1 FIR Low Pass Filter

Given,

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

1.1 DSFT of $h(m,n)$: $H(e^{j\mu}, e^{j\nu})$

We know that,

$$F(e^{j\mu}, e^{j\nu}) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} f(m,n) e^{-j(\mu m + \nu n)} \tag{1}$$

Therefore,

$$\begin{aligned} H(e^{j\mu}, e^{j\nu}) &= \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} h(m,n) e^{-j(\mu m + \nu n)} && \text{from (1)} \\ &= \sum_{n=-4}^{4} \sum_{m=-4}^{4} \frac{1}{81} e^{-j(\mu m + \nu n)} \\ &= \frac{1}{81} \sum_{n=-4}^{4} e^{-j\nu n} \sum_{m=-4}^{4} e^{-j\mu m} \end{aligned}$$

The sum of the geometric progression is given by,

$$\sum_{k=0}^{n} ar^k = a\Big(\frac{1-r^{n+1}}{1-r}\Big) \tag{2}$$

Hence,

$$\begin{aligned} \sum_{n=-N}^{N} e^{-j\nu n} &= \sum_{n=0}^{2N} e^{-j\nu(n-N)} \\ &= e^{j\nu N} \sum_{n=0}^{2N} e^{-j\nu n} \\ &= e^{j\nu N} \Big(\frac{1-(e^{-j\nu})^{2N+1}}{1-e^{-j\nu}}\Big) && \text{from (2)} \\ &= e^{j\nu N} \Big(\frac{1-e^{-j\nu(2N+1)}}{1-e^{-j\nu}}\Big) \end{aligned}$$

Therefore, the DSFT of $h(m,n)$ becomes,

$$\begin{aligned} H(e^{j\mu}, e^{j\nu}) &= \frac{1}{81} \sum_{n=-4}^{4} e^{-j\nu n} \sum_{m=-4}^{4} e^{-j\mu m} \\ &= \frac{1}{81} e^{j\nu 4} \Big(\frac{1-e^{-j\nu 9}}{1-e^{-j\nu}}\Big) e^{j\mu 4} \Big(\frac{1-e^{-j\mu 9}}{1-e^{-j\mu}}\Big) \\ &= \frac{1}{81} \frac{e^{j\nu 4} e^{-j\nu\frac{9}{2}}}{e^{-j\frac{\nu}{2}}} \Big(\frac{e^{j\nu\frac{9}{2}} - e^{-j\nu\frac{9}{2}}}{e^{j\frac{\nu}{2}} - e^{-j\frac{\nu}{2}}}\Big) \frac{e^{j\mu 4} e^{-j\mu\frac{9}{2}}}{e^{-j\frac{\mu}{2}}} \Big(\frac{e^{j\mu\frac{9}{2}} - e^{-j\mu\frac{9}{2}}}{e^{j\frac{\mu}{2}} - e^{-j\frac{\mu}{2}}}\Big) \\ &= \frac{1}{81} \Big(\frac{e^{-j\nu\frac{9}{2}} - e^{-j\nu\frac{9}{2}}}{e^{j\frac{\nu}{2}} - e^{-j\frac{\nu}{2}}}\Big) \Big(\frac{e^{-j\mu\frac{9}{2}} - e^{-j\mu\frac{9}{2}}}{e^{j\frac{\mu}{2}} - e^{-j\frac{\mu}{2}}}\Big) \end{aligned}$$

We know,

$$\begin{aligned} e^{j\nu N} &= \cos(\nu N) + j\sin(\nu N) \\ \implies e^{j\nu N} - e^{-j\nu N} &= j2\sin(\nu N) \end{aligned} \tag{3}$$

Hence,

$$H(e^{j\mu}, e^{j\nu}) = \frac{1}{81}\frac{2j\sin(\nu\frac{9}{2})}{2j\sin(\nu\frac{1}{2})}\frac{2j\sin(\mu\frac{9}{2})}{2j\sin(\mu\frac{1}{2})} \qquad \text{from (3)}$$
$$= \frac{1}{81}\frac{\sin(\nu\frac{9}{2})}{\sin(\nu\frac{1}{2})}\frac{\sin(\mu\frac{9}{2})}{\sin(\mu\frac{1}{2})}$$

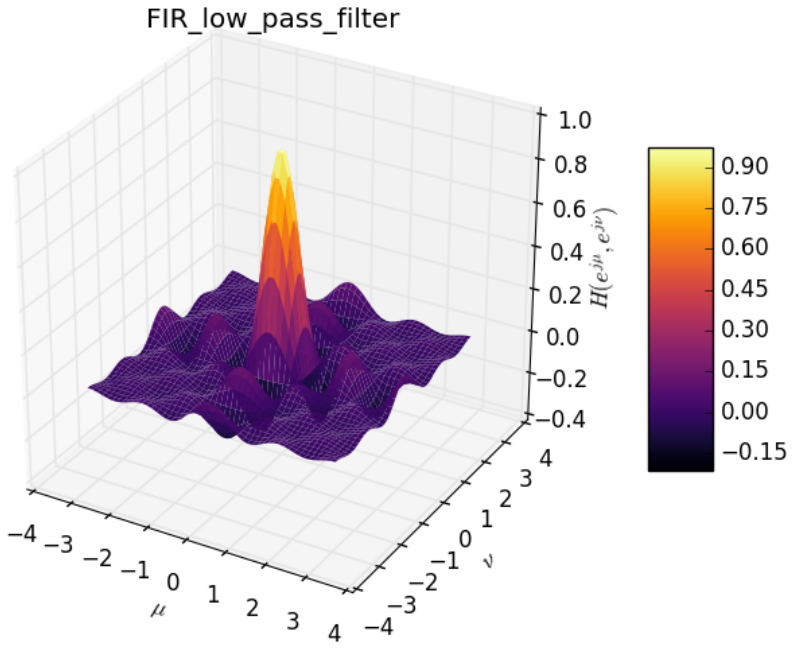


Figure 1: Magnitude of the frequency response $H(e^{j\mu}, e^{j\nu})$ of FIR Low Pass Filter

1.2 C-code

(a) green.tif (b) color.tif

Figure 2: Output of running C-code

1.3 Filtered Image

(a) Original (b) Filtered

Figure 3: Output of running FIR low pass filter on img03.tif

2 FIR Sharpening Filter

Given,

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

2.1 DSFT of $h(m,n)$: $H(e^{j\mu}, e^{j\nu})$

$$\begin{aligned}
H(e^{j\mu}, e^{j\nu}) &= \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} h(m,n) e^{-j(\mu m + \nu n)} && \text{from (1)} \\
&= \sum_{n=-2}^{2} \sum_{m=-2}^{2} \frac{1}{25} e^{-j(\mu m + \nu n)} \\
&= \frac{1}{25} \sum_{n=-2}^{2} e^{-j\nu n} \sum_{m=-2}^{2} e^{-j\mu m} \\
&= \frac{1}{25} e^{j\nu 2} \Big(\frac{1 - e^{-j\nu 5}}{1 - e^{-j\nu}} \Big) e^{j\mu 2} \Big(\frac{1 - e^{-j\mu 5}}{1 - e^{-j\mu}} \Big) \\
&= \frac{1}{25} \frac{e^{j\nu 2} e^{-j\nu \frac{5}{2}}}{e^{-j\frac{\nu}{2}}} \Big(\frac{e^{j\nu\frac{5}{2}} - e^{-j\nu\frac{5}{2}}}{e^{j\frac{\nu}{2}} - e^{-j\frac{\nu}{2}}} \Big) \frac{e^{j\mu 2} e^{-j\mu\frac{5}{2}}}{e^{-j\frac{\mu}{2}}} \Big(\frac{e^{j\mu\frac{5}{2}} - e^{-j\mu\frac{5}{2}}}{e^{j\frac{\mu}{2}} - e^{-j\frac{\mu}{2}}} \Big) \\
&= \frac{1}{25} \Big(\frac{e^{-j\nu\frac{5}{2}} - e^{-j\nu\frac{5}{2}}}{e^{j\frac{\nu}{2}} - e^{-j\frac{\nu}{2}}} \Big) \Big(\frac{e^{-j\mu\frac{5}{2}} - e^{-j\mu\frac{5}{2}}}{e^{j\frac{\mu}{2}} - e^{-j\frac{\mu}{2}}} \Big) \\
&= \frac{1}{25} \frac{2j\sin(\nu\frac{5}{2})}{2j\sin(\nu\frac{1}{2})} \frac{2j\sin(\mu\frac{5}{2})}{2j\sin(\mu\frac{1}{2})} && \text{from (3)} \\
&= \frac{1}{25} \frac{\sin(\nu\frac{5}{2})}{\sin(\nu\frac{1}{2})} \frac{\sin(\mu\frac{5}{2})}{\sin(\mu\frac{1}{2})}
\end{aligned}$$

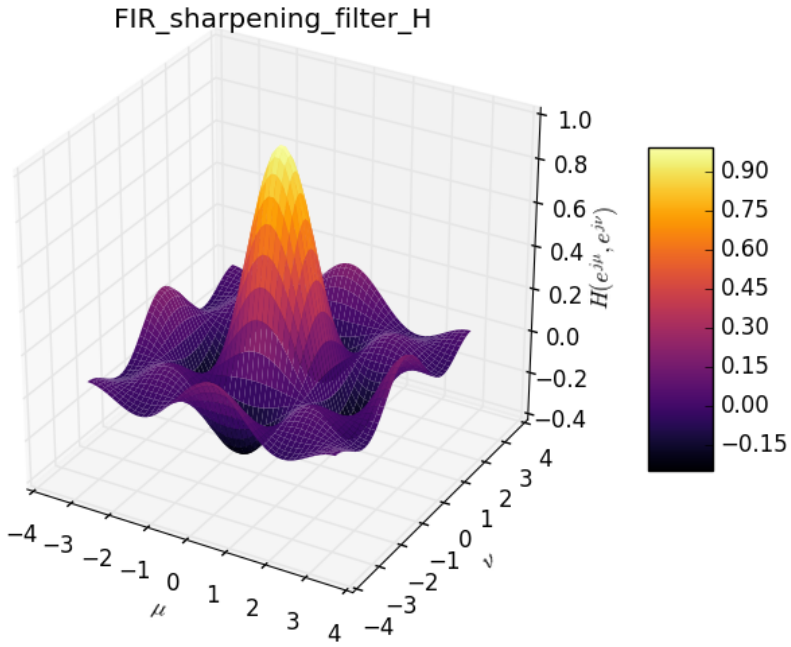


Figure 4: Magnitude of the frequency response $H(e^{j\mu}, e^{j\nu})$ of FIR Sharpening Filter

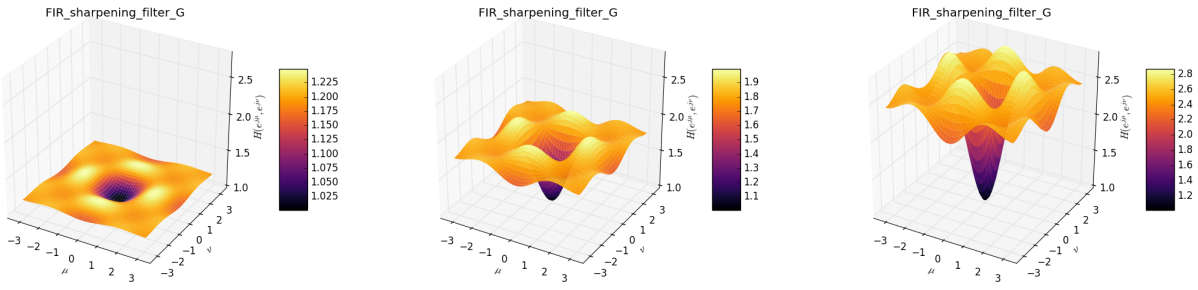
2.2 DSFT of $g(m,n)$: $G(e^{j\mu}, e^{j\nu})$

Given,

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

Taking DSFT gives,

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



(a) $\lambda = 0.2$ (b) $\lambda = 0.8$ (c) $\lambda = 1.5$

Figure 5: Comparison of Magnitude of the frequency response $G(e^{j\mu}, e^{j\nu})$ of FIR Sharpening Filter for different values of λ

2.3 Filtered Image

(a) Original

(b) Filtered $\lambda = 0.2$

(c) Filtered $\lambda = 0.8$

(d) Filtered $\lambda = 1.5$

Figure 6: Output of running FIR sharpening filter on imgblur.tif

3 IIR Filter

Given,

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

3.1 DSFT of $h(m,n)$: $H(e^{j\mu}, e^{j\nu})$

Replacing $x(m,n) = \delta(m,n)$ gives us $h(m,n)$

$$h(m,n) = 0.01\delta(m,n) + 0.9(h(m-1,n) + h(m,n-1)) - 0.81h(m-1,n-1)$$

Taking DSFT of the above equation gives us

$$\begin{aligned}
& H(e^{j\mu}, e^{j\nu}) = 0.01 + 0.9(e^{-j\mu}H(e^{j\mu}, e^{j\nu}) + e^{-j\nu}H(e^{j\mu}, e^{j\nu})) - 0.81e^{-j\mu}e^{-j\nu}H(e^{j\mu}, e^{j\nu}) \\
\implies & H(e^{j\mu}, e^{j\nu}) = 0.01 + 0.9H(e^{j\mu}, e^{j\nu})(e^{-j\mu} + e^{-j\nu}) - 0.81e^{-j\mu}e^{-j\nu}H(e^{j\mu}, e^{j\nu}) \\
\implies & H(e^{j\mu}, e^{j\nu})(1 - 0.9(e^{-j\mu} + e^{-j\nu}) + 0.81e^{-j\mu}e^{-j\nu}) = 0.01 \\
\implies & H(e^{j\mu}, e^{j\nu}) = \frac{0.01}{(1 - 0.9(e^{-j\mu} + e^{-j\nu}) + 0.81e^{-j\mu}e^{-j\nu})}
\end{aligned}$$

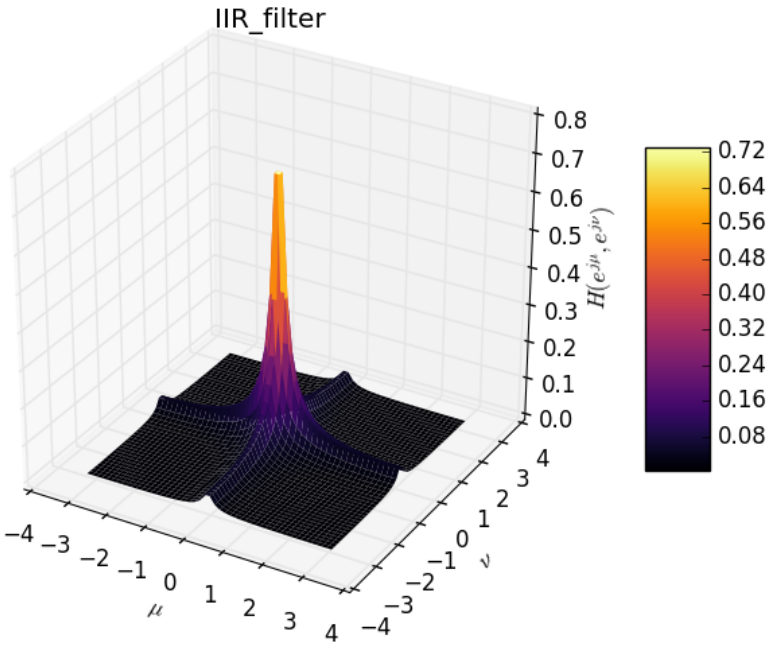


Figure 7: Magnitude of the frequency response $H(e^{j\mu}, e^{j\nu})$ of IIR Filter

3.2 Point Spread

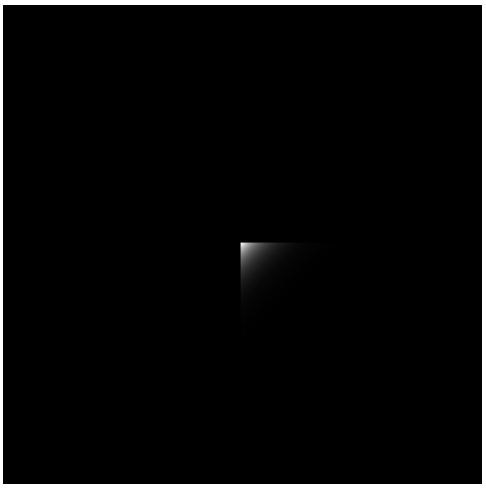


Figure 8: Point spread function of the IIR Filter

3.3 Filtered Image

(a) Original (b) Filtered

Figure 9: Output of running IIR filter on img03.tif

4 Code snippets

4.1 filters.c

```
#include <stdlib.h>
#include <string.h>

#include "allocate.h"
#include "tiff.h"


double **get_Delta(int32_t wd, int32_t ht)
{
  // Initialize delta
  double **delta = (double **)get_img(wd, ht ,sizeof(double));

  for(int i=0; i<ht; i++)
    for(int j=0; j<wd; j++)
      if(i==ht/2 && j==wd/2)
        delta[i][j] = 1.0;
      else
        delta[i][j] = 0.0;

  return(delta);
}

double **FIR_low_pass_filter(int32_t wd, int32_t ht)
{
  double **filter = (double **)get_img(wd, ht ,sizeof(double));

  for(int i=0; i<ht; i++)
    for(int j=0; j<wd; j++)
      filter[i][j] = 1.0/(ht*wd);

  return(filter);
}

double **FIR_sharpening_filter(int32_t wd, int32_t ht, float lambda)
{
```

```
36    //  Allocate memory for filter
37    double **filter = (double **)get_img(wd, ht ,sizeof(double));
38
39    // Get low pass filter
40    double **low_pass_filter = FIR_low_pass_filter(wd, ht);
41
42    // Get delta
43    double **delta = get_Delta(wd, ht);
44
45    // Initialize filter
46    for(int i=0; i<ht; i++)
47      for(int j=0; j<wd; j++)
48        filter[i][j] = delta[i][j] + lambda * ( delta[i][j] - low_pass_filter
      [i][j] );
49
50    // Clear used memory for delta and low_pass_filter
51    free_img( (void**)low_pass_filter );
52    free_img( (void**)delta );
53
54    return(filter);
55  }
56
57  uint8_t fix_boundary_clip_val(double pixel_val)
58  {
59    if(pixel_val>255)
60      return(255);
61    else if(pixel_val<0)
62      return(0);
63    else
64      return((uint8_t) pixel_val);
65  }
66
67
68  // Apply FIL low pass filter or sharpening filter on the input image
69  void apply_filter_type12(struct TIFF_img *input_img, struct TIFF_img *
      out_img, char **argv)
70  {
71    int row, col;
72
73    int filter_type = atoi(argv[2]);
74    int filter_wd = atoi(argv[3]);
75    int filter_ht = atoi(argv[4]);
76
77    double **filter;
78    if(filter_type==1)
79      filter = FIR_low_pass_filter(filter_wd, filter_ht);
80    else
81    {
82      float lambda = atof(argv[5]);
83      filter = FIR_sharpening_filter(filter_wd, filter_ht, lambda);
84    }
85
86    // Apply the filter on the input image to get the output image
87    for(int c=0; c<3; c++)
88      for(int i=0; i<(input_img->height); i++)
89        for(int j=0; j<(input_img->width); j++)
90        {
91          double pixel_val = 0.0;
92          for (int fi=0; fi<filter_ht; fi++)
93            for(int fj=0; fj<filter_wd; fj++)
94            {
```

```
            row = i - filter_ht/2 + fi;
            col = j - filter_wd/2 + fj;

            if( row>=0 && col>=0 && row<input_img->height && col<input_img
    ->width )
              pixel_val += filter[fi][fj] * input_img->color[c][row][col];
          }
        out_img->color[c][i][j] = fix_boundary_clip_val(pixel_val);
      }


  // Clear used memory for filter
  free_img( (void**)filter );
}


// Apply IIR Filter on the input image
void apply_filter_type3(struct TIFF_img *input_img, struct TIFF_img *
    out_img, int flag)
{
  double pixel_val;

  // If flag==1 : Create a 256x256 image of the form x(m, n) =   (m-127, n
    -127).
  if(flag==1)
  {
    double **delta = get_Delta(input_img->width, input_img->height);
    for(int c=0; c<3; c++)
      for(int i=0; i<(input_img->height); i++)
        for(int j=0; j<(input_img->width); j++)
          input_img->color[c][i][j] = (uint8_t) delta[i][j];
  }

  // Apply the filter on the input image to get the output image
  for(int c=0; c<3; c++)
    for(int i=0; i<(input_img->height); i++)
      for(int j=0; j<(input_img->width); j++)
      {
        // y(m, n) = 0.01x(m, n) + 0.9(y(m - 1, n) + y(m, n - 1))     0.81y
    (m - 1, n     1)
        if(i>=1 && j>=1)
          pixel_val =  0.01 * input_img->color[c][i][j] + 0.9 * ( out_img->
    color[c][i-1][j] + out_img->color[c][i][j-1] ) - 0.81 * out_img->color[
    c][i-1][j-1];
        else if(i>=1)
          pixel_val =  0.01 * input_img->color[c][i][j] + 0.9 * ( out_img->
    color[c][i-1][j] );
        else if(j>=1)
          pixel_val =  0.01 * input_img->color[c][i][j] + 0.9 * ( out_img->
    color[c][i][j-1] );
        else
          pixel_val =  0.01 * input_img->color[c][i][j];

        if(flag==0)
          out_img->color[c][i][j] = fix_boundary_clip_val(pixel_val);
        else
          out_img->color[c][i][j] = fix_boundary_clip_val(255*100*pixel_val
    );

      }
```

```
146 }
```

Listing 1: filters.c : Contains definitions of all the filters

4.2 FilterImage.c

```
1
2  #include <math.h>
3
4  #include "tiff.h"
5  #include "allocate.h"
6  #include "typeutil.h"
7  #include "filters.h"
8
9  void error(char *name);
10
11 int main (int argc, char **argv)
12 {
13   FILE *fp;
14   struct TIFF_img input_img, out_img;
15   char* outfile;
16   int flag = 0;
17
18   if (argc!=4 && argc != 6 && argc!=7) error( argv[0] );
19
20   /* open image file */
21   if(strcmp(argv[1], "nil")==0)
22     flag = 1;
23
24   if(flag==0)
25   {
26     if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL )
27     {
28       fprintf ( stderr, "cannot open file %s\n", argv[1] );
29       exit ( 1 );
30     }
31
32     if ( read_TIFF ( fp, &input_img ) )
33     {
34       fprintf ( stderr, "error reading file %s\n", argv[1] );
35       exit ( 1 );
36     }
37
38     /* close image file */
39     fclose ( fp );
40
41     /* check the type of image data */
42     if ( input_img.TIFF_type != 'c' ) {
43       fprintf ( stderr, "error:  image must be 24-bit color\n" );
44       exit ( 1 );
45     }
46   }
47   else
48     get_TIFF(&input_img, 256, 256, 'c');
49
50   /* Get appropriate filter :
51       1 : FIR_low_pass_filter
52       2 : FIR_sharpening_filter
53       3 : IIR Filter             */
54
55   int filter_type = atoi(argv[2]);
56   get_TIFF(&out_img, input_img.height, input_img.width, 'c');
57
58   switch(filter_type) {
59       case 1 :
```

```
60            outfile = argv[5];
61            apply_filter_type12(&input_img, &out_img, argv);
62            break;
63        case 2 :
64            outfile = argv[6];
65            apply_filter_type12(&input_img, &out_img, argv);
66            break;
67        case 3 :
68            outfile = argv[3];
69            apply_filter_type3(&input_img, &out_img, flag);
70            break;
71        default :
72            fprintf ( stderr, "error:  Invalid filter_type : value must be 1,
      2 or 3\n" );
73            exit ( 1 );
74     }
75
76    // open output image file
77    if ( ( fp = fopen ( outfile, "wb" ) ) == NULL ) {
78      fprintf ( stderr, "cannot open file %s\n", outfile);
79      exit ( 1 );
80    }
81
82    // write output image
83    if ( write_TIFF ( fp, &out_img ) ) {
84      fprintf ( stderr, "error writing TIFF file %s\n", outfile );
85      exit ( 1 );
86    }
87
88    // close output image file
89    fclose ( fp );
90
91
92    // de-allocate space which was used for the images
93    free_TIFF ( &(input_img) );
94    free_TIFF ( &(out_img) );
95
96    return(0);
97  }
98
99  void error(char *name)
100 {
101     printf("usage:  %s  image.tiff \n\n",name);
102     printf("this program reads in a 24-bit color TIFF image.\n");
103     printf("It then horizontally filters the the image,\n");
104     printf("using the specified filter type : \n");
105     printf("\t{1 : FIR_low_pass_filter, 2 : FIR_sharpening_filter ,3 : IIR
     Filter}\n");
106     printf("and generates an 8-bit color image,\n");
107     printf("that is saved in the output folder");
108     exit(1);
109 }
```

Listing 2: FilterImage.c : The main function that calls filters.c

4.3 plot_DSFT.py

```
from mpl_toolkits import mplot3d

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm

import sys

i = 1j

def FIR_low_pass_filter(mu, nu):
  return (1/81) * np.sin(9/2 * mu) * np.sin(9/2 * nu) / ( np.sin(1/2 * mu) * np.sin(1/2 * nu) + 1e-15 )

def FIR_sharpening_filter_H(mu, nu):
  return (1/25) * np.sin(5/2 * mu) * np.sin(5/2 * nu) / ( np.sin(1/2 * mu) * np.sin(1/2 * nu) + 1e-15 )

def FIR_sharpening_filter_G(mu, nu, lbda):
  return 1 + lbda * ( 1 - (1/25) * np.sin(5/2 * mu) * np.sin(5/2 * nu) / ( np.sin(1/2 * mu) * np.sin(1/2 * nu) + 1e-15 ) )

def IIR_filter(mu, nu):
  H = 0.01 / ( 1 - 0.9*np.exp(-i*mu) - 0.9*np.exp(-i*nu) + 0.81*np.exp(-i*(mu+nu)) + 1e-15)
  return np.abs(H)

def plot(filter_type, title, outfile, lbda):
  mu = np.linspace(-np.pi, np.pi, 50)
  nu = np.linspace(-np.pi, np.pi, 50)

  X, Y = np.meshgrid(mu, nu)

  if filter_type == "lpf":
    Z = FIR_low_pass_filter(X, Y)
  elif filter_type == "sharp_h":
    Z = FIR_sharpening_filter_H(X, Y)
  elif filter_type == "sharp_g":
    Z = FIR_sharpening_filter_G(X, Y, lbda)
  elif filter_type == "iir":
    Z = IIR_filter(X, Y)
  else:
    print("Wrong filter type specified!!\n\n\n Please choose from : lpf, sharp_h, sharp_g, iir")
    return

  fig, ax = plt.subplots(subplot_kw={"projection": "3d"})

  surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap='inferno', edgecolor='none')
  fig.colorbar(surf, shrink=0.5, aspect=5)

  ax.set_xlabel(r'$\mu$')
  ax.set_ylabel(r'$\nu$')
  ax.set_zlabel(r'$H(e^{j\mu}, e^{j\nu})$')

  if filter_type == "sharp_g":
    ax.set_xlim(-3.5, 3.5)
    ax.set_ylim(-3.5, 3.5)
```

```
54      ax.set_zlim(1, 2.8)
55
56    ax.set_title(title)
57
58    plt.savefig(outfile)
59
60  if __name__=="__main__":
61    plot(sys.argv[1], sys.argv[2], sys.argv[3], float(sys.argv[4]))
```

Listing 3: plot_DSFT.py : Function for plotting the magnitude of the frequency response of the filters

4.4 iir_point_spread.py

```
from PIL import Image
import numpy as np
import sys
import matplotlib.pyplot as plt

# y(m, n) = 0.01x(m, n) + 0.9(y(m     1, n) + y(m, n     1))     0.81y(m
      1, n     1)
def IIR_Filter(x):

  # Initialize output array
  y = np.zeros_like(x)

  for i in range(x.shape[0]):
    for j in range(x.shape[1]):
      for c in range(x.shape[2]):
        if(i>=1 and j>=1):
          pixel_val =  0.01 * x[i][j][c] + 0.9 * ( y[i-1][j][c] + y[i][j
   -1][c] ) - 0.81 * y[i-1][j-1][c]
        elif i>=1:
          pixel_val =  0.01 * x[i][j][c] + 0.9 * ( y[i-1][j][c] )
        elif j>=1:
          pixel_val =  0.01 * x[i][j][c] + 0.9 * ( y[i][j-1][c] )
        else:
          pixel_val =  0.01 * x[i][j][c]

        y[i][j][c] = pixel_val

  return y

def main(h, w, c, outfile):
  # Creating delta input : x(m, n) =   ( m 127 ,  n 127 )
  x = np.zeros((h, w, c))
  x[127, 127, :] = [1, 1, 1]

  # Get point spread
  # y(m, n) = 0.01x(m, n) + 0.9(y(m     1, n) + y(m, n     1))     0.81y(m
      1, n     1) ; for x(m, n) =   ( m 127 ,  n 127 )
  y = IIR_Filter(x)

  # Save output image
  img_out = Image.fromarray( (255*100*y).astype(np.uint8) )
  img_out.save(outfile)

if __name__=="__main__":
  main(int(sys.argv[1]), int(sys.argv[2]), int(sys.argv[3]), sys.argv[4])
```

Listing 4: iir_point_spread.py : Function for plotting point spread of the IIR Filter

4.5 tif_to_png.py

```
from PIL import Image
import numpy as np
import sys

im = Image.open(sys.argv[1])
# im.show()
x = np.array(im)

img_out = Image.fromarray(x.astype(np.uint8))
img_out.save(sys.argv[2])
```

Listing 5: tif_to_png.py : Function for converting the tif images to png format

4.6 MakeFile

```
# For Linux or any machines with gcc compiler
CC = gcc
CFLAGS = -std=c99 -Wall -pedantic
BIN = ../bin

all: FilterImage

clean:
  /bin/rm *.o $(BIN)/*

OBJ = tiff.o allocate.o filters.o

FilterImage: FilterImage.o $(OBJ)
  $(CC) $(CFLAGS) -o FilterImage FilterImage.o $(OBJ) -lm
  mv FilterImage $(BIN)
```

Listing 6: MakeFile : Creates binary file for FilterImage.c

4.7 run.sh

```
make

#****************** RUN C# Scipts for Image Filtering *******************
# FIR Low Pass Filter
echo "\n\n\n\nRun FilterImage function using FIR_low_pass_filter on images/
    img03.tif with a 9x9 filter...\n"
bin/FilterImage images/img03.tif 1 9 9 img03_FIR_low_pass_filter.tif
mv img03_FIR_low_pass_filter.tif output
echo "Output saved in output/img03_FIR_low_pass_filter.tif"


# FIR Sharpening Filter
echo "\n\n\n\nRun FilterImage function using FIR_sharpening_filter on
    images/imgblur.tif with a 9x9 filter...\n"
bin/FilterImage images/imgblur.tif  2 9 9 0.2
    imgblur_FIR_sharpening_filter_02.tif
mv imgblur_FIR_sharpening_filter_02.tif output
echo "lambda = 0.2 : output saved in output/
    imgblur_FIR_sharpening_filter_02.tif\n\n"

bin/FilterImage images/imgblur.tif  2 9 9 0.8
    imgblur_FIR_sharpening_filter_08.tif
mv imgblur_FIR_sharpening_filter_08.tif output
echo "lambda = 0.8 : output saved in output/
    imgblur_FIR_sharpening_filter_08.tif\n\n"

bin/FilterImage images/imgblur.tif  2 9 9 1.5
    imgblur_FIR_sharpening_filter_15.tif
mv imgblur_FIR_sharpening_filter_15.tif output
echo "lambda = 1.5 : output saved in output/
    imgblur_FIR_sharpening_filter_15.tif\n\n"


# IIR Filter
echo "\n\n\n\nRun FilterImage function using IIR_filter on images/img03.tif
    ...\n"
bin/FilterImage images/img03.tif 3 img03_IIR_filter.tif
mv img03_IIR_filter.tif output
echo "Output saved in output/img03_IIR_filter.tif"

# IIR Filter using delta as input image
echo "\n\n\n\nRun FilterImage function using IIR_filter on a 256 256  image
     of the form x(m, n) =   ( m 127 ,  n 127 )...\n"
bin/FilterImage nil 3 delta_IIR_filter.tif
mv delta_IIR_filter.tif output
echo "Output saved in output/delta_IIR_filter.tif"



#*************************** Run python scripts ***************************
# IIR Filter using delta as input image
echo "\n\n\n\nGenerate point spread of IIR_filter on a 256 256  image of
    the form x(m, n) =   ( m 127 ,  n 127 )...\n"
python3 src/iir_point_spread.py 256 256 3 output/py_point_spread.tif
echo "Output saved in output/py_point_spread.tif"


#******************** Convert all tiff images to png **********************
echo "\n\n\n\nConverting all .tif files into png format using tif_to_png.py
```

```
    ...\n"
python3 src/tif_to_png.py images/img03.tif output_png/img03.png
python3 src/tif_to_png.py images/imgblur.tif output_png/imgblur.png

python3 src/tif_to_png.py C-code-main/demo/output/color.tif output_png/
    color.png
python3 src/tif_to_png.py C-code-main/demo/output/green.tif output_png/
    green.png


python3 src/tif_to_png.py output/img03_FIR_low_pass_filter.tif output_png/
    py_img03_FIR_low_pass_filter_out.png
python3 src/tif_to_png.py output/imgblur_FIR_sharpening_filter_02.tif
    output_png/py_imgblur_FIR_sharpening_filter_02_out.png
python3 src/tif_to_png.py output/imgblur_FIR_sharpening_filter_08.tif
    output_png/py_imgblur_FIR_sharpening_filter_08_out.png
python3 src/tif_to_png.py output/imgblur_FIR_sharpening_filter_15.tif
    output_png/py_imgblur_FIR_sharpening_filter_15_out.png
python3 src/tif_to_png.py output/img03_IIR_filter.tif output_png/
    img03_IIR_filter.png
python3 src/tif_to_png.py output/delta_IIR_filter.tif output_png/
    delta_IIR_filter.png

python3 src/tif_to_png.py output/py_point_spread.tif output_png/
    py_point_spread.png
echo "Outputs saved in output_png/"



#************************** Plot DSFT **************************
echo "\n\n\n\nGenerating the DSFT magnitude of frequency response plots for
     different filters...\n"
python3 src/plot_DSFT.py lpf FIR_low_pass_filter output_png/
    DSFT_FIR_low_pass_filter.png 0
python3 src/plot_DSFT.py sharp_h FIR_sharpening_filter_H output_png/
    DSFT_FIR_sharpening_filter_H.png 0.2
python3 src/plot_DSFT.py sharp_g FIR_sharpening_filter_G output_png/
    DSFT_FIR_sharpening_filter_G_02.png 0.2
python3 src/plot_DSFT.py sharp_g FIR_sharpening_filter_G output_png/
    DSFT_FIR_sharpening_filter_G_08.png 0.8
python3 src/plot_DSFT.py sharp_g FIR_sharpening_filter_G output_png/
    DSFT_FIR_sharpening_filter_G_15.png 1.5
python3 src/plot_DSFT.py iir IIR_filter output_png/DSFT_IIR_filter.png 0
```

Listing 7: run.sh : Generates all outputs in the report