

ECE637 Digital Image Processing I

Laboratory work 1: Image Filtering

Boris Diner

January 28, 2021

3 FIR Low Pass Filter

3.1 The analytical expression for $H(e^{j\mu}, e^{j\nu})$

Given a low pass filter $h(m, n)$ such that

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

By definition, the DSFT of $h(m, n)$ is

$$H(e^{j\mu}, e^{j\nu}) \triangleq \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} h(m, n) \cdot e^{-j(\mu m + \nu n)} \quad (2)$$

Plugging expression 1 in 2, we get

$$H(e^{j\mu}, e^{j\nu}) = \frac{1}{81} \cdot \sum_{m=-4}^4 e^{-j\mu m} \sum_{n=-4}^4 e^{-j\nu n} \quad (3)$$

In general, for any arbitrary natural number N a sum of complex exponentials can be rewritten as

$$\sum_{m=-N}^N e^{-j\mu m} = \sum_{m'=0}^{2N} e^{-j\mu(m'-N)} = e^{j\mu N} \cdot \sum_{m'=0}^{2N} e^{-j\mu m'} \quad (4)$$

The sum of the first $2N + 1$ terms of the geometric sequence of complex exponentials is equal to

$$\sum_{m'=0}^{2N} e^{-j\mu m'} = \frac{1 - e^{-j\mu(2N+1)}}{1 - e^{-j\mu}} \quad (5)$$

Combining 4 with 5 and performing some operations, we obtain the following result

$$e^{j\mu N} \cdot \frac{1 - e^{-j\mu(2N+1)}}{1 - e^{-j\mu}} = e^{j\mu N} \cdot \frac{e^{-j\mu(2N+1)/2}}{e^{-j\mu/2}} \cdot \frac{e^{j\mu(2N+1)/2} - e^{-j\mu(2N+1)/2}}{e^{j\mu/2} - e^{-j\mu/2}} \quad (6)$$

It can be derived from Euler's formula that

$$\sin x = \frac{e^{jx} - e^{-jx}}{2j} \quad (7)$$

Applying 7 to 6, we get

$$\sum_{m=-N}^N e^{-j\mu m} = \frac{\sin\left(\frac{(2N+1)\mu}{2}\right)}{\sin\left(\frac{\mu}{2}\right)} \quad (8)$$

Using the fact that we are interested in the case of $N = 4$ and generalizing the result for 2-D case, we find the analytical expression for $H(e^{j\mu}, e^{j\nu})$

$$H(e^{j\mu}, e^{j\nu}) = \frac{1}{81} \cdot \frac{\sin \frac{9\mu}{2} \cdot \sin \frac{9\nu}{2}}{\sin \frac{\mu}{2} \cdot \sin \frac{\nu}{2}} \quad (9)$$

Therefore, the magnitude of the frequency response is

$$|H(e^{j\mu}, e^{j\nu})| = \frac{1}{81} \cdot \left| \frac{\sin \frac{9\mu}{2} \cdot \sin \frac{9\nu}{2}}{\sin \frac{\mu}{2} \cdot \sin \frac{\nu}{2}} \right|$$

3.2 The plot of $|H(e^{j\mu}, e^{j\nu})|$

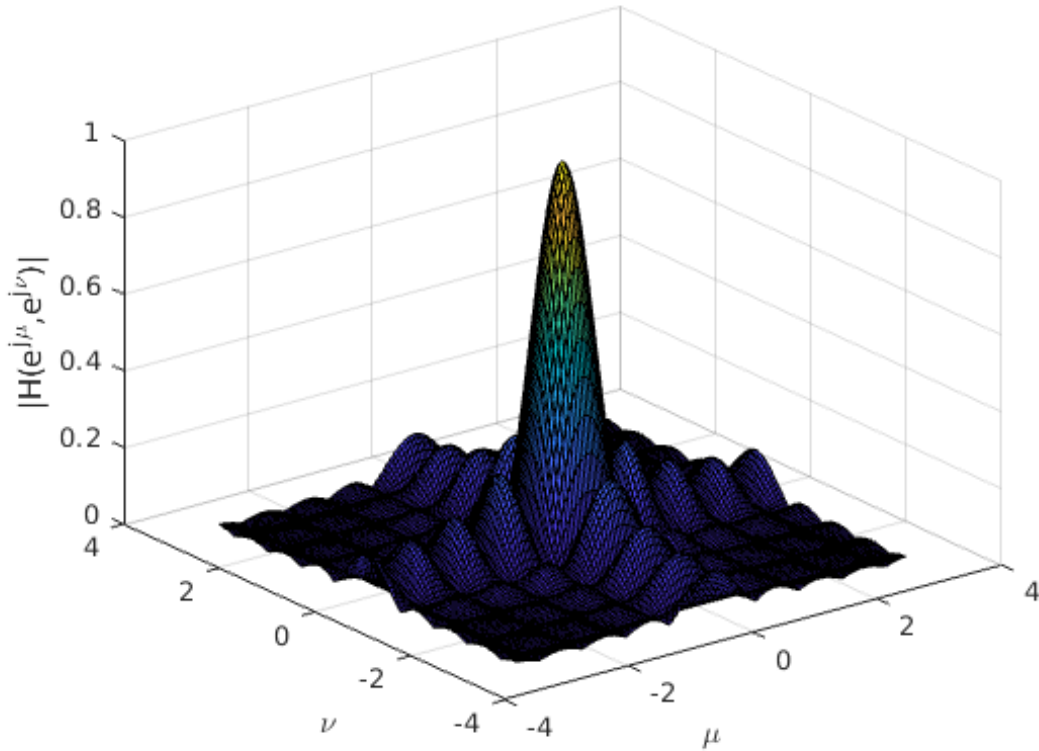


Figure 1: The magnitude of the frequency response $|H(e^{j\mu}, e^{j\nu})|$

3.3 The color image and the filtered image



Figure 2: The result of filtering the original image (a) with the low-pass filter $H(e^{j\mu}, e^{j\nu})$

3.4 C code listing for filtering the image

```
1 #include <math.h>
2 #include "tiff.h"
3 #include "allocate.h"
4 #include "randlib.h"
5 #include "typeutil.h"
6
7 #define FILTER_WIDTH 9
8 #define FILTER_HEIGHT 9
9
10 void error(char *name);
11 double crop_image(double pixel);
12
13 int main (int argc, char **argv)
```

```

14 {
15 FILE *fp;
16 struct TIFF_img input_img, color_img;
17 double **img_temp;
18 int i,j,k,l;
19 uint8_t channel;
20 double sum;
21 double h = 1.0 / (FILTER_WIDTH * FILTER_HEIGHT);
22 int half_filter_width = (FILTER_WIDTH - 1) / 2;
23 int half_filter_height = (FILTER_HEIGHT - 1) / 2;
24
25
26 if ( argc != 2 ) error( argv[0] );
27 /* open image file */
28 if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
29     fprintf ( stderr, "cannot open file %s\n", argv[1] );
30     exit ( 1 );
31 }
32
33 /* read image */
34 if ( read_TIFF ( fp, &input_img ) ) {
35     fprintf ( stderr, "error reading file %s\n", argv[1] );
36     ;exit ( 1 );
37 }
38
39 /* close image file */
40 fclose ( fp );
41
42 /* check the type of image data */
43 if ( input_img.TIFF_type != 'c' ) {
44     fprintf ( stderr, "error: image must be 24-bit color\n"
45         " );
46     exit ( 1 );
47 }

```

```

47  get_TIFF ( &color_img, input_img.height, input_img.width
    , 'c' );
48
49  /* Filter each of three color channels of the image */
50  for (channel = 0; channel < 3; channel++){
51
52      img_temp = (double**)get_img(input_img.width + (
        FILTER_WIDTH - 1), input_img.height + (FILTER_HEIGHT -
        1),sizeof(double)); // Allocate image of double
        precision floats
53
54      /* Free boundaries */
55      for(i = 0; i < input_img.height + (FILTER_HEIGHT - 1);
        i++){
56          for(j = 0; j < input_img.width + (FILTER_WIDTH - 1);
            j++){
57              if ((i >= half_filter_height && i < (input_img.
                height + half_filter_height)) && (j >= half_filter_width
                    && ( j < input_img.width + half_filter_width))){
58                  img_temp[i][j] = input_img.color[channel][i-
                    half_filter_height][j-half_filter_width];
59              }
60              else {
61                  img_temp[i][j] = 0;
62              }
63          }
64      }
65      /* Filtering process */
66      for ( i = 0; i < input_img.height; i++ ){
67          for ( j = 0; j < input_img.width; j++ ){
68              sum = 0.0;
69              for (k = 0; k < FILTER_HEIGHT; k++ ){
70                  for (l = 0; l < FILTER_WIDTH; l++ ){
71                      sum += h * img_temp[i+k][j+l];
72                  }

```

```

73     }
74     color_img.color[channel][i][j] = crop_image(sum);
75 }
76 }
77
78     free_img((void**)img_temp); // Clear allocated memory
    after each iteration
79
80 }
81
82 /* open color image file */
83 if ( ( fp = fopen ( "img-lowpass.tif", "wb" ) ) == NULL
    ) {
84     fprintf ( stderr, "cannot open file color.tif\n");
85     exit ( 1 );
86 }
87
88 /* write color image */
89 if ( write_TIFF ( fp, &color_img ) ) {
90     fprintf ( stderr, "error writing TIFF file %s\n", argv
    [2] );
91     exit ( 1 );
92 }
93
94 /* close color image file */
95 fclose ( fp );
96
97 /* de-allocate space which was used for the images */
98 free_TIFF ( &(input_img) );
99 free_TIFF ( &(color_img) );
100
101 return(0);
102 }
103
104 void error(char *name)

```

```

105 {
106     printf("usage: %s image.tiff \n\n",name);
107     printf("this program reads in a 24-bit color TIFF image
        .\n");
108     printf("It then horizontally filters the green component
        , adds noise,\n");
109     printf("and writes out the result as an 8-bit image\n");
110     printf("with the name 'green.tiff'.\n");
111     printf("It also generates an 8-bit color image,\n");
112     printf("that swaps red and green components from the
        input image");
113     exit(1);
114 }
115 double crop_image(double pixel)
116 {
117     if (pixel > 255) return 255;
118     if (pixel < 0) return 0;
119     return pixel;
120 }

```

4 FIR Sharpening Filter

4.1 The analytical expression for $H(e^{j\mu}, e^{j\nu})$

Using the method described in [Section 3](#) for $N = 2$ and for the normalization factor of $\frac{1}{25}$, we get

$$H(e^{j\mu}, e^{j\nu}) = \frac{1}{25} \cdot \frac{\sin \frac{5\mu}{2} \cdot \sin \frac{5\nu}{2}}{\sin \frac{\mu}{2} \cdot \sin \frac{\nu}{2}} \quad (10)$$

4.2 The analytical expression for $G(e^{j\mu}, e^{j\nu})$

Taking into account the fact that

DSFT $\{\delta(m, n)\} = \text{DTFT}\{\delta(m)\} \cdot \text{DTFT}\{\delta(n)\} = 1$
and using the linearity property of the DTFT, we get

$$G(e^{j\mu}, e^{j\nu}) = 1 + \lambda[1 - H(e^{j\mu}, e^{j\nu})] \quad (11)$$

Plugging 10 in 11, we derive the analytical expression for the sharpening filter $G(e^{j\mu}, e^{j\nu})$

$$G(e^{j\mu}, e^{j\nu}) = 1 + \lambda \left(1 - \frac{1}{25} \cdot \frac{\sin \frac{5\mu}{2} \cdot \sin \frac{5\nu}{2}}{\sin \frac{\mu}{2} \cdot \sin \frac{\nu}{2}} \right)$$

As $\text{Im}\{G\} = 0$, the magnitude of the frequency response is

$$|G(e^{j\mu}, e^{j\nu})| = \left| 1 + \lambda \left(1 - \frac{1}{25} \cdot \frac{\sin \frac{5\mu}{2} \cdot \sin \frac{5\nu}{2}}{\sin \frac{\mu}{2} \cdot \sin \frac{\nu}{2}} \right) \right|$$

4.3 The plot of $|H(e^{j\mu}, e^{j\nu})|$

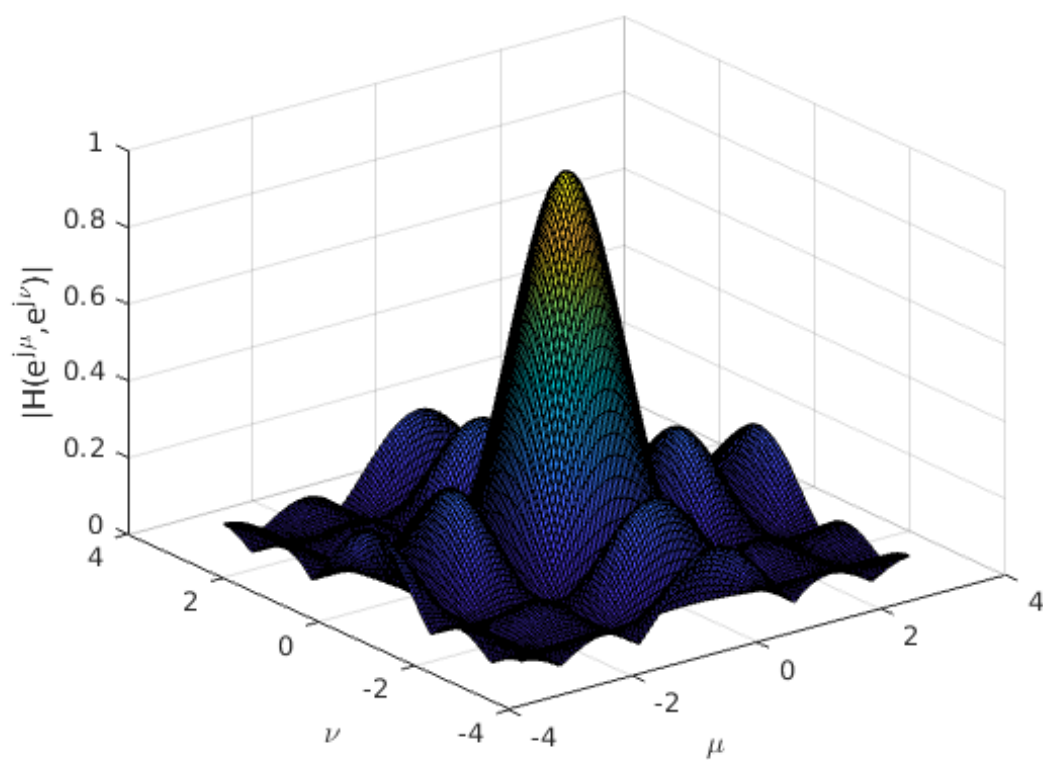


Figure 3: The magnitude of the frequency response $|H(e^{j\mu}, e^{j\nu})|$

4.4 The plot of $|G(e^{j\mu}, e^{j\nu})|$ for $\lambda = 1.5$

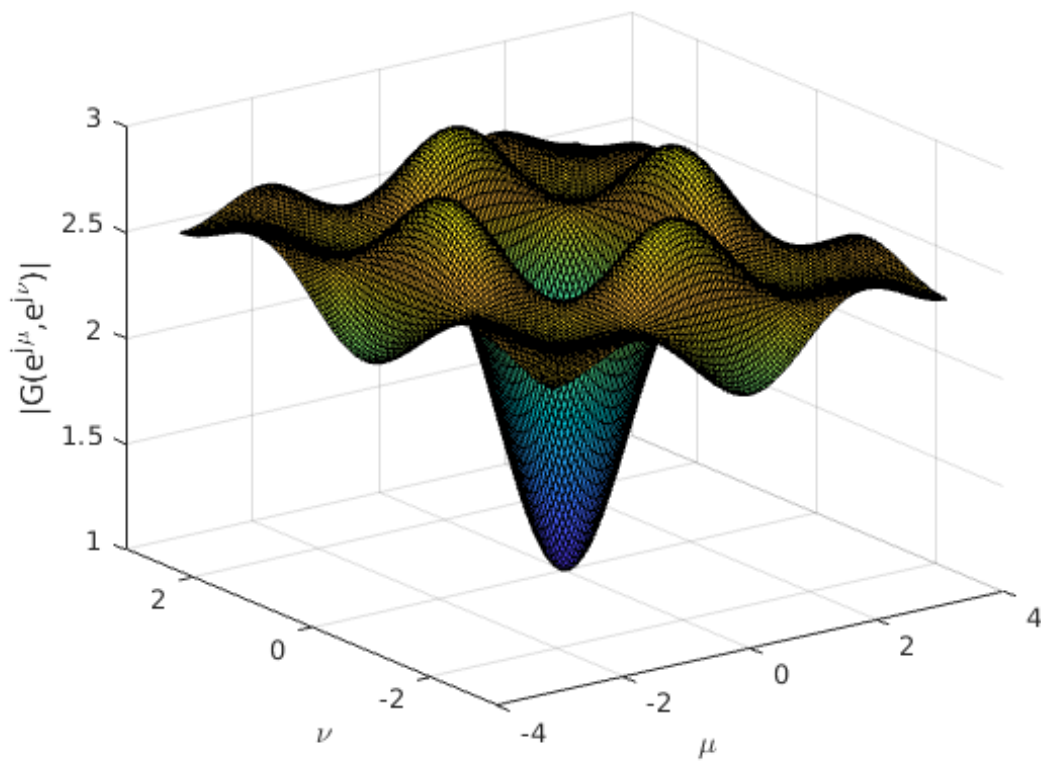


Figure 4: The magnitude of the frequency response $|G(e^{j\mu}, e^{j\nu})|$ for $\lambda = 1.5$

4.5 The color image and the sharpened image

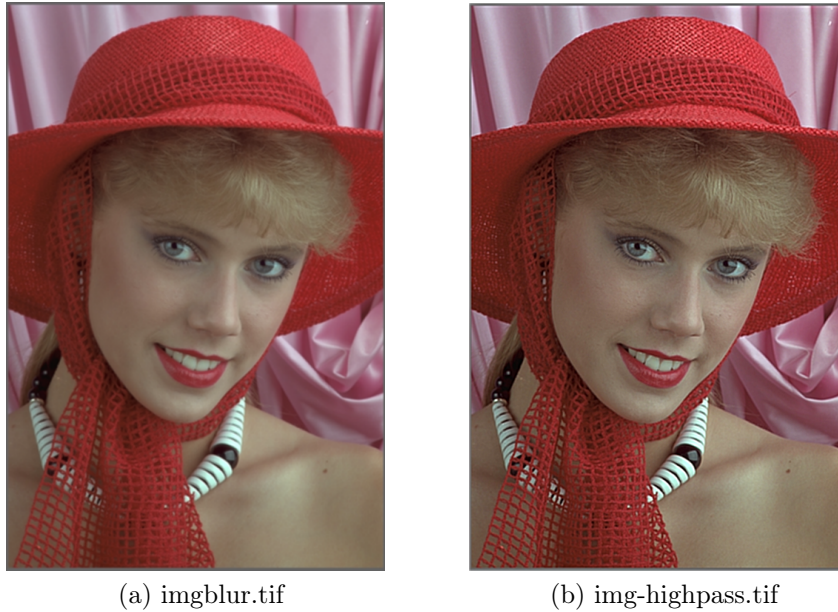


Figure 5: The result of sharpening the image (a) with the high-pass filter $G(e^{j\mu}, e^{j\nu})$

4.6 C code listing for filtering the image

```
1 #include <math.h>
2 #include "tiff.h"
3 #include "allocate.h"
4 #include "randlib.h"
5 #include "typeutil.h"
6
7 #define FILTER_WIDTH 5
8 #define FILTER_HEIGHT 5
9
10 void error(char *name);
11 double crop_image(double pixel);
12
13 int main (int argc, char **argv)
```

```

14 {
15 FILE *fp;
16 struct TIFF_img input_img, color_img;
17 double **img_temp;
18 int i,j,k,l;
19 uint8_t channel;
20 double sum;
21 double h = 1.0 / (FILTER_WIDTH * FILTER_HEIGHT);
22 double lambda = 1.5;
23 double delta;
24 double sharpening_filter[FILTER_WIDTH][FILTER_HEIGHT];
25 int half_filter_width = (FILTER_WIDTH - 1) / 2;
26 int half_filter_height = (FILTER_HEIGHT - 1) / 2;
27
28
29 if ( argc != 2 ) error( argv[0] );
30 /* open image file */
31 if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
32     fprintf ( stderr, "cannot open file %s\n", argv[1] );
33     exit ( 1 );
34 }
35
36 /* read image */
37 if ( read_TIFF ( fp, &input_img ) ) {
38     fprintf ( stderr, "error reading file %s\n", argv[1] );
39     ;exit ( 1 );
40 }
41
42 /* close image file */
43 fclose ( fp );
44
45 /* check the type of image data */
46 if ( input_img.TIFF_type != 'c' ) {
47     fprintf ( stderr, "error: image must be 24-bit color\n"
48 " );

```

```

47     exit ( 1 );
48 }
49
50 get_TIFF ( &color_img, input_img.height, input_img.width
51           , 'c' );
52
53 /* Fill in the sharpening filter */
54 for (i = 0; i < FILTER_WIDTH; i++) {
55     for(j = 0; j < FILTER_HEIGHT; j++) {
56         if(i == half_filter_width && j ==
57            half_filter_height) {
58             delta = 1.0; // the center of the filter
59         }
60         else {
61             delta = 0.0;
62         }
63         sharpening_filter[i][j] = delta + lambda * (delta - h)
64     ;
65 }
66 }
67
68 /* Filter each of three color channels of the image */
69 for (channel = 0; channel < 3; channel++){
70
71     img_temp = (double**)get_img(input_img.width + (
72     FILTER_WIDTH - 1), input_img.height + (FILTER_HEIGHT -
73     1),sizeof(double)); // Allocate image of double
74     precision floats
75
76     /* Free boundaries */
77     for(i = 0; i < input_img.height + (FILTER_HEIGHT - 1);
78        i++){
79         for(j = 0; j < input_img.width + (FILTER_WIDTH - 1);
80            j++){
81             if ((i >= half_filter_height && i < (input_img.

```

```

height + half_filter_height)) && (j >= half_filter_width
&& ( j < input_img.width + half_filter_width))) {
74     img_temp[i][j] = input_img.color[channel][i-
half_filter_height][j-half_filter_width];
75 }
76 else {
77     img_temp[i][j] = 0;
78 }
79 }
80 }
81 /* Filtering process */
82 for ( i = 0; i < input_img.height; i++ ){
83     for ( j = 0; j < input_img.width; j++ ){
84         sum = 0.0;
85         for (k = 0; k < FILTER_HEIGHT; k++ ){
86             for (l = 0; l < FILTER_WIDTH; l++ ){
87                 sum += sharpening_filter[k][l] * img_temp[i+k
][j+l];
88             }
89         }
90         color_img.color[channel][i][j] = crop_image(sum);
91     }
92 }
93
94 free_img((void**)img_temp); // Clear allocated memory
after each iteration
95
96 }
97
98 /* open color image file */
99 if ( ( fp = fopen ( "img-highpass.tif", "wb" ) ) == NULL
) {
100     fprintf ( stderr, "cannot open file color.tif\n");
101     exit ( 1 );
102 }

```

```

103
104  /* write color image */
105  if ( write_TIFF ( fp, &color_img ) ) {
106      fprintf ( stderr, "error writing TIFF file %s\n", argv
107               [2] );
108      exit ( 1 );
109  }
110
111  /* close color image file */
112  fclose ( fp );
113
114  /* de-allocate space which was used for the images */
115  free_TIFF ( &(input_img) );
116  free_TIFF ( &(color_img) );
117
118  return(0);
119 }
120
121 void error(char *name)
122 {
123     printf("usage: %s image.tiff \n\n",name);
124     printf("this program reads in a 24-bit color TIFF image
125            .\n");
126     printf("It then horizontally filters the green component
127            , adds noise,\n");
128     printf("and writes out the result as an 8-bit image\n");
129     printf("with the name 'green.tiff'.\n");
130     printf("It also generates an 8-bit color image,\n");
131     printf("that swaps red and green components from the
132            input image");
133     exit(1);
134 }
135
136 double crop_image(double pixel)
137 {
138     if (pixel > 255) return 255;

```



```

134  if (pixel < 0)  return 0;
135  return pixel;
136  }

```

5 IIR Filter

5.1 The analytical expression for $H(e^{j\mu}, e^{j\nu})$

Given the 2-D difference equation

$$\begin{aligned}
 y(m, n) = & 0.01x(m, n) + 0.9y(m-1, n) \\
 & + 0.9y(m, n-1) - 0.81y(m-1, n-1)
 \end{aligned} \tag{12}$$

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

Let $h(m, n)$ be the impulse response of an IIR filter with the corresponding difference equation.

Applying Z-transform to this expression, we get

$$\begin{aligned}
 Y(z_1, z_2) = & 0.01X(z_1, z_2) + 0.9z_1^{-1}Y(z_1, z_2) \\
 & + 0.9z_2^{-1}Y(z_1, z_2) - 0.81z_1^{-1}z_2^{-1}Y(z_1, z_2)
 \end{aligned}$$

Then the DSFT of the impulse response $h(m, n)$ is equal to

$$\begin{aligned}
 H(z_1, z_2) \Big|_{\substack{z_1=\exp(j\mu) \\ z_2=\exp(j\nu)}} &= \frac{Y(z_1, z_2)}{X(z_1, z_2)} \Big|_{\substack{z_1=\exp(j\mu) \\ z_2=\exp(j\nu)}} = \\
 &= \frac{0.01}{1 - 0.9e^{-j\mu} - 0.9e^{-j\nu} + 0.81e^{-j\mu}e^{-j\nu}}
 \end{aligned}$$

5.2 The plot of $|H(e^{j\mu}, e^{j\nu})|$

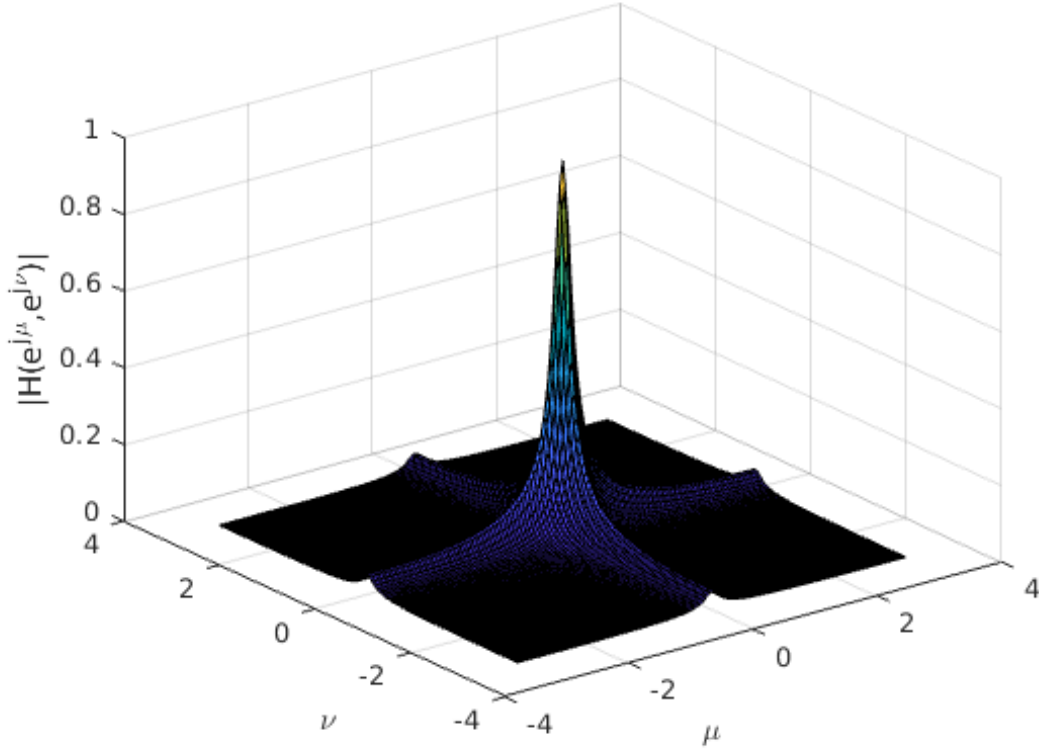


Figure 6: The magnitude of the frequency response $|H(e^{j\mu}, e^{j\nu})|$

5.3 The image of the point spread function

Applying the difference equation to a 256×256 image of the form $x(m, n) = \delta(m - 127, n - 127)$, we get the following result.

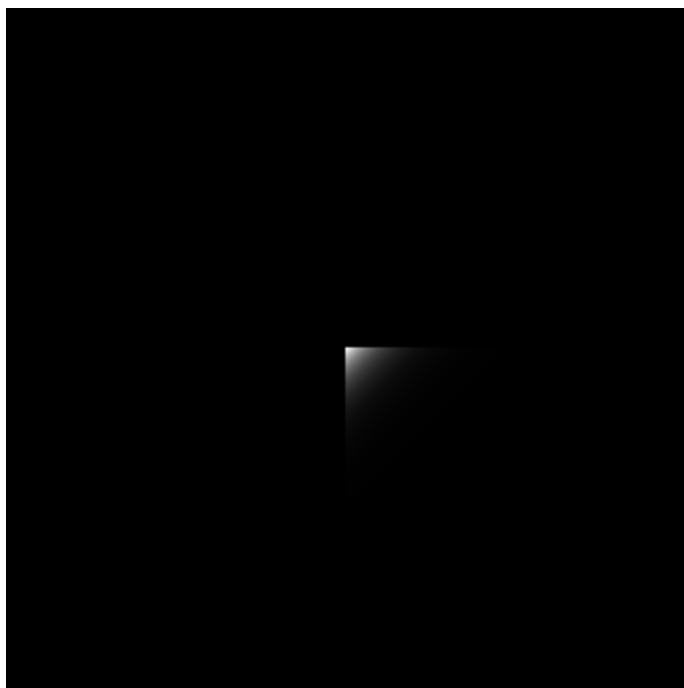


Figure 7: The image of the point spread function

5.4 The filtered output color image



Figure 8: The filtered output image img-iir.tif

5.5 C code listing for filtering the image

```
1 #include <math.h>
2 #include "tiff.h"
3 #include "allocate.h"
4 #include "randlib.h"
5 #include "typeutil.h"
6
7 void error(char *name);
8 double crop_image(double pixel);
9
10 int main (int argc, char **argv)
11 {
12     FILE *fp;
13     struct TIFF_img input_img, color_img;
14     double **img_temp;
15     int i,j,k,l;
16     uint8_t channel;
17
18     if ( argc != 2 ) error( argv[0] );
19     /* open image file */
20     if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
21         fprintf ( stderr, "cannot open file %s\n", argv[1] );
22         exit ( 1 );
23     }
24
25     /* read image */
26     if ( read_TIFF ( fp, &input_img ) ) {
27         fprintf ( stderr, "error reading file %s\n", argv[1] );
28         ;exit ( 1 );
29     }
30
31     /* close image file */
32     fclose ( fp );
```

```

33  /* check the type of image data */
34  if ( input_img.TIFF_type != 'c' ) {
35      fprintf ( stderr, "error: image must be 24-bit color\n
36      " );
37      exit ( 1 );
38  }
39  get_TIFF ( &color_img, input_img.height, input_img.width
40      , 'c' );
41  /* Filter each of three color channels of the image */
42  for (channel = 0; channel < 3; channel++){
43
44      img_temp = (double**)get_img(input_img.width + 1,
45      input_img.height + 1,sizeof(double)); // Allocate image
46      of double precision floats
47
48      for(i = 0;i < input_img.height + 1; i++){
49          for(j = 0;j < input_img.width + 1; j++){
50              img_temp[i][j] = 0;
51          }
52      }
53      for (k = 0; k < input_img.height; k++){
54          for (l = 0; l < input_img.width; l++){
55              img_temp[k+1][l+1] = 0.01*input_img.color[channel
56              ][k][l]+0.9*img_temp[k][l+1]+0.9*img_temp[k+1][l]-0.81*
57              img_temp[k][l];
58              color_img.color[channel][k][l] = crop_image(
59              img_temp[k+1][l+1]);
60          }
61      }
62      free_img( (void**)img_temp); // Clear allocated memory
63      after each iteration
64  }

```

```

60  /* open color image file */
61  if ( ( fp = fopen ( "img-iir.tif", "wb" ) ) == NULL ) {
62      fprintf ( stderr, "cannot open file color.tif\n");
63      exit ( 1 );
64  }
65
66  /* write color image */
67  if ( write_TIFF ( fp, &color_img ) ) {
68      fprintf ( stderr, "error writing TIFF file %s\n", argv
69      [2] );
70      exit ( 1 );
71  }
72
73  /* close color image file */
74  fclose ( fp );
75
76  /* de-allocate space which was used for the images */
77  free_TIFF ( &(input_img) );
78  free_TIFF ( &(color_img) );
79
80  return(0);
81 }
82
83 void error(char *name)
84 {
85     printf("usage: %s image.tiff \n\n",name);
86     printf("this program reads in a 24-bit color TIFF image
87     .\n");
88     printf("It then horizontally filters the green component
89     , adds noise,\n");
90     printf("and writes out the result as an 8-bit image\n");
91     printf("with the name 'green.tiff'.\n");
92     printf("It also generates an 8-bit color image,\n");
93     printf("that swaps red and green components from the
94     input image");

```

```
91     exit(1);
92 }
93
94 double crop_image(double pixel)
95 {
96     if (pixel > 255) return 255;
97     if (pixel < 0)  return 0;
98     return pixel;
99 }
```