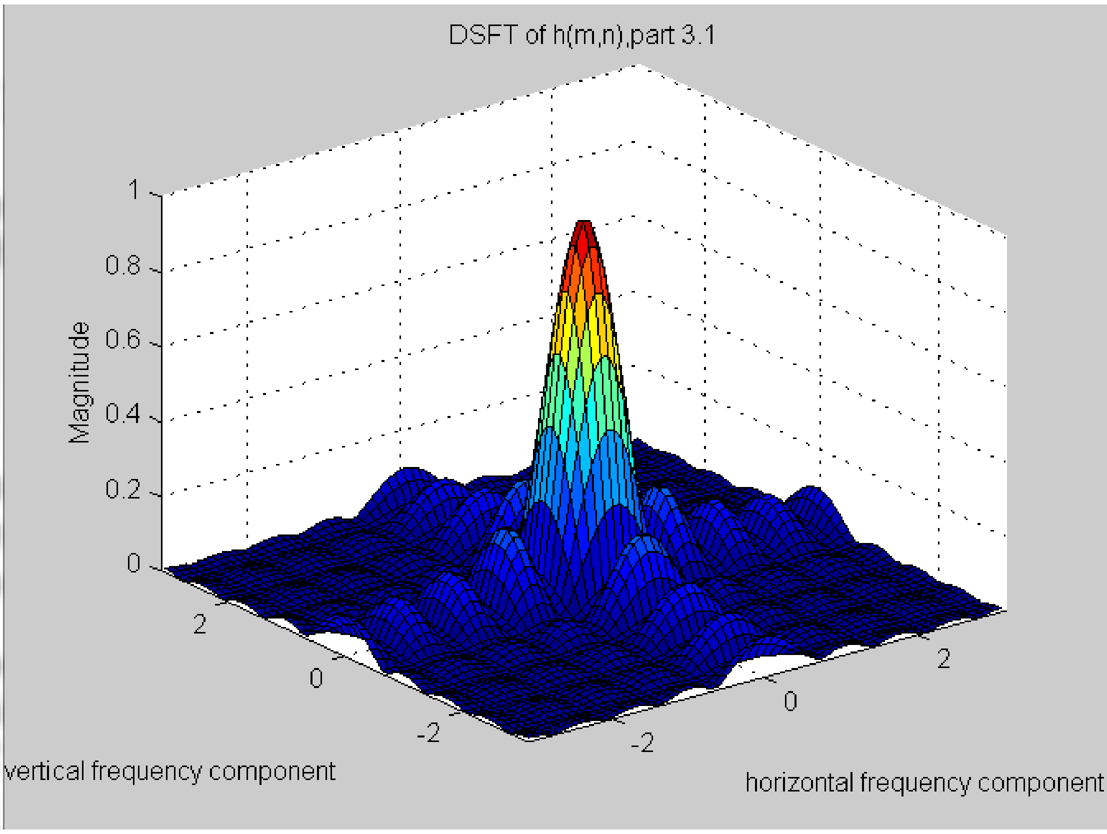
$$H(e^{ju}, e^{jv}) = \sum_{k=-4}^{4} \sum_{l=-4}^{4} h(k,l) e^{-j(ku+lv)}$$

$$= \frac{1}{81} \sum_{k=-4}^{4} e^{-jku} \sum_{l=-4}^{4} e^{-jlv}$$

$$= \frac{1}{81} \frac{\sin\frac{9}{2}\mu}{sin\frac{1}{2}\mu} \frac{\sin\frac{9}{2}v}{sin\frac{1}{2}v}$$

See MATLAB code in Appendix.

Part 2. Plot for the magnitude of $H(e^{j\mu}, e^{jv})$



Part 3. The color image in img03.tif

Part 4. The filtered color image

Part 5. Listed code in the Appendix

Section 4. FIR Sharpening Filter

Part 1. Analytical expression for $\boldsymbol{H(e^{j\mu}, e^{jv})}$

Since we know $|m| \leq 2$ and $|n| \leq 2$ in this question, the formula should looks like $H\left(e^{ju}, e^{jv}\right) = \sum_{k=-2}^{2} \sum_{l=-2}^{2} h(k,l) e^{-j(ku+lv)}$,

Thus, we have

$$H\left(e^{ju}, e^{jv}\right) = \sum_{k=-2}^{2} \sum_{l=-2}^{2} h(k,l) e^{-j(ku+lv)}$$

$$= \frac{1}{25} \sum_{k=-2}^{2} e^{-jku} \sum_{l=-2}^{2} e^{-jlv}$$

$$= \frac{1}{25} \frac{\sin\frac{5}{2}\mu}{sin\frac{1}{2}\mu} \frac{\sin\frac{5}{2}v}{sin\frac{1}{2}v}$$

Part 2. Analytical expression for $\boldsymbol{G(e^{j\mu}, e^{jv})}$

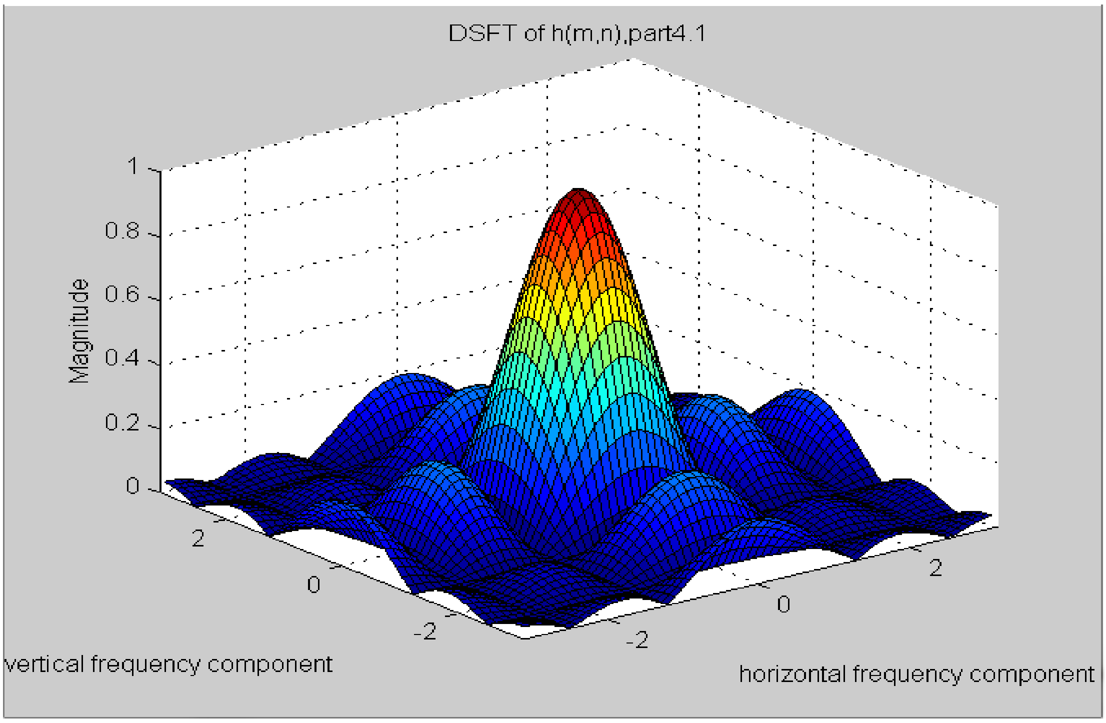
By definition, we know that $g(m,n) = \delta(m,n) + \lambda(\delta(m,n) - h(m,n))$

Thus, $G\left(e^{j\mu}, e^{jv}\right) = \lambda + 1 - \lambda H\left(e^{j\mu}, e^{jv}\right)$.

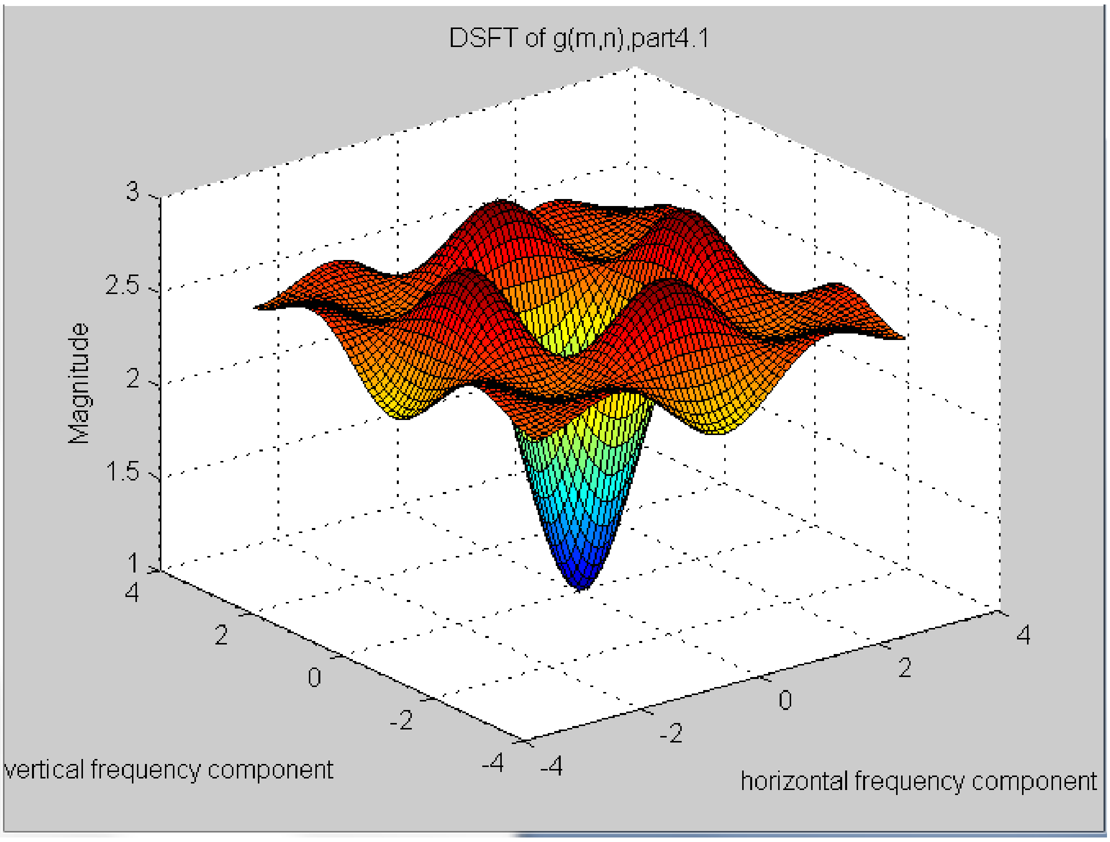
Finally, we can get $G\left(e^{j\mu}, e^{jv}\right) = \lambda + 1 - \lambda \frac{1}{25} \frac{\sin\frac{5}{2}\mu}{sin\frac{1}{2}\mu} \frac{\sin\frac{5}{2}v}{sin\frac{1}{2}v}$

See MATLAB code in Appendix.

Part 3. Plot for the magnitude of $H(e^{j\mu}, e^{jv})$



Part 4. Plot for the magnitude of $G(e^{j\mu}, e^{jv})$



Part 5. Color image of imgblur.tif

Part 6. Output sharpened color image for λ = 1.5.

Part 7. List of c code in appendix

Section 5. IIR Filter

Part 1. Analytical expression for $H(e^{j\mu}, e^{jv})$

We know that:

$$y(m,n) = 0.01x(m,n) + 0.9\big(y(m\text{-}1,n) + y(m,n\text{-}1)\big) - 0.81y(m\text{-}1,n\text{-}1)$$

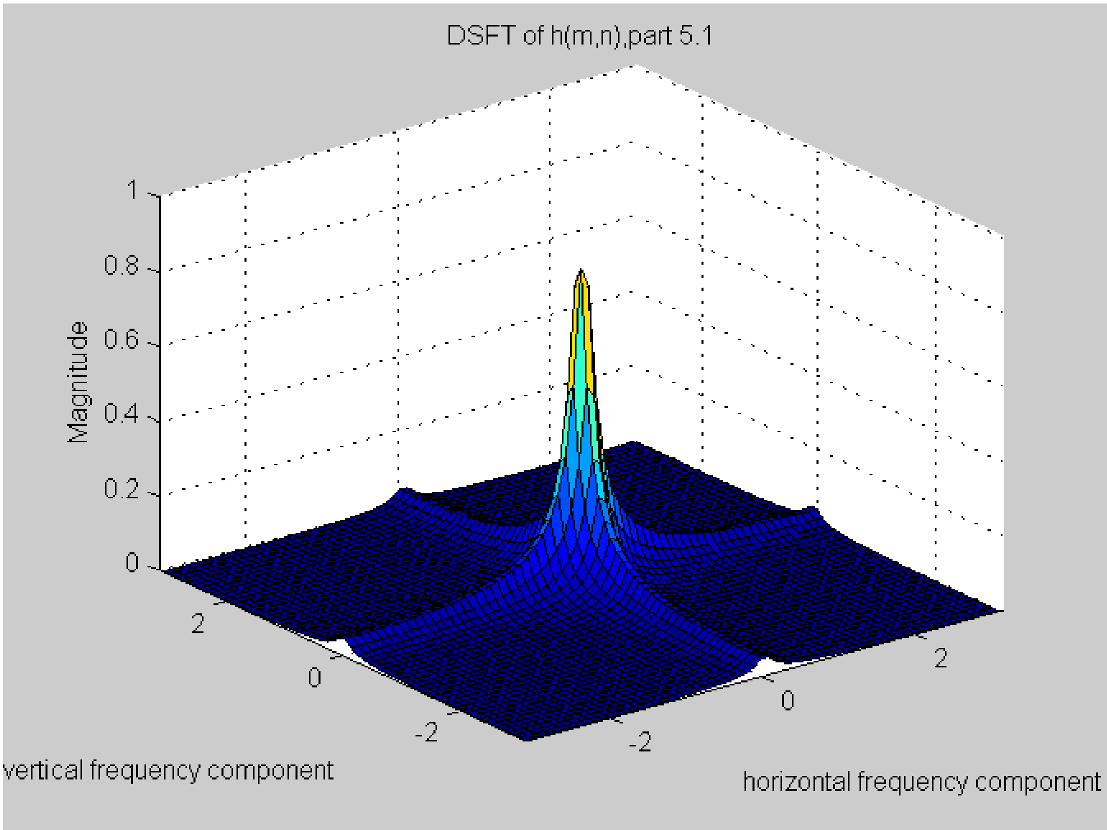
From there we can get the equation that:

$$Y(e^{j\mu}, e^{jv}) = 0.01X(e^{j\mu}, e^{jv}) + 0.9Y(e^{j\mu}, e^{jv})\big(e^{-j\mu} + e^{-jv}\big) - 0.81Y(e^{j\mu}, e^{jv})(e^{-j\mu}\, e^{-jv})$$

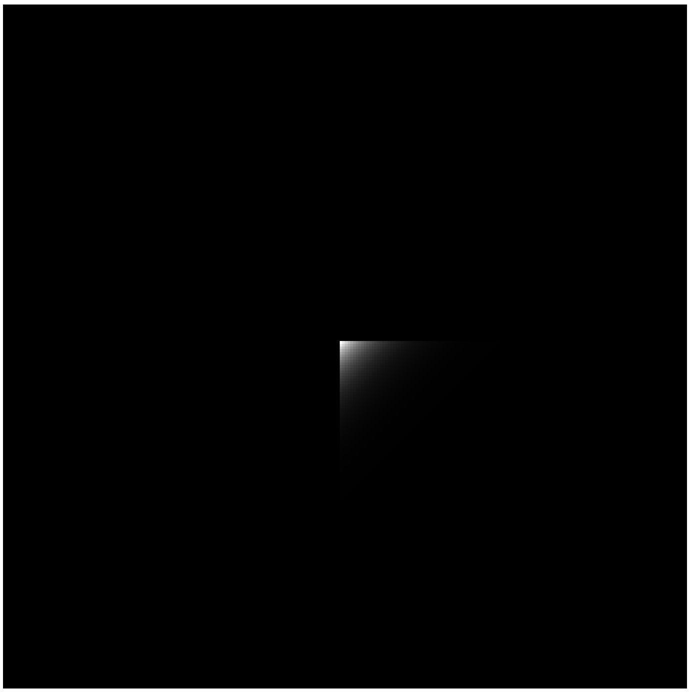
Thus: $H(e^{j\mu}, e^{jv}) = \frac{0.01}{(1-0.9e^{-j\mu})(1-0.9e^{jv})}$

See MATLAB code in Appendix.

Part 2. Plot for the magnitude of $H(e^{j\mu}, e^{jv})$



Part 3. An image of the point spread function



Part 4. The filtered output color image

Part 5. A list of c code is in appendix

Appendix

Section 3(MATLAB code):

```
clc;
clear;

h = 1/81;
H = 0;
u = -pi:0.1:pi; %horizontal frequency component
v = -pi:0.1:pi; %vertical frequency component
size_u = length(u);
size_v = length(v);
size_h = size_u * size_v;
H = zeros(size_u,size_v);
for i_ind = 1:1:size_u
    u_comp = u(i_ind);
    for j_ind = 1:1:size_v
        v_comp = v(j_ind);
        for k = -4:1:4
            for l = -4:1:4
                H(i_ind,j_ind) = H(i_ind,j_ind) +
h*exp(-i*(k*u_comp + l*v_comp));
            end
        end
    end
end
[m,n] = meshgrid(-pi :0.1: pi,-pi :0.1: pi);
H_mag = abs(H);
%mesh(m,n,H_mag)
surf(m,n,H_mag)
axis([-pi pi -pi pi 0 1])
title('DSFT of h(m,n),part 3.1')
zlabel('Magnitude')
xlabel('horizontal frequency component')
ylabel('vertical frequency component')
```

Section 3(C code):

```c
#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"

void error(char *name);

int main (int argc, char **argv)
{
  FILE *fp;
  struct TIFF_img input_img, color_img;
  double **img_red,**img_green,**img_blue;

  int32_t i,j,k,l;
  double r_value,g_value,b_value;

  if ( argc != 2 ) error( argv[0] );

  /* open image file */
  if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
    fprintf ( stderr, "cannot open file %s\n", argv[1] );
    exit ( 1 );
  }

  /* read image */
  if ( read_TIFF ( fp, &input_img ) ) {
    fprintf ( stderr, "error reading file %s\n", argv[1] );
    exit ( 1 );
  }

  /* close image file */
  fclose ( fp );

  /* check the type of image data */
  if ( input_img.TIFF_type != 'c' ) {
    fprintf ( stderr, "error:  image must be 24-bit color\n" );
    exit ( 1 );
  }

  /* Allocate image of double precision floats */
```

```
  img_red = (double
**)get_img(input_img.width+8,input_img.height+8,sizeof(double));//imgsize +
filtersize
  //img2 = (double **)get_img(input_img.width,input_img.height,sizeof(double));
  img_green = (double
**)get_img(input_img.width+8,input_img.height+8,sizeof(double));
  img_blue = (double
**)get_img(input_img.width+8,input_img.height+8,sizeof(double));


  /* set up structure for output color image */
  /* Note that the type is 'c' rather than 'g' */
  get_TIFF ( &color_img, input_img.height, input_img.width, 'c' );//the output img

  for (i = 0;i<input_img.height+8;i++){
    for(j = 0;j<input_img.width+8;j++){
      img_red[i][j] = 0;
      img_green[i][j] = 0;
      img_blue[i][j] = 0;
    }
  }

  for(i = 4;i<input_img.height+4;i++){
    for(j = 4;j<input_img.width+4;j++){
      img_red[i][j] = input_img.color[0][i-4][j-4];
      img_green[i][j] = input_img.color[1][i-4][j-4];
      img_blue[i][j] = input_img.color[2][i-4][j-4];     //create new imgs with
boundary filled with 0
    }
  }

  /* Illustration: constructing a sample color image -- interchanging the red and
green components from the input color image */
  for ( i = 0; i < input_img.height; i++ ){
      for ( j = 0; j < input_img.width; j++ ) {
       r_value = 0;
       g_value = 0;
       b_value = 0;
       for (k = -4; k <=4; k++ ){
               for ( l = -4; l <= 4; l++ ){
                  r_value = r_value + (img_red[i+4-k][j+4-l])/81.0;   //i+4,j+4 are
the index should start to read
                  g_value = g_value + (img_green[i+4-k][j+4-l])/81.0;
                  b_value = b_value + (img_blue[i+4-k][j+4-l])/81.0;
```

```c
                }
              }
            color_img.color[0][i][j] = (int)r_value;
            color_img.color[1][i][j] = (int)g_value;
            color_img.color[2][i][j] = (int)b_value;
      }
}



  /* open color image file */
  if ( ( fp = fopen ( "result_Q3.tif", "wb" ) ) == NULL ) {
      fprintf ( stderr, "cannot open file color.tif\n");
      exit ( 1 );
  }

  /* write color image */
  if ( write_TIFF ( fp, &color_img ) ) {
      fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
      exit ( 1 );
  }

  /* close color image file */
  fclose ( fp );

  /* de-allocate space which was used for the images */
  free_TIFF ( &(input_img) );
  free_TIFF ( &(color_img) );
  free_img( (void**)img_red );
  free_img( (void**)img_blue );
  free_img( (void**)img_green );
  return(0);
}

void error(char *name)
{
    printf("usage:  %s  image.tiff \n\n",name);
    printf("this program reads in a 24-bit color TIFF image.\n");
    printf("It then horizontally filters the green component, adds noise,\n");
    printf("and writes out the result as an 8-bit image\n");
    printf("with the name 'green.tiff'.\n");
    printf("It also generates an 8-bit color image,\n");
    printf("that swaps red and green components from the input image");
    exit(1);
```

```
}
```

Section 4(MATLAB):

```
clc;
clear;

lamda = 1.5;
h = 1/25;
d = 1;
H = 0;
u = -pi:0.1:pi; %horizontal frequency component
v = -pi:0.1:pi; %vertical frequency component
size_u = length(u);
size_v = length(v);
size_h = size_u * size_v;
H = zeros(size_u,size_v); %function H
D = zeros(size_u,size_v); %the delta function
for i_ind = 1:1:size_u
    u_comp = u(i_ind);
    for j_ind = 1:1:size_v
        v_comp = v(j_ind);
        for k = -2:1:2
            for l = -2:1:2
                H(i_ind,j_ind) = H(i_ind,j_ind) + 
h*exp(-i*(k*u_comp + l*v_comp));
                if ((k == 0) & (l == 0))
                    D(i_ind,j_ind) = d*exp(-i*(k*u_comp + 
l*v_comp));
                end
            end
        end
    end
end
[m,n] = meshgrid(-pi :0.1: pi,-pi :0.1: pi);
H_mag = abs(H);
%mesh(m,n,H_mag)
figure(1);
surf(m,n,H_mag);
axis([-pi pi -pi pi 0 1]);
title('DSFT of h(m,n),part4.1');
zlabel('Magnitude');
xlabel('horizontal frequency component');
ylabel('vertical frequency component');
```

```
G = zeros(size_u,size_v); %function G
G = D+lamda*(D-H);
G_mag = abs(G);
figure(2);
surf(m,n,G_mag);
%axis([-pi pi -pi pi 0 1]);
title('DSFT of g(m,n),part4.1');
zlabel('Magnitude');
xlabel('horizontal frequency component');
ylabel('vertical frequency component');
```

Section 4(C code):

```
#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"

void error(char *name);

int main (int argc, char **argv)
{
  FILE *fp;
  struct TIFF_img input_img, color_img;
  double **img_red,**img_green,**img_blue;

  float lamda = atof(argv[2]);

  int32_t i,j,k,l;
  double r_value,g_value,b_value,r_temp,g_temp,b_temp;

  if ( argc != 3 ) error( argv[0] );

  /* open image file */
  if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
    fprintf ( stderr, "cannot open file %s\n", argv[1] );
    exit ( 1 );
  }

  /* read image */
  if ( read_TIFF ( fp, &input_img ) ) {
    fprintf ( stderr, "error reading file %s\n", argv[1] );
    exit ( 1 );
```

```
  }

  /* close image file */
  fclose ( fp );

  /* check the type of image data */
  if ( input_img.TIFF_type != 'c' ) {
    fprintf ( stderr, "error:  image must be 24-bit color\n" );
    exit ( 1 );
  }

  /* Allocate image of double precision floats */
  img_red = (double
**)get_img(input_img.width+4,input_img.height+4,sizeof(double));//imgsize +
filtersize
  img_green = (double
**)get_img(input_img.width+4,input_img.height+4,sizeof(double));
  img_blue = (double
**)get_img(input_img.width+4,input_img.height+4,sizeof(double));


  /* set up structure for output color image */
  /* Note that the type is 'c' rather than 'g' */
  get_TIFF ( &color_img, input_img.height, input_img.width, 'c' );//the output img

  for (i = 0;i<input_img.height+4;i++){
    for(j = 0;j<input_img.width+4;j++){
      img_red[i][j] = 0;
      img_green[i][j] = 0;
      img_blue[i][j] = 0;
    }
  }

  for(i = 2;i<input_img.height+2;i++){
    for(j = 2;j<input_img.width+2;j++){
      img_red[i][j] = input_img.color[0][i-2][j-2];
      img_green[i][j] = input_img.color[1][i-2][j-2];
      img_blue[i][j] = input_img.color[2][i-2][j-2];     //create new imgs with
boundary filled with 0
    }
  }

  /* Illustration: constructing a sample color image -- interchanging the red and
green components from the input color image */
```

```
  for ( i = 0; i < input_img.height; i++ ){
      for ( j = 0; j < input_img.width; j++ ) {
      r_value = 0;
      g_value = 0;
      b_value = 0;
      for (k = -2; k <=2; k++ ){
              for ( l = -2; l <= 2; l++ ){
                 r_value = r_value + (img_red[i+2-k][j+2-l])/25.0;   //i+4,j+4 are
the index should start to read
                 g_value = g_value + (img_green[i+2-k][j+2-l])/25.0;
                 b_value = b_value + (img_blue[i+2-k][j+2-l])/25.0;
               }
             }
      r_temp = ((lamda+1)*img_red[i+2][j+2] - lamda*r_value);
      g_temp = ((lamda+1)*img_green[i+2][j+2] - lamda*g_value);
      b_temp = ((lamda+1)*img_blue[i+2][j+2] - lamda*b_value);

      /*pixel limitation*/
      if (r_temp < 0){
        r_temp = 0;
      }
      else if (r_temp > 255){
        r_temp = 255;
      }
      if (g_temp < 0){
        g_temp = 0;
      }
      else if(g_temp > 255){
        g_temp = 255;
      }
      if(b_temp < 0){
        b_temp = 0;
      }
      else if(b_temp > 255){
        b_temp = 255;
      }

      color_img.color[0][i][j] = (int)(r_temp);
      color_img.color[1][i][j] = (int)(g_temp);
      color_img.color[2][i][j] = (int)(b_temp);
      }
}
```

```c
  /* open color image file */
  if ( ( fp = fopen ( "Revised_result_Q4_lamda=1.5.tif", "wb" ) ) == NULL ) {
      fprintf ( stderr, "cannot open file color.tif\n");
      exit ( 1 );
  }

  /* write color image */
  if ( write_TIFF ( fp, &color_img ) ) {
      fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
      exit ( 1 );
  }

  /* close color image file */
  fclose ( fp );

  /* de-allocate space which was used for the images */
  free_TIFF ( &(input_img) );
  free_TIFF ( &(color_img) );
  free_img( (void**)img_red );
  free_img( (void**)img_blue );
  free_img( (void**)img_green );

  return(0);
}

void error(char *name)
{
    printf("usage:  %s  image.tiff \n\n",name);
    printf("this program reads in a 24-bit color TIFF image.\n");
    printf("It then horizontally filters the green component, adds noise,\n");
    printf("and writes out the result as an 8-bit image\n");
    printf("with the name 'green.tiff'.\n");
    printf("It also generates an 8-bit color image,\n");
    printf("that swaps red and green components from the input image");
    exit(1);
}
```

Section 5(MATLAB):

```matlab
clc;
clear;

u = -pi:0.1:pi; %horizontal frequency component
v = -pi:0.1:pi; %vertical frequency component
```

```
size_u = length(u);
size_v = length(v);
size_h = size_u * size_v;
H = zeros(size_u,size_v);
for i_ind = 1:1:size_u
    u_comp = u(i_ind);
    for j_ind = 1:1:size_v
        v_comp = v(j_ind);
        H(i_ind,j_ind) =
0.01*(1./((1-0.9*exp(-i*u_comp)).*(1-0.9*exp(-i*v_comp)))
);
    end
end
[m,n] = meshgrid(-pi :0.1: pi,-pi :0.1: pi);
H_mag = abs(H);
%mesh(m,n,H_mag)
surf(m,n,H_mag)
axis([-pi pi -pi pi 0 1])
title('DSFT of h(m,n),part 5.1')
zlabel('Magnitude')
xlabel('horizontal frequency component')
ylabel('vertical frequency component')

X = zeros(256,256);
X(127,127) = 1;
Y = zeros(257,257);% add a row a col with zeros for calculation
conviance
for m = 1:1:256
    for n = 1:1:256

Y(m+1,n+1)=0.01*X(m,n)+0.9*Y(m,n+1)+0.9*Y(m+1,n)-0.81*Y(m
,n);
    end
end

Y_final = zeros(256,256);
for m = 1:1:256
    for n = 1:1:256
        Y_final(m,n)=Y(m+1,n+1); %get rid of the extra row and
col
    end
end

imwrite(uint8(255*100*Y_final),'h_out.tif');
```

Section 5(C code):

```
#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"

void error(char *name);

int main (int argc, char **argv)
{
  FILE *fp;
  struct TIFF_img input_img, color_img;
  double **img_red,**img_green,**img_blue,**img_0,**img_1,**img_2;


  int32_t i,j;

  if ( argc != 2 ) error( argv[0] );

  /* open image file */
  if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
  //if ( ( fp = fopen ("E:\2016spring\ECE637\lab1_image_filter\Debug\img12.tif",
"rb") ) == NULL ) {
    fprintf ( stderr, "cannot open file %s\n", argv[1] );
    exit ( 1 );
  }

  /* read image */
  if ( read_TIFF ( fp, &input_img ) ) {
    fprintf ( stderr, "error reading file %s\n", argv[1] );
    exit ( 1 );
  }

  /* close image file */
  fclose ( fp );

  /* check the type of image data */
  if ( input_img.TIFF_type != 'c' ) {
    fprintf ( stderr, "error:  image must be 24-bit color\n" );
    exit ( 1 );
```

```
  }

  /* Allocate image of double precision floats */
  img_red = (double
**)get_img(input_img.width+1,input_img.height+1,sizeof(double));//imgsize +
filtersize
  //img2 = (double **)get_img(input_img.width,input_img.height,sizeof(double));
  img_green = (double
**)get_img(input_img.width+1,input_img.height+1,sizeof(double));
  img_blue = (double
**)get_img(input_img.width+1,input_img.height+1,sizeof(double));


  img_0= (double **)get_img(input_img.width,input_img.height,sizeof(double));
  img_1 = (double **)get_img(input_img.width,input_img.height,sizeof(double));
  img_2 = (double **)get_img(input_img.width,input_img.height,sizeof(double));
  /* set up structure for output color image */
  /* Note that the type is 'c' rather than 'g' */
  get_TIFF ( &color_img, input_img.height, input_img.width, 'c' );//the output img

  for (i = 0;i<input_img.height+1;i++){   //In the question, consider
img_red,img_blue,img_green as y(m,n), an extra col added to the left, an extra row
added to the top, with value 0.
    for(j = 0;j<input_img.width+1;j++){
      img_red[i][j] = 0;
      img_green[i][j] = 0;
      img_blue[i][j] = 0;
    }
  }

  for (i = 0;i<input_img.height;i++){
    for(j = 0;j<input_img.width;j++){
      img_0[i][j] = input_img.color[0][i][j];
      img_1[i][j] = input_img.color[1][i][j];
      img_2[i][j] = input_img.color[2][i][j];
    }
  }
  /* Illustration: constructing a sample color image -- interchanging the red and
green components from the input color image */
  for ( i = 0; i < input_img.height; i++ ){
      for ( j = 0; j < input_img.width; j++ ) {
      img_red[i+1][j+1] = 0.01*img_0[i][j]
+0.9*(img_red[i][j+1]+img_red[i+1][j])-0.81*img_red[i][j];
        img_green[i+1][j+1] = 0.01*img_1[i][j]
```

```
+0.9*(img_green[i][j+1]+img_green[i+1][j])-0.81*img_green[i][j];
      img_blue[i+1][j+1] = 0.01*img_2[i][j]
+0.9*(img_blue[i][j+1]+img_blue[i+1][j])-0.81*img_blue[i][j];

          if ((int32_t)img_red[i+1][j+1]<0)
         {
           img_red[i+1][j+1]=0;
         }
         else if ((int32_t)img_red[i+1][j+1]>255)
         {
           img_red[i+1][j+1]=255;
         }

  if ((int32_t)img_green[i+1][j+1]<0)
         {
           img_green[i+1][j+1]=0;
         }
         else if ((int32_t)img_green[i+1][j+1]>255)
         {
           img_green[i+1][j+1]=255;
         }

  if ((int32_t)img_blue[i+1][j+1]<0)
         {
           img_blue[i+1][j+1]=0;
         }
         else if ((int32_t)img_blue[i+1][j+1]>255)
         {
           img_blue[i+1][j+1]=255;
         }

      }
}

 /*pixel value limitation*/
for ( i = 0; i < input_img.height; i++ ){
  for ( j = 0; j < input_img.width; j++ ) {
           color_img.color[0][i][j]=img_red[i+1][j+1]; //jump the first row an
col,these are 0s, created for the calculation conviance.

           color_img.color[1][i][j]=img_green[i+1][j+1];

           color_img.color[2][i][j]=img_blue[i+1][j+1];
```

```
      }
    }

  /* open color image file */
  if ( ( fp = fopen ( "result_Q5.tif", "wb" ) ) == NULL ) {
      fprintf ( stderr, "cannot open file color.tif\n");
      exit ( 1 );
  }

  /* write color image */
  if ( write_TIFF ( fp, &color_img ) ) {
      fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
      exit ( 1 );
  }

  /* close color image file */
  fclose ( fp );
  /* de-allocate space which was used for the images */
  free_TIFF ( &(input_img) );
  free_TIFF ( &(color_img) );
  free_img( (void**)img_red );
  free_img( (void**)img_blue );
  free_img( (void**)img_green );
  free_img( (void**)img_0 );
  free_img( (void**)img_1 );
  free_img( (void**)img_2 );
  return(0);
}
void error(char *name)
{
    printf("usage:  %s  image.tiff \n\n",name);
    printf("this program reads in a 24-bit color TIFF image.\n");
    printf("It then horizontally filters the green component, adds noise,\n");
    printf("and writes out the result as an 8-bit image\n");
    printf("with the name 'green.tiff'.\n");
    printf("It also generates an 8-bit color image,\n");
    printf("that swaps red and green components from the input image");
    exit(1);
}
```