ECE 637 Digital Image Filtering Laboratory: Image Filtering

Yang Wang

January 21, 2016

1 C Programming

Nothing due for report.

2 Displaying and Exporting Images in Matlab

Nothing due for report.

3 FIR Low Pass Filter

In this section, an supplied image is filtered with a 9 $\times$ 9 low pass filter given as:

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

3.1 Find an expression for low pass filter

By plugging into the DSFT formula, we have:

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

We know the formula:

$$\sum_{n=-N}^{N} e^{-j\omega n} = e^{j\omega N} \frac{1 - e^{-j\omega(2N+1)}}{1 - e^{-j\omega}} \tag{2}$$

Therefore,

$$H(e^{j\mu}, e^{j\nu}) = \frac{1}{81} \sum_{n=-4}^{4} \frac{e^{j4\mu} - e^{-j5\mu}}{1 - e^{-j\mu}} \sum_{n=-4}^{4} \frac{e^{j4\nu} - e^{-j5\nu}}{1 - e^{-j\nu}}$$
$$= \frac{1}{81} \frac{sin(\frac{9}{2}\mu)}{sin(\frac{1}{2}\mu)} \frac{sin(\frac{9}{2}\nu)}{sin(\frac{1}{2}\nu)}$$

3.2 Plot the magnitude of LPF

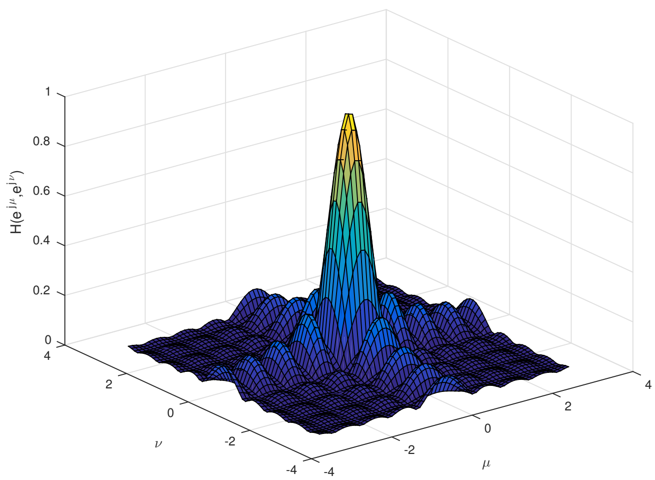

Figure 1: Magnitude of LPF.

3.3 Original vs. filtered image

The images from the next page shows that the filtered image is more blurred out than the original image as it goes through the low pass filter.

(a) img03.tif

(b) lpfimg03.tif

Figure 2: Original vs. Low-Pass Filtered Image

3.4 Code listing

3.4.1 firlpf.c

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

#define FH 9
#define FW 9

int main (int argc, char **argv)
{
    FILE *fp;
    struct TIFF_img input_img, output_img;
    int i, j;

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

```
double *lpf[FH];
for (i = 0; i < FH; i++) {
    lpf[i] = (double *)malloc(FW * sizeof(double));
}

for (i = 0; i < FH; i++) {
    for(j = 0; j < FW; j++) {
        lpf[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 color image */
get_TIFF(&output_img, input_img.height, input_img.width, 'c');

/* filter the image with lpf */
conv2d(&input_img, &output_img, FH, FW, lpf);

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

/* write color image */
```

```
    if(write_TIFF(fp, &output_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(&(output_img));

    /* free filter array */
    for (i = 0; i < FW; i++) {
        free(lpf[i]);
    }

    return(0);
}
```

3.4.2 defs.c

```
#include "defs.h"

void erroriirf(char *name);
void errorlsf(char *name);
void errorlpf(char *name);
uint8_t constrain(double pixel_color);
void conv2d(struct TIFF_img *iimg, struct TIFF_img *oimg,
            int fh, int fw, double **filter);

uint8_t constrain(double pixel_color) {
    uint8_t color;
    if (pixel_color > 255) {
        color = 255;
    } else if (pixel_color < 0) {
        color = 0;
    } else {
        color = (uint8_t)pixel_color;
    }
    return color;
}

void conv2d(struct TIFF_img *iimg, struct TIFF_img *oimg,
            int fh, int fw, double **filter) {
    int hl = (fh - 1) / 2;
```

```
    int wl = (fw - 1) / 2;
    int ih = iimg->height;
    int iw = iimg->width;
    int32_t i,j,m,n,r,c;
    double rt, gt, bt;

    for (i = 0; i < ih; i++) {
        for (j = 0; j < iw; j++) {
            rt = 0.0;
            gt = 0.0;
            bt = 0.0;
            for (m = -hl; m <= hl; m++) {
                for (n = -wl; n <= wl; n++) {
                    r = i-m;
                    c = j-n;
                    if (r < ih && r >=0 && c < iw && c >= 0) {
                        rt += filter[m+hl][n+wl] * iimg->color[0][r][c];
                        gt += filter[m+hl][n+wl] * iimg->color[1][r][c];
                        bt += filter[m+hl][n+wl] * iimg->color[2][r][c];
                    }
                }
            }
            oimg->color[0][i][j] = constrain(rt);
            oimg->color[1][i][j] = constrain(gt);
            oimg->color[2][i][j] = constrain(bt);
        }
    }
}

void errorlpf(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 filters the image with a 9x9 FIR low pass filter.\n\n");
  exit(1);
}

void errorsf(char *name)
{
  printf("usage: %s image.tiff lambda\n\n", name);
  printf("This program reads in a 24-bit color TIFF image.\n");
    printf("It then filters the image with a 5x5 FIR sharpening filter,\n");
  printf("with supplied lambda value.\n");
  printf("The greater lambda is, the higher sharpening intensity.\n\n");
  exit(1);
}
```

```
void erroriirf(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 filters the image with a IIR filter,\n\n");
  exit(1);
}
```

3.4.3 defs.h

```
#ifndef _DEFS_H_
#define _DEFS_H_

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdarg.h>
#include <math.h>

#include "typeutil.h"
#include "tiff.h"

void errorlpf(char *name);
void errorsf(char *name);
void erroriirf(char *name);
uint8_t constrain(double pixel_color);
void conv2d(struct TIFF_img *iimg, struct TIFF_img *oimg,
            int fh, int fw, double **filter);

#endif /* _DEFS_H_ */
```

4 FIR Sharpening Filter

In this section, the effect of a FIR sharpening filter on an image is analyzed. An supplied image is filtered with a FIR sharpening filter H(m,n) which is constructed using a 5 × 5 FIR low pass filter given as:

$$h(m,n) = \begin{cases} 1/25 & \text{for } |m| \leq 2,\ |n| \leq 2 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

and,

$$g(m,n) = \delta(m,n) + \lambda(\delta(m,n) - h(m,n)) \tag{4}$$

4.1 Find an expression for FIR low pass filter

Using the same method as in Section 3, we have:

$$\begin{aligned} H(e^{j\mu}, e^{j\nu}) &= \sum_{n=-2}^{2} \sum_{m=-2}^{2} \frac{1}{25} e^{-j(m\mu+n\nu)} \\ &= \frac{1}{25} \sum_{n=-2}^{2} e^{-jm\mu} \sum_{n=-2}^{2} e^{-jm\nu} \end{aligned}$$

Using equation (2):

$$\begin{aligned} H(e^{j\mu}, e^{j\nu}) &= \frac{1}{25} \sum_{n=-2}^{2} \frac{e^{j2\mu} - e^{-j3\mu}}{1 - e^{-j\mu}} \sum_{n=-2}^{2} \frac{e^{j2\nu} - e^{-j3\nu}}{1 - e^{-j\nu}} \\ &= \frac{1}{25} \frac{sin(\frac{5}{2}\mu)}{sin(\frac{1}{2}\mu)} \frac{sin(\frac{5}{2}\nu)}{sin(\frac{1}{2}\nu)} \end{aligned}$$

4.2 Find an expression for FIR sharpening filter

Using the result in Section 4.2, we have:

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

4.3 Plot the magnitude of HPF

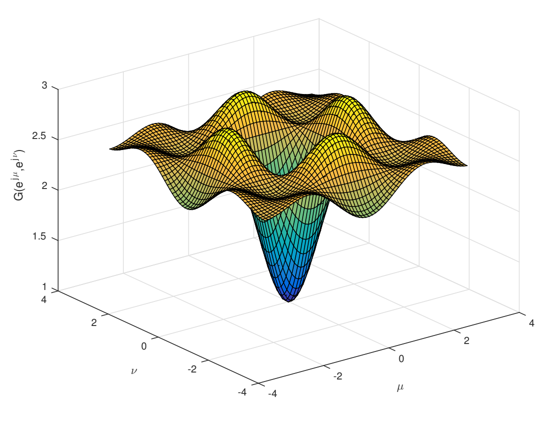

Figure 3: Magnitude of HPF.

4.4 Original vs. filtered image

The images from the next page shows that the filtered image is more sharpened than the original image as it goes through the high pass filter.

(a) imgblur.tif

(b) sfimg03.tif

Figure 4: Original vs. High-Pass Filtered Image

4.5 Code listing

4.5.1 firsf.c

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

#define FH 5
#define FW 5

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

```c
struct TIFF_img input_img, output_img;
int i, j;
double lamda;
double delta;
char *lamdastr;

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

lamda = strtod(argv[2], &lamdastr);

double *sf[FH];
for (i = 0; i < FH; i++) {
    sf[i] = (double *)malloc(FW * sizeof(double));
}

for (i = 0; i < FH; i++) {
    for(j = 0; j < FW; j++) {
        if(i == 2 && j == 2) {
            delta = 1.0;
        } // m = n = 0
        else {
            delta = 0.0;
        }
        sf[i][j] = delta + lamda * (delta-1.0/25.0);
    }
}

/* 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);
```

```
    }

    /* set up structure for output color image */
    get_TIFF(&output_img, input_img.height, input_img.width, 'c');

    /* filter the image with lpf */
    conv2d(&input_img, &output_img, FH, FW, sf);

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

    /* write color image */
    if(write_TIFF(fp, &output_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(&(output_img));

    /* free filter array */
    for (i = 0; i < FW; i++) {
        free(sf[i]);
    }

    return(0);
}
```

5 IIR Filter

In this section, the effect of an IIR filter specified by a 2-D difference equation is analyzed. The 2-D equation is given as the following:

$$y(m,n) = 0.01x(m,n)+0.9(y(m-1,n)+y(m,n-1))-0.81(y(m-1,n-1)) \quad (5)$$

5.1 Find an expression for IIR

Taking the Z-transform of both sides on equation (5):

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

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

$$\frac{Y(z_1, z_2)}{X(z_1, z_2)} = H(z_1, z_2) = \frac{0.01}{1 - 0.9z_1^{-1} - 0.9z_2^{-1} + 0.81z_1^{-1}z_2^{-1}} \tag{6}$$

We have the relations:

$$z_1 = e^{j\mu}$$
$$z_2 = e^{j\nu}$$

Substituting into equation (6):

$$H(e^{j\mu}, e^{j\nu}) = \frac{0.01}{1 - 0.9e^{-j\mu} - 0.9e^{-j\nu} + 0.81e^{-j(\mu+\nu)}} \tag{7}$$

5.2 Plot the magnitude of IIR Filter

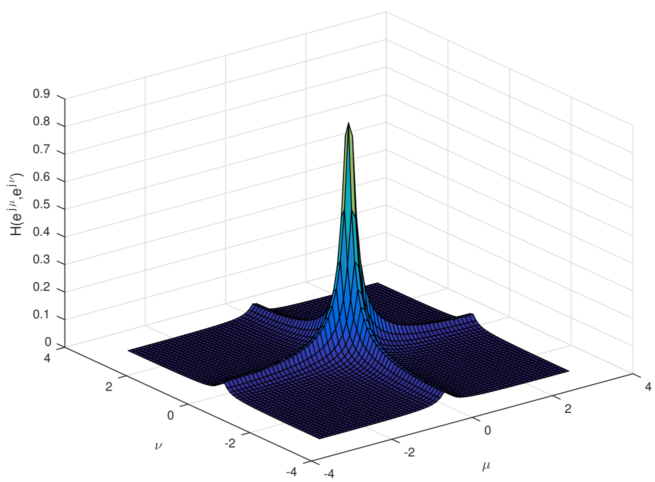

Figure 5: Magnitude of IIR Filter.

5.3 Original vs. filtered image

The images from the next page shows that the filtered image is ”smeared out”.

(a) img12.tif

(b) iirfimg12.tif

Figure 6: Original vs. IIR Filtered Image

5.4 Code listing

5.4.1 iirf.c

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

int main (int argc, char **argv)
{
    FILE *fp;
    struct TIFF_img input_img, output_img;
    int i, j, m, ih, iw;

    if (argc != 2) erroriirf(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 */
```

```c
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 color image */
get_TIFF(&output_img, input_img.height, input_img.width, 'c');

ih = input_img.height;
iw = input_img.width;
double ***ct = (double ***)malloc(3 * sizeof(double **));
for (m = 0; m < 3; m++) {
    ct[m] = (double **)malloc(ih * sizeof(double *));
    for (i = 0; i < ih; i++) {
        ct[m][i] = (double *)malloc(iw * sizeof(double));
    }
}

/* filter image with IIR filter */
for (m = 0; m < 3; m++) {
    for (i = 0; i < ih; i++) {
        for (j = 0; j < iw; j++) {
            ct[m][i][j] = 0.01 * input_img.color[m][i][j];
            if (i > 0) {
                ct[m][i][j] += 0.9 * ct[m][i-1][j];
            }
            if (j > 0) {
                ct[m][i][j] += 0.9 * ct[m][i][j-1];
            }
            if (i > 0 && j > 0) {
                ct[m][i][j] += -0.81 * ct[m][i-1][j-1];
            }
        }
    }
} // it needs to run until the end

/* write RGB colors to output image*/
for (m = 0; m < 3; m++) {
    for (i = 0; i < ih; i++) {
        for (j = 0; j < iw; j++) {
            output_img.color[m][i][j] = constrain(ct[m][i][j]);
        }
    }
```

```
    }

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

    /* write color image */
    if(write_TIFF(fp, &output_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(&(output_img));

    /* free color temp array */
    for (m = 0; m < 3; m++) {
        for (i = 0; i < ih; i++) {
            free(ct[m][i]);
        }
        free(ct[m]);
    }
    free(ct);

    return(0);
}
```