# Digital Image Processing Laboratory 1

Image Filtering

Praneet Singh

September 13, 2020

Note: The **clipping()**and **_2Dconv()** functions are listed in **"utils.c"** which is provided at the end of this document along with it's header file. The **MATLAB** code to generate plots is also provided at the end.

## 1 FIR Low Pass Filter

In this problem, we have to analyze and implement a simple low pass filter given by the 9x9 point spread function,

$$h(m,n) = \begin{cases} \frac{1}{81} & for |m| \leq 4 \ \ and \ \ |n| \leq 4 \\ 0 & otherwise \end{cases}$$

### 1.1 DSFT of $h(m,n)$

$$\begin{aligned} H(u,v) &= \sum_{m=-4}^{4} \sum_{n=-4}^{4} \frac{1}{81} e^{-j(mu+nv)} \\ &= \frac{1}{81} \sum_{m=-4}^{4} e^{-jmu} \sum_{n=-4}^{4} e^{-jnv} \end{aligned}$$

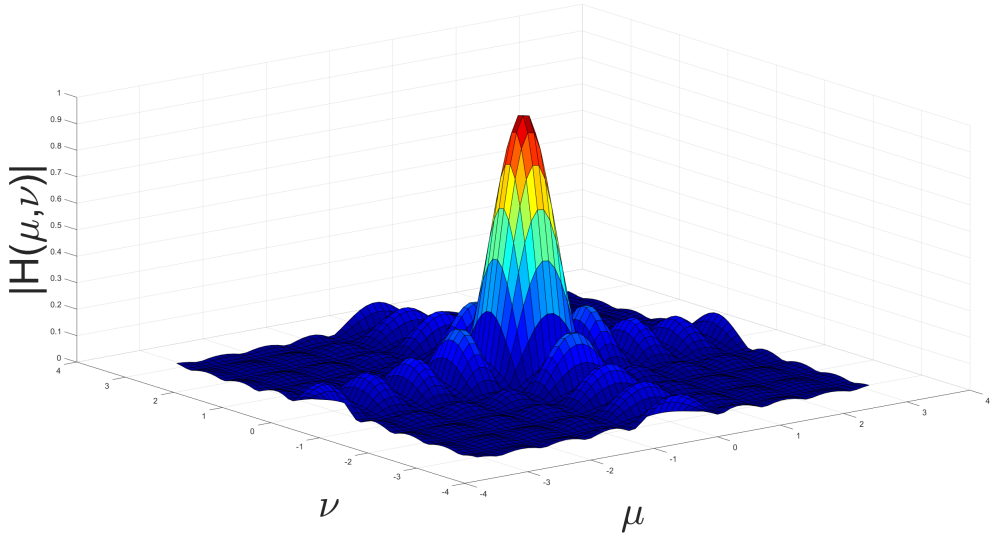
We also know that,

$$\sum_{n=-N}^{N} e^{-jwn} = e^{jwN} . \frac{1 - e^{-(2N+1)jw}}{1 - e^{-jw}} \tag{1}$$

Thus, using (1) we get,

$$\begin{aligned} H(u,v) &= \frac{1}{81} \left\{ e^{ju4} . \frac{1 - e^{-9ju}}{1 - e^{-ju}} \right\} \left\{ e^{jv4} . \frac{1 - e^{-9jv}}{1 - e^{-jv}} \right\} \\ &= \frac{1}{81} \left\{ \frac{sin\left(\frac{9u}{2}\right)}{sin\left(\frac{u}{2}\right)} \right\} \left\{ \frac{sin\left(\frac{9v}{2}\right)}{sin\left(\frac{v}{2}\right)} \right\} \end{aligned}$$

## 1.2 Plot of frequency response of $|H(u,v)|$




## 1.3 Original Color Image & Filtered Color Image

## 1.4 C-Code

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

void error(char *name);

int main (int argc, char **argv)
{
  FILE *fp;
```

```
  struct TIFF_img input_img, output_img;
  int32_t i,j;
  int32_t fil_h=9;
  int32_t fil_w=9;
  float *f_low_pass[fil_h];

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

  /* allocating memory for filter */
  for(i=0;i<fil_h;i++){
    f_low_pass[i]=(float *)malloc(fil_w * sizeof(float));
  }
 /* assigning filter values */
  for(i=0;i< fil_h;i++){
    for(j=0;j<fil_w;j++){
    f_low_pass[i][j]=1.0/81;
    }
  }

  /* 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 );
  }

  /* set up structure for output achromatic image */
  /* to allocate a full color image use type 'c' */
  get_TIFF ( &output_img, input_img.height, input_img.width, 'c' );

  /* Convolution Function called from utils.c */
  _2Dconv(f_low_pass,fil_h,fil_w,&input_img,&output_img);

  /* open output image file */
  if ( ( fp = fopen ( "firlow-pass-output.tif", "wb" ) ) == NULL ) {
    fprintf ( stderr, "cannot open file green.tif\n");
    exit ( 1 );
  }

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

  /* close output image file */
```

```
  fclose ( fp );

  /* de-allocate space which was used for the images */
  free_TIFF ( &(input_img) );
  free_TIFF ( &(output_img) );

  /* free filter memory */
  for(i=0;i<fil_h;i++){
    free(f_low_pass[i]);
  }
  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);
}
```

# 2 FIR Sharpening Filter

In this problem, we will analyze the effect of a sharpening filter known as an unsharp mask.The terminology comes from the fact that an unsharp mask filter removes the unsharp(low frequency) component of the image, and therefore produces an image with a sharper appearance.

Let $h(m,n)$ as shown below be the low-pass filter,

$$h(m,n) = \begin{cases} \frac{1}{25} & for|m| \leq 2 \ \ and \ \ |n| \leq 2 \\ 0 & otherwise \end{cases}$$

The unsharp mask filter $g(m,n)$ is given as follows,

$$g(m,n) = \delta(m,n) + \lambda(\delta(m,n) - h(m,n)) \ \ , \lambda > 0$$

## 2.1 DSFT of $h(m,n)$

$$\begin{aligned} H(u,v) &= \sum_{m=-2}^{2} \sum_{n=-2}^{2} \frac{1}{25} e^{-j(mu+nv)} \\ &= \frac{1}{25} \sum_{m=-2}^{2} e^{-jmu} \sum_{n=-4}^{4} e^{-jnv} \end{aligned}$$

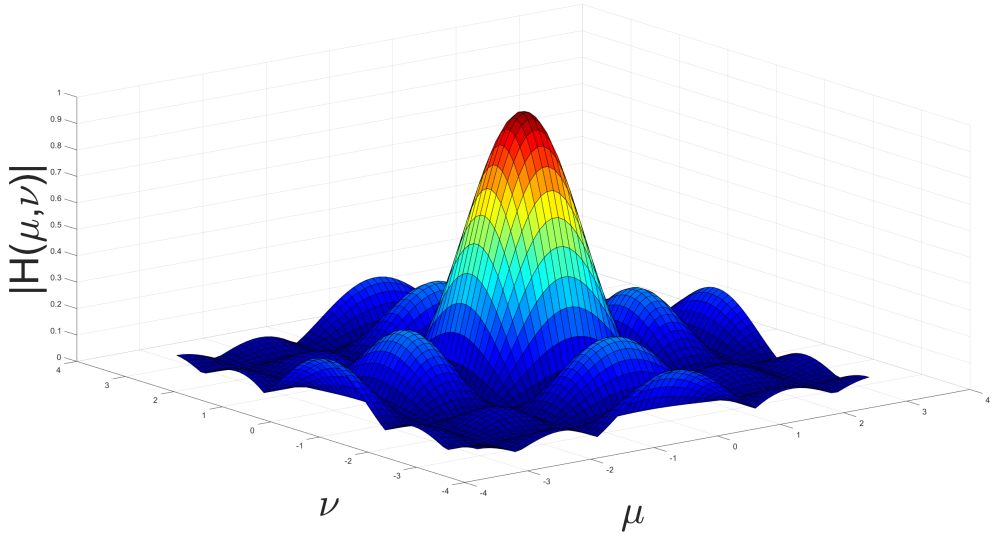
Thus, using (1) we get,

$$H(u,v) = \frac{1}{25}\left\{e^{ju2}.\frac{1-e^{-5ju}}{1-e^{-ju}}\right\}\left\{e^{jv5}.\frac{1-e^{-5jv}}{1-e^{-jv}}\right\}$$
$$= \frac{1}{25}\left\{\frac{sin\left(\frac{5u}{2}\right)}{sin\left(\frac{u}{2}\right)}\right\}\left\{\frac{sin\left(\frac{5v}{2}\right)}{sin\left(\frac{v}{2}\right)}\right\})$$

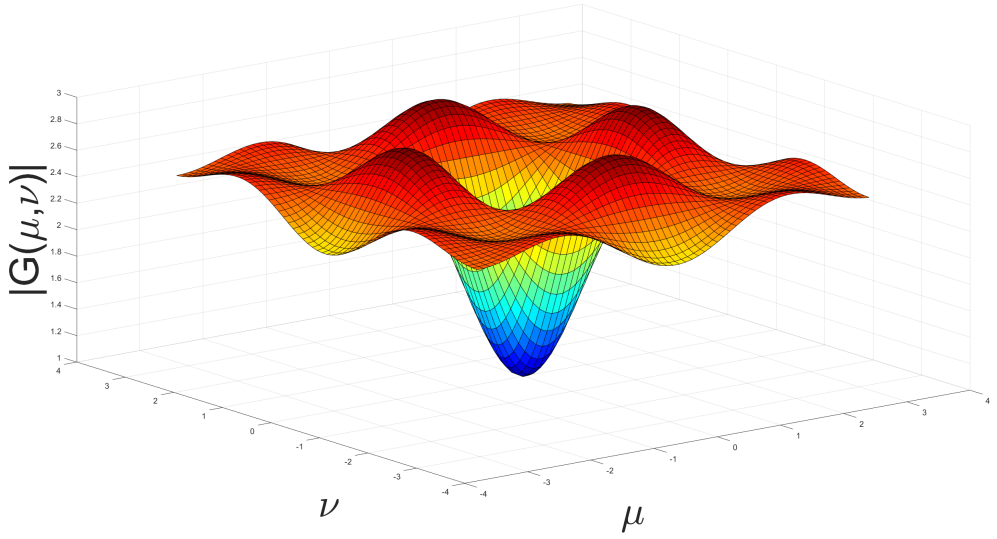
## 2.2 DSFT of $g(m,n)$

$$G(u,v) = 1 + \lambda(1 - H(u,v))$$
$$= 1 + \lambda\left(1 - \frac{1}{25}\left\{\frac{sin\left(\frac{5u}{2}\right)}{sin\left(\frac{u}{2}\right)}\right\}\left\{\frac{sin\left(\frac{5v}{2}\right)}{sin\left(\frac{v}{2}\right)}\right\}\right)$$

## 2.3 Plot of frequency response of $|H(u,v)|$



## 2.4 Plot of frequency response of $|G(u,v)|$, $\lambda = 1.5$




## 2.5 Original Color Image & Filtered Color Image ($\lambda = 1.5$)

## 2.6 Filtered Images at $\lambda$ *values* $= 0.2,\ 0.8$ & $1.5$ **respectively**



We can clearly see that, as the $\lambda$ value increases, the image gets sharper.

## 2.7 C-Code

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

void error(char *name);

int main (int argc, char **argv)
{
  FILE *fp;
  struct TIFF_img input_img, output_img;
  int32_t i,j;
  int32_t fil_h=5;
  int32_t fil_w=5;
  float lambda;

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

  /* reading lambda value from user */
  sscanf(argv[2],"%f",&lambda);

  /* allocating filter memory */
  float *f_sharp[fil_h];
  for(i=0;i<fil_h;i++){
    f_sharp[i]=(float *)malloc(fil_w * sizeof(float));
  }

  /* assigning filter values */
  for(i=0;i< fil_h;i++){
    for(j=0;j<fil_w;j++){
```

```
    if (i==2 && j==2){
        f_sharp[i][j]=1.0 + lambda *(1.0 - (1.0/25));
      }
    else{
        f_sharp[i][j]=0.0 + lambda *(0.0 - (1.0/25));
    }
    }
  }

  /* 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 );
  }

  /* set up structure for output achromatic image */
  /* to allocate a full color image use type 'c' */
  get_TIFF ( &output_img, input_img.height, input_img.width, 'c' );

  /* convolution function called from utils.c */
  _2Dconv(f_sharp,fil_h,fil_w,&input_img,&output_img);

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

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

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

  /* de-allocate space which was used for the images */
  free_TIFF ( &(input_img) );
  free_TIFF ( &(output_img) );

  /*free all filter memory*/
  for(i=0;i<fil_h;i++){
    free(f_sharp[i]);
  }
```

```
94
95    return(0);
96  }
97
98  void error(char *name)
99  {
100     printf("usage:  %s  image.tiff \n\n",name);
101     printf("this program reads in a 24-bit color TIFF image.\n");
102     printf("It then horizontally filters the green component, adds noise,\n");
103     printf("and writes out the result as an 8-bit image\n");
104     printf("with the name 'green.tiff'.\n");
105     printf("It also generates an 8-bit color image,\n");
106     printf("that swaps red and green components from the input image");
107     exit(1);
108 }
```

# 3 IIR Filter

In this problem, we will analyze the effect of an IIR filter specified by a 2-D difference equation. Let $h(m,n)$ be the impulse response of an IIR filter with corresponding difference equation,

$$y(m,n) = 0.01x(m,n) + 0.9(y(m-1,n) + y(m,n-1)) - 0.81y(m-1,n-1)$$

where x(m,n) is the input & y(m,n) is the output.

## 3.1 DSFT of $h(m,n)$

Applying Z-transform to the above equation, we get

$$Y(z_1,z_2) = 0.01X(z_1,z_2) + 0.9(z_1^{-1}Y(z_1,z_2) + z_2^{-1}Y(z_1,z_2)) - 0.81z_1^{-1}z_2^{-1}Y(z_1,z_2)$$
$$\frac{Y(z_1,z_2)}{X(z_1,z_2)} = \frac{0.01}{1 - 0.9z_1^{-1} - 0.9z_2^{-1} + 0.81z_1^{-1}z_2^{-1}}$$

And thus,

$$H(z_1,z_2) = \frac{0.01}{1 - 0.9z_1^{-1} - 0.9z_2^{-1} + 0.81z_1^{-1}z_2^{-1}}$$

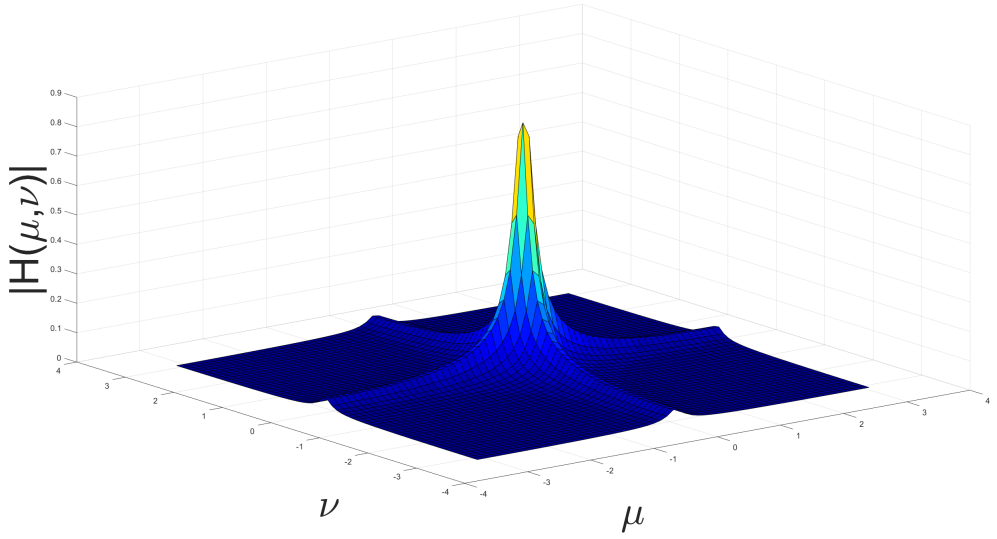
We know that,

$$H(w) = H(z)|_{z=e^{-jw}}$$

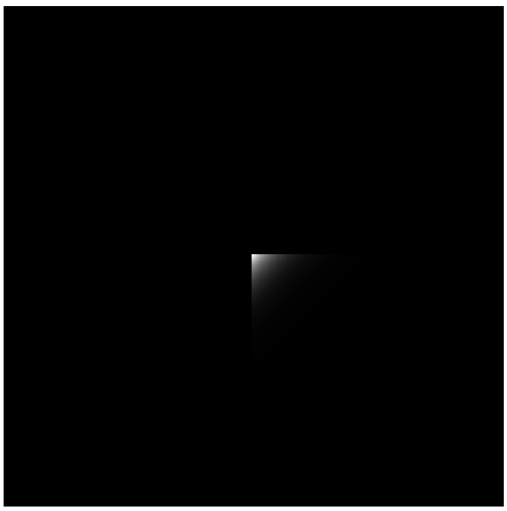
Let $z_1 = e^{-ju}$ and $z_2 = e^{-jv}$.
We then get,

$$H(u,v) = \frac{0.01}{1 - 0.9e^{-ju} - 0.9e^{-jv} + 0.81e^{-j(u+v)}}$$

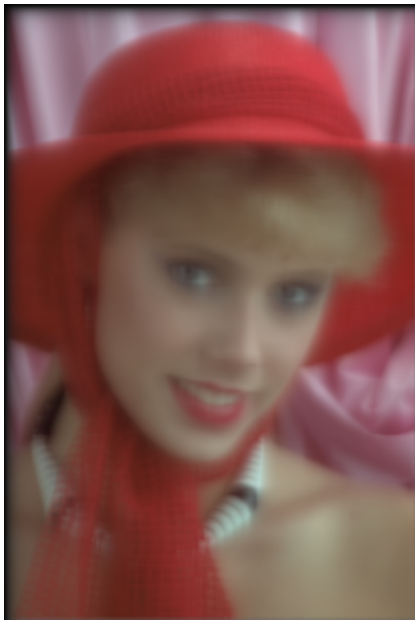
### 3.2 Plot of frequency response of $|H(u, v)|$




### 3.3 Image of the Point Spread Function



## 3.4 Original Color Image & Filtered Color Image



## 3.5 C-Code

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

void error(char *name);

int main (int argc, char **argv)
{
  FILE *fp;
  struct TIFF_img input_img, output_img;
  int32_t i,j,m;
  int32_t img_h;
  int32_t img_w;
  int32_t channels = 3;

  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 );
```

```
36
37  /* check the type of image data */
38  if ( input_img.TIFF_type != 'c' ) {
39    fprintf ( stderr, "error:  image must be 24-bit color\n" );
40    exit ( 1 );
41  }
42
43  get_TIFF(&output_img, input_img.height, input_img.width, 'c');
44
45  /* set up structure for output achromatic image */
46  /* to allocate a full color image use type 'c' */
47  img_h=input_img.height;
48  img_w=input_img.width;
49
50  double (*_2dtemp)[img_h][img_w]=malloc(sizeof(double[3][img_h][img_w]));
51
52  for (m = 0; m < channels; m++) {
53    for (i = 0; i < img_h; i++) {
54      for (j = 0; j < img_w; j++) {
55        _2dtemp[m][i][j] = 0.01 * input_img.color[m][i][j];
56        if (i > 0) {
57          _2dtemp[m][i][j] += 0.9 * _2dtemp[m][i-1][j];
58        }
59        if (j > 0) {
60          _2dtemp[m][i][j] += 0.9 * _2dtemp[m][i][j-1];
61        }
62        if (i > 0 && j > 0) {
63          _2dtemp[m][i][j] += -0.81 * _2dtemp[m][i-1][j-1];
64        }
65
66      }
67    }
68  }
69
70  for (m = 0; m < 3; m++) {
71    for (i = 0; i < img_h; i++) {
72      for (j = 0; j < img_w; j++) {
73        output_img.color[m][i][j]=clipping(_2dtemp[m][i][j]);
74      }
75    }
76  }
77
78  /* open output image file */
79  if ( ( fp = fopen ( "iir.tif", "wb" ) ) == NULL ) {
80    fprintf ( stderr, "cannot open file green.tif\n");
81    exit ( 1 );
82  }
83
84  /* write output image */
85  if ( write_TIFF ( fp, &output_img ) ) {
86    fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
87    exit ( 1 );
88  }
89
90  /* close green image file */
91  fclose ( fp );
92
93
94  /* de-allocate space which was used for the images */
95  free_TIFF ( &(input_img) );
```

```
  free_TIFF ( &(output_img) );

  /*free all filter memory*/


  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);
}
```

# 4 Utility functions

## 4.1 utils.h

```
#include <stdio.h>
#include <math.h>

#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"

struct pixel {
  int m;
  int n;
};

struct px_linked_list {
  struct pixel px;
  struct px_linked_list *next_px;
};

void ConnectedNeighbors(struct pixel s,double T,unsigned char **img,int width,int
   height,int *M,struct pixel c[4]);

void ConnectedSet(struct pixel s,double T,unsigned char **img,int width,int height,int
    ClassLabel,unsigned int **seg,int *NumConPixels);

void CheckThreshold(struct pixel s,struct pixel p,double T, unsigned char **img,int
   width,int height,int *M,struct pixel c[4]);
```

## 4.2 utils.c

```
#include "utils.h"

void CheckThreshold(struct pixel s,struct pixel p,double T,unsigned char **img,int
   width,int height,int *M,struct pixel c[4]){

      if (p.n >= 0 && p.n <width && p.m>=0 && p.m<height){
```

```
                if (abs(img[s.m][s.n] - img[p.m][p.n]) <= T) {
                c[*M].m = p.m;
                c[*M].n = p.n;
                (*M)++;
                }
        }

}

void ConnectedNeighbors(struct pixel s,double T,unsigned char **img,int width,int
    height,int *M,struct pixel c[4]) {
        *M = 0;

  struct pixel p1,p2,p3,p4;

  p1.m=s.m-1;
  p1.n=s.n;

  p2.m=s.m+1;
  p2.n=s.n;

  p3.m=s.m;
  p3.n=s.n+1;

  p4.m=s.m;
  p4.n=s.n-1;

  CheckThreshold(s,p1,T,img,width,height,M,c);
  CheckThreshold(s,p2,T,img,width,height,M,c);
  CheckThreshold(s,p3,T,img,width,height,M,c);
  CheckThreshold(s,p4,T,img,width,height,M,c);


}

void ConnectedSet(struct pixel s,double T,unsigned char **img,int width,int height,int
    ClassLabel,unsigned int **seg,int *NumConPixels) {
  struct px_linked_list *head_px, *next_px, *tmp_px;
  struct pixel c[4];
  int M;
  int i;

  (*NumConPixels) = 0;

  head_px = (struct px_linked_list *)malloc(sizeof(struct px_linked_list));
  head_px->px.m = s.m;
  head_px->px.n = s.n;
  head_px->next_px = NULL;
  next_px = head_px;

  /*Checking to end loop */

  while (head_px != NULL) {
    if (seg[head_px->px.m][head_px->px.n] != ClassLabel) {

      seg[head_px->px.m][head_px->px.n] = ClassLabel;
      (*NumConPixels)++;
      ConnectedNeighbors(head_px->px, T, img, width, height, &M, c);
      for (i = 0; i < M; i++) {
        if (seg[c[i].m][c[i].n] != ClassLabel) {
```

```
            tmp_px = (struct px_linked_list *)malloc(sizeof(struct px_linked_list));
            tmp_px->px.m = c[i].m;
            tmp_px->px.n = c[i].n;
            tmp_px->next_px = NULL;
            next_px->next_px = tmp_px;
            next_px = tmp_px;
          }
        }
      }
      tmp_px = head_px->next_px;
      free(head_px);
      head_px = tmp_px;
    }
}
```

# 5 MATLAB Code

```
u=-pi:0.1:pi;
v=-pi:0.1:pi;

%Without meshgrid
u=u+0.0000001;
v=v+0.0000001;
new_u=sin(4.5*u)./sin(0.5*u);
new_v=sin(4.5*v)./sin(0.5*v);
res=new_u' * new_v;

res=(1/81)*res;
fig0 = figure(1);
surf(u,v,abs(res))
colormap jet
xlabel('\mu');
ylabel('\nu');
zlabel('|H(\mu,\nu)|');



[U,V] = meshgrid(u,v);
h=(1/81)*(sin(4.5*U)./sin(0.5*U)).*(sin(4.5*V)./sin(0.5*V));
fig1 = figure(2);
surf(U,V,abs(h))
colormap jet
xlabel('\mu');
ylabel('\nu');
zlabel('|H(\mu,\nu)|');




h=(1/25)*(sin(2.5*U)./sin(0.5*U)).*(sin(2.5*V)./sin(0.5*V));
fig2 = figure(3);
surf(U,V,abs(h))
colormap jet
xlabel('\mu');
ylabel('\nu');
zlabel('|H(\mu,\nu)|');

lamda=1.5;
g = 1 + lamda*(1 - h);
```

```
fig3 = figure(3);
surf(U,V,abs(g));
colormap jet
xlabel('\mu');
ylabel('\nu');
zlabel('G(\mu,\nu)');


h = 0.01./(1-0.9*exp(-1i*U)-0.9*exp(-1i*V)+0.81*exp(-1i*(U+V)));
fig4 = figure(4);
surf(U,V,abs(h));
colormap jet
xlabel('\mu');
ylabel('\nu');
zlabel('|H(\mu,\nu)|');



% PSF
x = zeros(256,256);
y = zeros(256,256);
x(128,128) = 1;

for m = 1:256
  for n = 1:256
    y(m,n) = 0.01*x(m,n);
    if (m > 1)
      y(m,n) = y(m,n) + 0.9*y(m-1,n);
    end
    if (n > 1)
      y(m,n) = y(m,n) + 0.9*y(m,n-1);
    end
    if (m > 1 && n > 1)
      y(m,n) = y(m,n) - 0.81*y(m-1,n-1);
    end
  end
end

y(y(:) > 255) = 255;
y(y(:) < 0) = 0;



%Comparing FFT of images sharpened at different values of lambda
img    = imread('fir-sharp-output02.tif');
img    = fftshift(img(:,:,2));
F      = fft2(img);
figure;
imagesc(100*log(1+abs(fftshift(F)))); colormap(gray);
title('magnitude spectrum');


img    = imread('fir-sharp-output08.tif');
img    = fftshift(img(:,:,2));
F      = fft2(img);
figure;
imagesc(100*log(1+abs(fftshift(F)))); colormap(gray);
title('magnitude spectrum');

img    = imread('fir-sharp-output15.tif');
```

```
103 img    = fftshift(img(:,:,2));
104 F      = fft2(img);
105 figure;
106 imagesc(100*log(1+abs(fftshift(F)))); colormap(gray);
107 title('magnitude spectrum');
```