ECE 637: Lab 1

Rahul Deshmukh

January 28, 2021

Section 3 Report

1. Derivation of $H(e^{j\mu}, e^{j\nu})$:

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

$$\text{Let} \quad h_1(m) = \frac{1}{9} \sum_{i=-4}^{4} \delta(m-i)$$

$$\text{Then} \quad h(m,n) = h_1(m)h_1(n)$$

$$\Rightarrow H_1(e^{ju}) = \frac{1}{9} \sum_{i=-4}^{4} e^{-jiu}$$

$$= \frac{1}{9}(1 + e^{-ju} + e^{-2ju} + e^{-3ju} + e^{-4ju} + e^{ju} + e^{2ju} + e^{3ju} + e^{4ju})$$

$$= \frac{1}{9}(1 + 2(\sum_{i=1}^{4} cos(iu)))$$

$$\Rightarrow H(e^{ju}, e^{jv}) = H_1(e^{ju})H_1(e^{jv})$$

$$= \frac{1}{81}(1 + 2(\sum_{i=1}^{4} cos(iu)))(1 + 2(\sum_{k=1}^{4} cos(kv)))$$

2. Plot of $|H(e^{j\mu}, e^{j\nu})|$

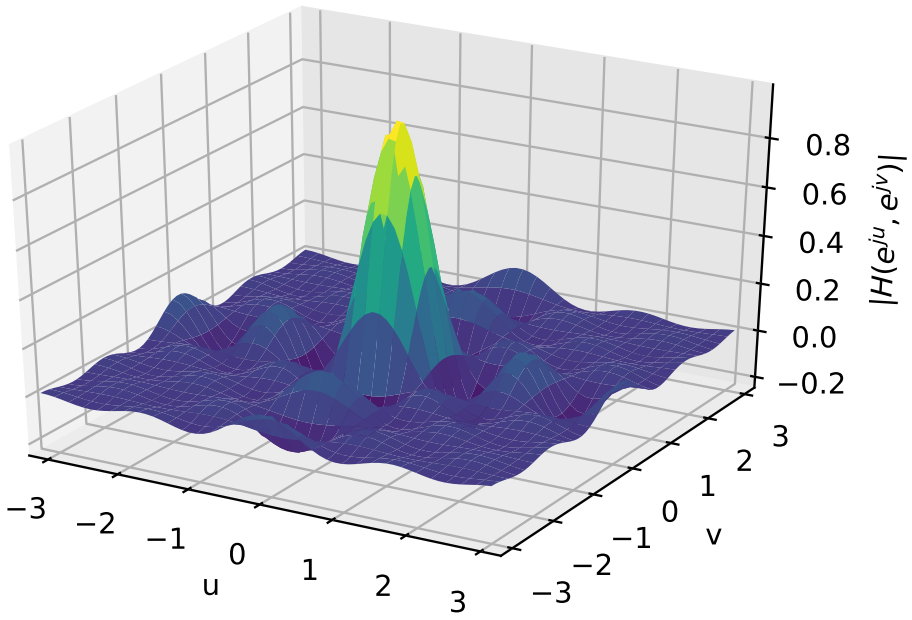


Figure 1: Surface plot

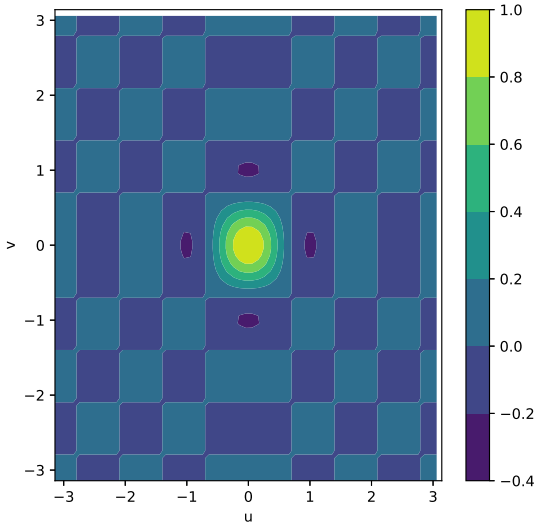


Figure 2: Contour plot

3. Color image of img03.tif

Figure 3: color image img03.tif

4. FIR Filtered image of img03.tif

Figure 4: FIR filtered color image

5. For C-code for FIR filter refer to Listing 1 at page 14.

Section 4 Report

1. Derivation of $H(e^{j\mu}, e^{j\nu})$:

$$
\begin{aligned}
h(m,n) &= \begin{cases} \frac{1}{25} & |m| \le 2, |n| \le 2 \\ 0 & \text{elsewhere} \end{cases} \\
\text{Let} \quad h_1(m) &= \frac{1}{5}\sum_{i=-2}^{2} \delta(m-i) \\
\text{Then} \quad h(m,n) &= h_1(m)h_1(n) \\
\Rightarrow H_1(e^{ju}) &= \frac{1}{5}\sum_{i=-2}^{2} e^{-jiu} \\
&= \frac{1}{5}(1 + e^{-ju} + e^{-2ju} + e^{ju} + e^{2ju}) \\
&= \frac{1}{5}(1 + 2(\sum_{i=1}^{2} cos(iu))) \\
\Rightarrow H(e^{ju}, e^{jv}) &= H_1(e^{ju})H_1(e^{jv}) \\
&= \frac{1}{25}(1 + 2(\sum_{i=1}^{2} cos(iu)))(1 + 2(\sum_{k=1}^{2} cos(kv)))
\end{aligned}
$$

2. Derivation of $G(e^{j\mu}, e^{j\nu})$:

$$
\begin{aligned}
g(m,n) &= \delta(m,n) + \lambda(\delta(m,n) - h(m,n)) \\
G(e^{ju}, e^{jv}) &= 1 + \lambda(1 - H(e^{ju}, e^{jv})) \\
&= 1 + \lambda(1 - \frac{1}{25}(1 + 2(\sum_{i=1}^{2} cos(iu)))(1 + 2(\sum_{k=1}^{2} cos(kv))))
\end{aligned}
$$

3. Plot of $|H(e^{j\mu}, e^{j\nu})|$

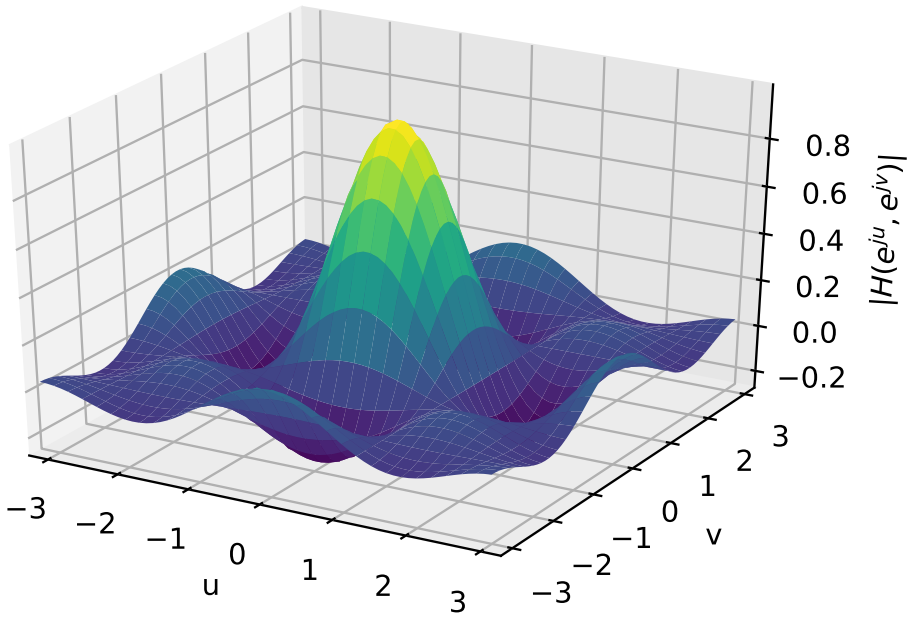


Figure 5: Surface plot

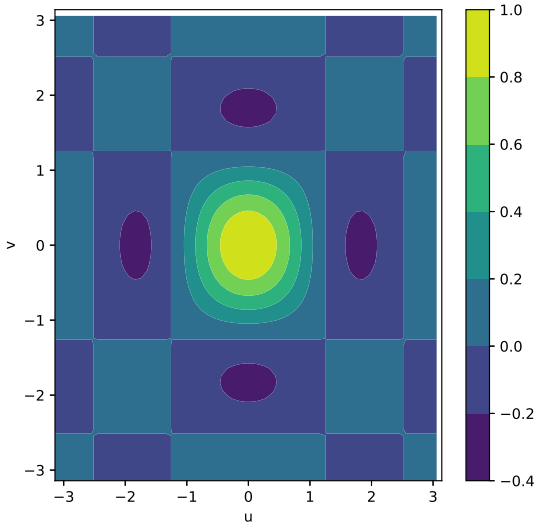


Figure 6: Contour plot

4. Plot of $|G(e^{j\mu}, e^{j\nu})|$ for $\lambda = 1.5$

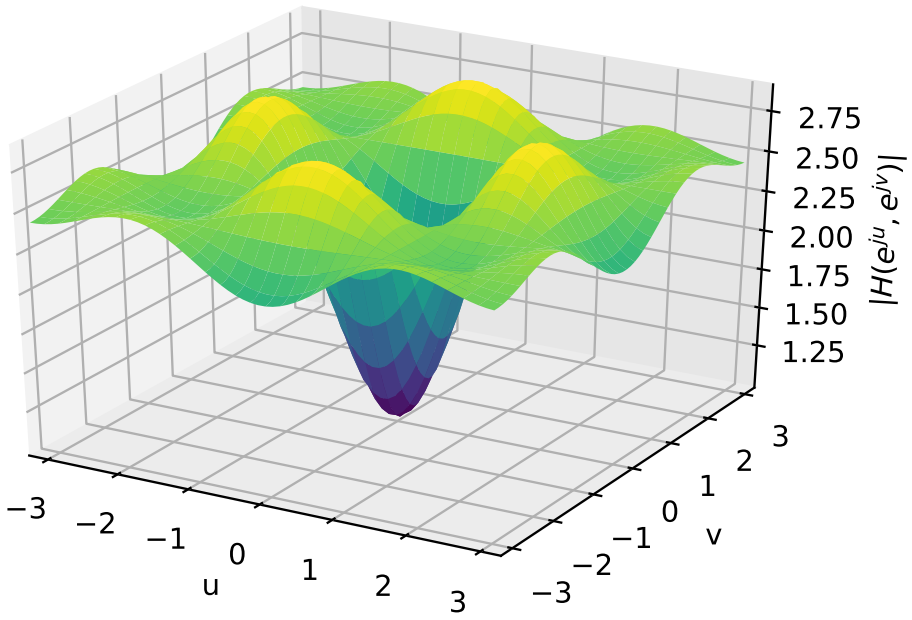


Figure 7: Surface plot

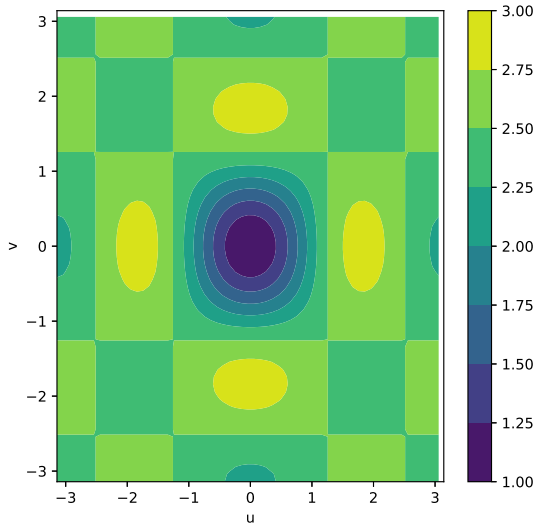


Figure 8: Contour plot

5. Input color image imgblur.tif

Figure 9: color image imgblur.tif

6. Output sharpened image for $\lambda = 1.5$

Figure 10: output sharpened image

7. For C-code for FIR sharpening filter refer to Listing 2 at page 15.

Section 5 Report

1. Derivation of $H(e^{j\mu}, e^{j\nu})$:

$$
\begin{aligned}
y(m,n) &= 0.01x(m,n) + 0.9(y(m-1,n) + y(m,n-1)) - 0.81y(m-1,n-1) \\
&\quad \text{Taking Z transform we get} \\
Y(z_1,z_2) &= 0.01X(z_1,z_2) + 0.9(z_1^{-1}Y(z_1,z_2) + z_2^{-1}Y(z_1,z_2)) - 0.81z_1^{-1}z_2^{-1}Y(z_1,z_2) \\
\Rightarrow H(z_1,z_2) &= \frac{Y(z_1,z_2)}{X(z_1,z_2)} \\
&= \frac{0.01}{0.9(z_1^{-1}+z_2^{-1}) - 0.81z_1^{-1}z_2^{-1}}\Big|_{z_1=e^{ju},z_2=e^{jv}} \\
\Rightarrow H(e^{ju},e^{jv}) &= \frac{0.01}{0.9(e^{-ju}+e^{-jv}) - 0.81e^{-ju}e^{-jv}}
\end{aligned}
$$

2. Plot of $|H(e^{j\mu}, e^{j\nu})|$

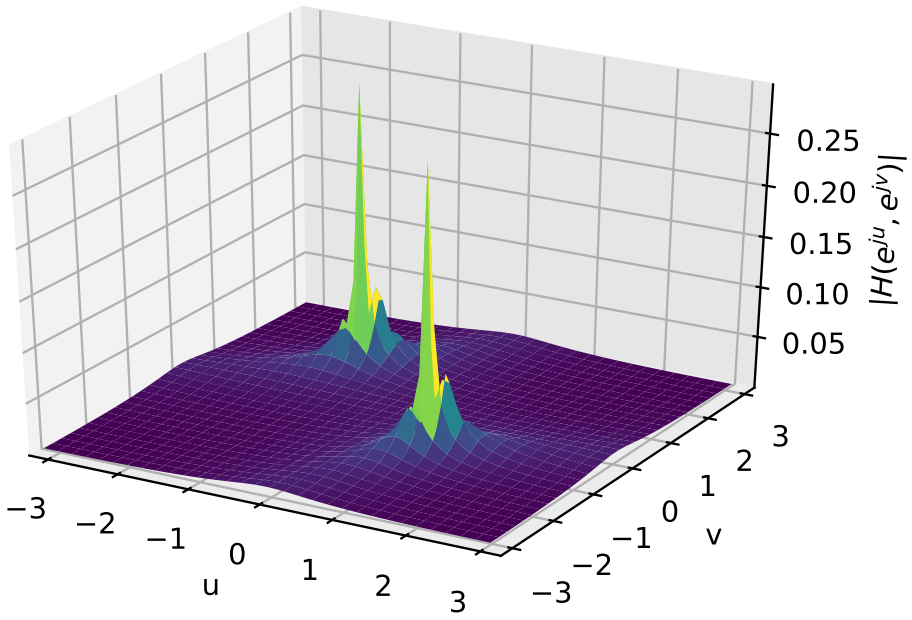


Figure 11: Surface plot

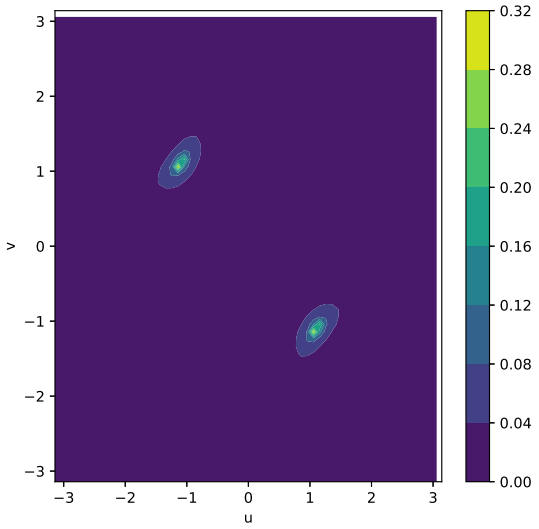
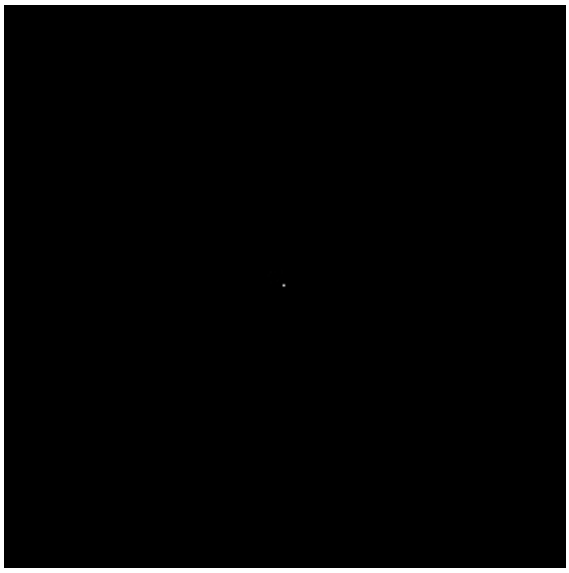
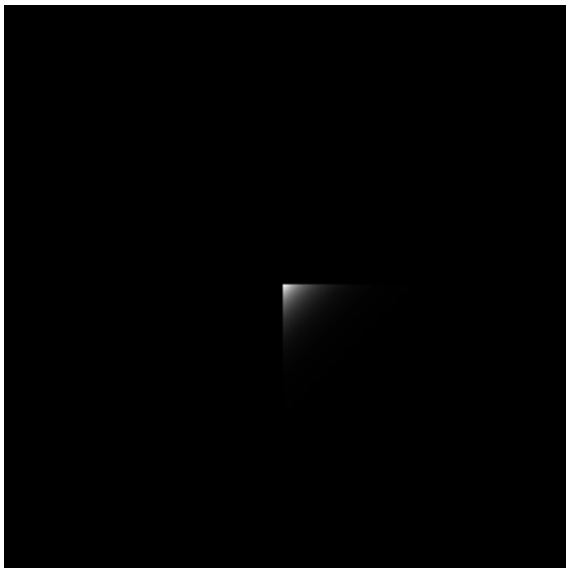


Figure 12: Contour plot

3. Image of the point spread functions



(a) image for $x(m,n) = \delta(m-127, n-127)$



(b) IIR filtered image $y(m,n)$

4. IIR Filtered output color image

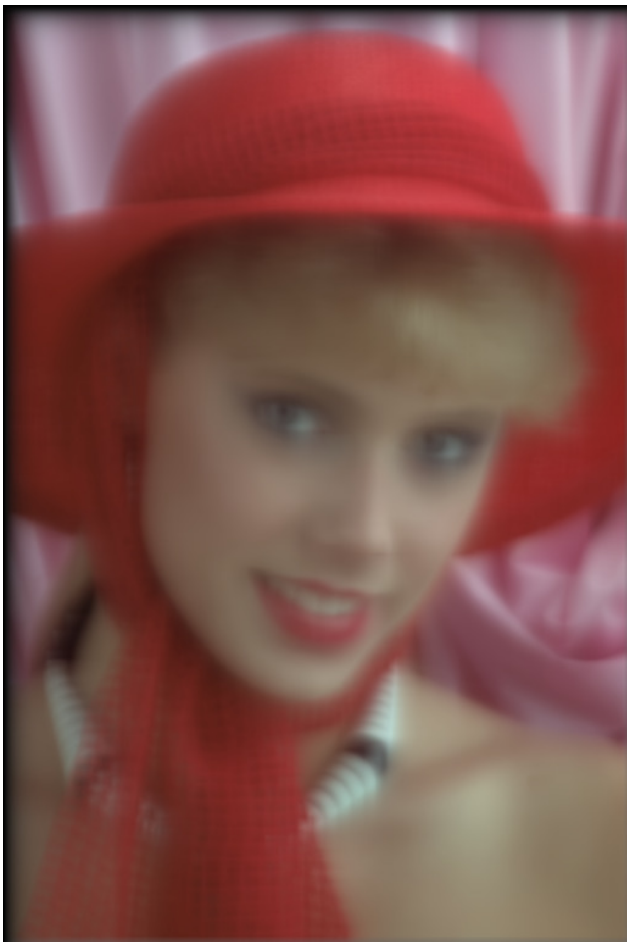


Figure 14: IIR filtered image

5. For C-code for IIR filter refer to Listing 3 at page 17.

Source Code

Listing 1: FIR

```
1
2  #include <math.h>
3  #include "tiff.h"
4  #include "allocate.h"
5  #include "randlib.h"
6  #include "typeutil.h"
7
8  void error(char *name);
9  int32_t clip(double pixel);
10
11 int main (int argc, char **argv)
12 {
13   FILE *fp;
14   struct TIFF_img input_img, color_img;
15   double h_mn;
16   int32_t i,j,k,m,n;
17
18   if ( argc != 2 ) error( argv[0] );
19
20   /* open image file */
21   if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
22     fprintf ( stderr, "cannot open file %s\n", argv[1] );
23     exit ( 1 );
24   }
25
26   /* read image */
27   if ( read_TIFF ( fp, &input_img ) ) {
28     fprintf ( stderr, "error reading file %s\n", argv[1] );
29     exit ( 1 );
30   }
31
32   /* close image file */
33   fclose ( fp );
34
35   /* check the type of image data */
36   if ( input_img.TIFF_type != 'c' ) {
37     fprintf ( stderr, "error:  image must be 24-bit color\n" );
38     exit ( 1 );
39   }
40
41   /* set up structure for output color image */
42   /* Note that the type is 'c' rather than 'g' */
43   get_TIFF ( &color_img, input_img.height, input_img.width, 'c' );
44
45   /* Filter image along horizontal direction */
46   for ( k = 0; k < 3; k++)
47   for ( i = 0; i < input_img.height; i++ )
48   for ( j = 0; j < input_img.width ; j++ ) {
49     if( ( (i>=4) && (i < input_img.height-4) ) &&\
50       ( (j>=4) && (j < input_img.width-4 ) ) ){
51       h_mn=0.0;
52       for( m=-4; m<=4; m++)
53       for( n=-4; n<=4; n++){
54       h_mn += input_img.color[k][i+m][j+n];
55       }
```

```
56          h_mn /= 81.0;
57          color_img.color[k][i][j] = clip(h_mn);
58        }
59        else{
60          color_img.color[k][i][j] = 0;
61        }
62      }
63
64      /* open color image file */
65      if ( ( fp = fopen ( "color.tif", "wb" ) ) == NULL ) {
66          fprintf ( stderr, "cannot open file color.tif\n");
67          exit ( 1 );
68      }
69
70      /* write color image */
71      if ( write_TIFF ( fp, &color_img ) ) {
72          fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
73          exit ( 1 );
74      }
75
76      /* close color image file */
77      fclose ( fp );
78
79      /* de-allocate space which was used for the images */
80      free_TIFF ( &(input_img) );
81      free_TIFF ( &(color_img) );
82
83      return(0);
84  }
85
86  int32_t clip(double pixel_double){
87      int32_t p;
88      p = ( int32_t ) pixel_double;
89      if(pixel_double>255.0){return 255;}
90      else{
91        if(pixel_double<0.0){return 0;}
92        return p;
93      }
94  }
95
96  void error(char *name)
97  {
98      printf("usage:  %s  image.tiff \n\n",name);
99      printf("this program reads in a 24-bit color TIFF image.\n");
100     printf("It then horizontally filters the green component, adds noise,\n");
101     printf("and writes out the result as an 8-bit image\n");
102     printf("with the name 'green.tiff'.\n");
103     printf("It also generates an 8-bit color image,\n");
104     printf("that swaps red and green components from the input image");
105     exit(1);
106 }
```

Listing 2: FIR sharpening

```
1
2  #include <math.h>
3  #include <stdlib.h>
4  #include "tiff.h"
5  #include "allocate.h"
```

```
6  #include "randlib.h"
7  #include "typeutil.h"
8
9  void error(char *name);
10 int32_t clip(double pixel);
11
12 //#define LAM 1.5
13
14 int main (int argc, char **argv)
15 {
16   FILE *fp;
17   struct TIFF_img input_img, color_img;
18   double h_mn, g_mn, d_mn, p, LAM;
19   int32_t i,j,k,m,n;
20
21   if ( argc != 3 ) error( argv[0] );
22
23   /* open image file */
24   if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
25     fprintf ( stderr, "cannot open file %s\n", argv[1] );
26     exit ( 1 );
27   }
28
29   /* read image */
30   if ( read_TIFF ( fp, &input_img ) ) {
31     fprintf ( stderr, "error reading file %s\n", argv[1] );
32     exit ( 1 );
33   }
34
35   /* close image file */
36   fclose ( fp );
37
38   /* check the type of image data */
39   if ( input_img.TIFF_type != 'c' ) {
40     fprintf ( stderr, "error:  image must be 24-bit color\n" );
41     exit ( 1 );
42   }
43
44   //convert lambda to double
45  LAM = atof(argv[2]);
46
47   /* set up structure for output color image */
48   /* Note that the type is 'c' rather than 'g' */
49   get_TIFF ( &color_img, input_img.height, input_img.width, 'c' );
50
51   /* Filter image along horizontal direction */
52   for ( k = 0; k < 3; k++)
53   for ( i = 0; i < input_img.height; i++ )
54   for ( j = 0; j < input_img.width ; j++ ) {
55     if( ( (i>=2) && (i < input_img.height-2) ) &&\
56       ( (j>=2) && (j < input_img.width-2 ) ) ){
57       p = 0.0;
58       h_mn=1/25.0;
59       for( m=-2; m<=2; m++)
60       for( n=-2; n<=2; n++){
61         //find delta
62         if((m==0) && (n==0)){d_mn=1.0;}
63         else{d_mn = 0.0;}
64         //calc g_mn and multiply with pixel val
```

```
65            g_mn = d_mn + LAM*(d_mn -h_mn);
66            p += g_mn*input_img.color[k][i+m][j+n];
67          }
68          color_img.color[k][i][j] = clip(p);
69        }
70        else{
71          color_img.color[k][i][j] = 0;
72        }
73      }
74
75      /* open color image file */
76      if ( ( fp = fopen ( "color.tif", "wb" ) ) == NULL ) {
77          fprintf ( stderr, "cannot open file color.tif\n");
78          exit ( 1 );
79      }
80
81      /* write color image */
82      if ( write_TIFF ( fp, &color_img ) ) {
83          fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
84          exit ( 1 );
85      }
86
87      /* close color image file */
88      fclose ( fp );
89
90      /* de-allocate space which was used for the images */
91      free_TIFF ( &(input_img) );
92      free_TIFF ( &(color_img) );
93
94      return(0);
95  }
96
97  int32_t clip(double pixel_double){
98      int32_t p;
99      p = ( int32_t ) pixel_double;
100     if(pixel_double>255.0){return 255;}
101     else{
102       if(pixel_double<0.0){return 0;}
103       return p;
104     }
105 }
106
107 void error(char *name)
108 {
109     printf("usage: %s image.tiff \n\n",name);
110     printf("this program reads in a 24-bit color TIFF image.\n");
111     printf("It then horizontally filters the green component, adds noise,\n");
112     printf("and writes out the result as an 8-bit image\n");
113     printf("with the name 'green.tiff'.\n");
114     printf("It also generates an 8-bit color image,\n");
115     printf("that swaps red and green components from the input image");
116     exit(1);
117 }
```

Listing 3: IIR

```
1
2 #include <math.h>
3 #include "tiff.h"
```

```
4 #include "allocate.h"
5 #include "randlib.h"
6 #include "typeutil.h"
7
8 void error(char *name);
9 int32_t clip(double pixel);
10
11 const double coeffA = 0.01;
12 const double coeffB = 0.9;
13 const double coeffC = -0.81;
14
15 int main (int argc, char **argv)
16 {
17   FILE *fp;
18   struct TIFF_img input_img, color_img;
19   double **img1;
20   int32_t i,j,k;
21
22   if ( argc != 2 ) error( argv[0] );
23
24   /* open image file */
25   if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
26     fprintf ( stderr, "cannot open file %s\n", argv[1] );
27     exit ( 1 );
28   }
29
30   /* read image */
31   if ( read_TIFF ( fp, &input_img ) ) {
32     fprintf ( stderr, "error reading file %s\n", argv[1] );
33     exit ( 1 );
34   }
35
36   /* close image file */
37   fclose ( fp );
38
39   /* check the type of image data */
40   if ( input_img.TIFF_type != 'c' ) {
41     fprintf ( stderr, "error:  image must be 24-bit color\n" );
42     exit ( 1 );
43   }
44   /* Allocate image of double precision floats */
45   img1 = (double **)get_img(input_img.width,input_img.height,sizeof(double));
46   //make img1 as zeros
47   for ( i = 0; i < input_img.height; i++ )
48   for ( j = 0; j < input_img.width ; j++ ){
49     img1[i][j]=0.0;
50   }
51
52   /* set up structure for output color image */
53   /* Note that the type is 'c' rather than 'g' */
54   get_TIFF ( &color_img, input_img.height, input_img.width, 'c' );
55
56   /* Filter image along horizontal direction */
57   for ( k = 0; k < 3; k++){
58   //compute img1 for kth channel
59   for ( i = 0; i < input_img.height; i++ )
60   for ( j = 0; j < input_img.width ; j++ ) {
61     if( ( i>0 ) &&\
62       ( j>0 ) ){
```

```
63        img1[i][j] = coeffA*input_img.color[k][i][j] +\
64                     coeffB*( img1[i-1][j] + img1[i][j-1] ) +\
65                     coeffC*img1[i-1][j-1];
66      }
67      else{
68        img1[i][j] = 0.0;
69      }
70    }
71    //copy data to output image
72    for ( i = 0; i < input_img.height; i++ )
73    for ( j = 0; j < input_img.width ; j++ ){
74      color_img.color[k][i][j] = clip(img1[i][j]);
75    }
76    }
77
78    /* open color image file */
79    if ( ( fp = fopen ( "color.tif", "wb" ) ) == NULL ) {
80        fprintf ( stderr, "cannot open file color.tif\n");
81        exit ( 1 );
82    }
83
84    /* write color image */
85    if ( write_TIFF ( fp, &color_img ) ) {
86        fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
87        exit ( 1 );
88    }
89
90    /* close color image file */
91    fclose ( fp );
92
93    /* de-allocate space which was used for the images */
94    free_TIFF ( &(input_img) );
95    free_TIFF ( &(color_img) );
96
97    free_img( (void**)img1 );
98
99    return(0);
100 }
101
102 int32_t clip(double pixel_double){
103   int32_t p;
104   p = ( int32_t ) pixel_double;
105   if(pixel_double>255.0){return 255;}
106   else{
107     if(pixel_double<0.0){return 0;}
108     return p;
109   }
110 }
111
112 void error(char *name)
113 {
114     printf("usage:  %s  image.tiff \n\n",name);
115     printf("this program reads in a 24-bit color TIFF image.\n");
116     printf("It then horizontally filters the green component, adds noise,\n");
117     printf("and writes out the result as an 8-bit image\n");
118     printf("with the name 'green.tiff'.\n");
119     printf("It also generates an 8-bit color image,\n");
120     printf("that swaps red and green components from the input image");
121     exit(1);
```

```
}
```

Appendix

Listing 4: Python code for frequency plots

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Jan 26 16:18:31 2021

@author: rahul
course: ECE637-DIP-I, Spring 2021
"""
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm

def FIR_LP1(u):
    a =1.0
    for i in range(4):
        a += 2.0*np.cos((i+1)*u)
    return a/9.0

def FIR_LP(u,v):
    return FIR_LP1(u)*FIR_LP1(v)

def FIR_1(u):
    a = 1.0 +2*(np.cos(u) + np.cos(2*u))
    return a/5.0

def FIR(u,v): return FIR_1(u)*FIR_1(v)

def FIR_unsharp(u,v,lam=1.5):
    return 1.0 + lam*(1-FIR(u,v))

def IIR(u,v):
    return np.abs(0.01/(0.9*(np.exp(-1j*u) + np.exp(-1j*v))
                        -0.81*np.exp(-1j*u)*np.exp(-1j*v)))
def grid(U,V,func):
    z = np.array(func(np.ravel(U),np.ravel(V)))
    Z = z.reshape(U.shape)
    return Z

def plot_fig(X,Y,Z,fig_name):
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    ax.plot_surface(X,Y,Z,cmap=cm.viridis)
    ax.set_xlim(-1*np.pi,1*np.pi)
    ax.set_ylim(-1*np.pi,1*np.pi)
    plt.xlabel('u')
    plt.ylabel('v')
    ax.set_zlabel('$|H(e^{ju},e^{jv})|$')
    ax.autoscale(enable=True,axis='z',tight=True)
    plt.savefig(fig_name+'.eps',format='eps')

    fig2, ax2 = plt.subplots(figsize=(6,6))
```

```
    cf = ax2.contourf(X,Y,Z)
    fig2.colorbar(cf, ax=ax2)
    ax2.set_xlim(-1*np.pi,1*np.pi)
    ax2.set_ylim(-1*np.pi,1*np.pi)
    plt.xlabel('u')
    plt.ylabel('v')
    plt.savefig(fig_name+'_contour.eps',format='eps')

def main():
    pi = np.pi
    step = 0.1
    u = np.arange(-1*pi,pi,step)
    v = np.arange(-1*pi,pi,step)
    U,V = np.meshgrid(u,v)

    Z1 = grid(U,V,FIR_LP)
    Z2 = grid(U,V,FIR)
    Z3 = grid(U,V,FIR_unsharp)
    Z4 = grid(U,V,IIR)

    plot_fig(U,V,Z1,'fig1')
    plot_fig(U,V,Z2,'fig2')
    plot_fig(U,V,Z3,'fig3')
    plot_fig(U,V,Z4,'fig4')

    return

if __name__=="__main__":
    main()
```

Listing 5: Python code for plotting psf for IIR filter

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Jan 27 20:41:15 2021

@author: rahul
course: ECE637-DIP-I
lab1- 5.2
"""

import numpy as np
import cv2

img = np.zeros((256,256))
img[127][127] = 255*100

cv2.imwrite('lab1_5.2.jpg',img) #original image

img2 = np.zeros((256,256))

for i in range(256):
    for j in range(256):
        if((i>0) and (j>0)):
            img2[i][j] = (0.01*img[i][j] + 0.9*(img2[i-1][j] + img2[i][j-1])
                          - 0.81*(img2[i-1][j-1]))

cv2.imwrite('lab1_5.2_IIR.jpg',img2)
```