ECE 637 Digital Image Processing
Lab 1 Image Filtering

Tong Shen

January 20th, 2017

3. FIR Low Pass Filter

In this problem, you will analyze and implement a simple low pass filter given by the 9 × 9 point spread function:

h(m, n) = 1/81 for $|m| \leq 4$ and $|n| \leq 4$ and 0 otherwise.

3.1 Derive DSFT of FIR Low Pass Filter

Figure 1 shows the handwriting calculation process:

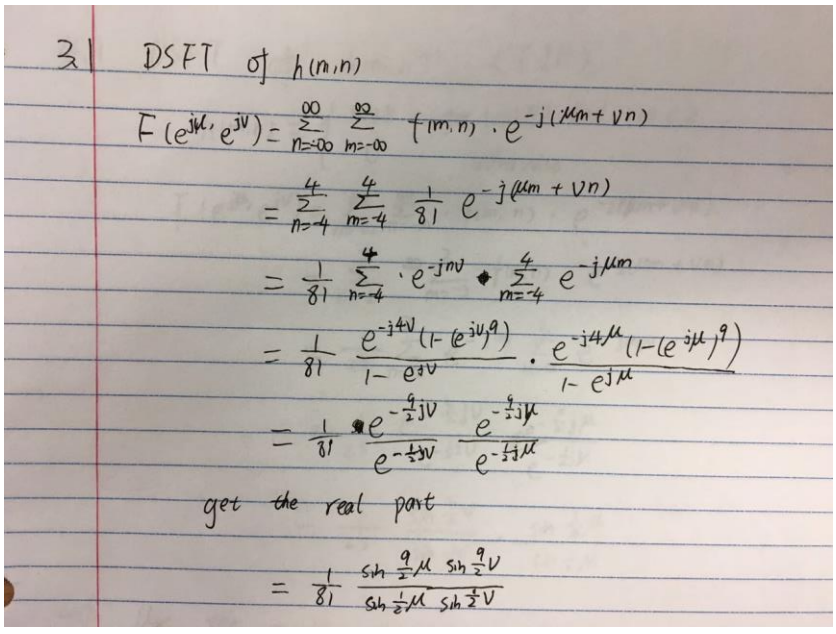


Figure 1. Derivation of the DSFT

3.2 Plot of DSFT of the Low Pass Filter

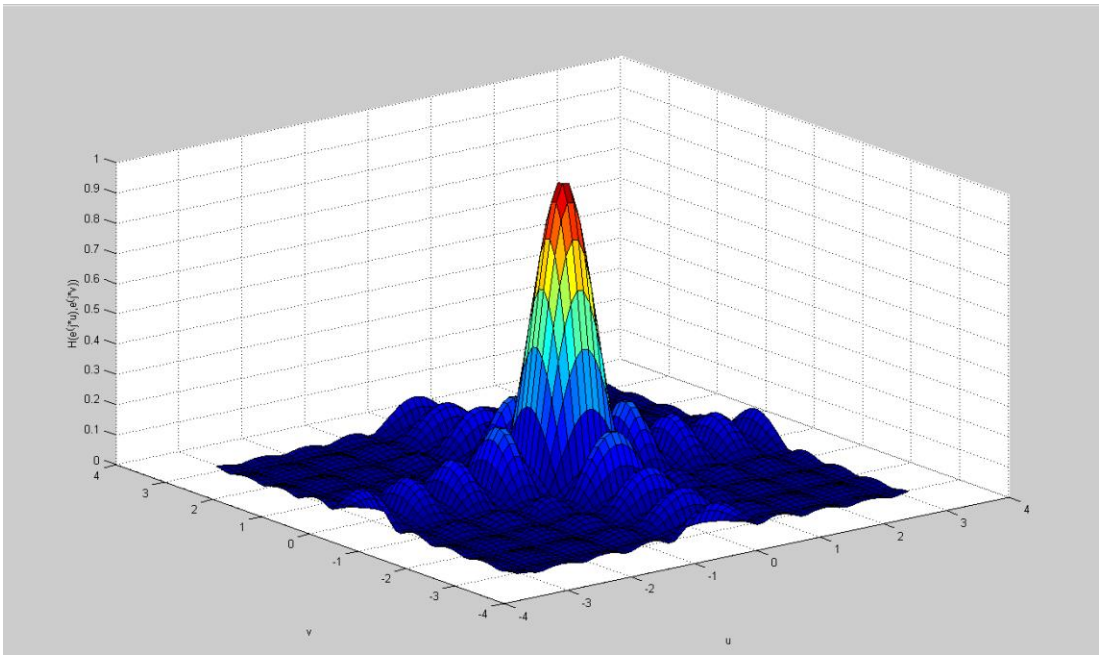
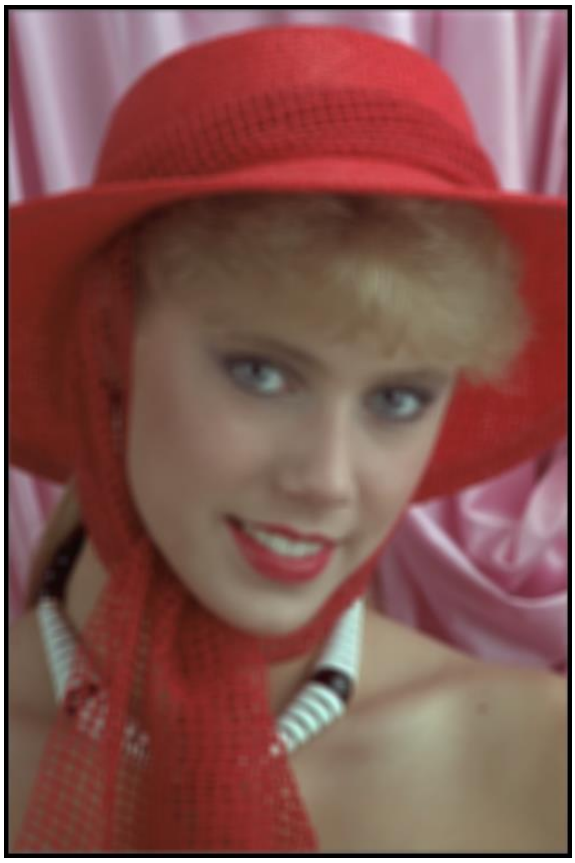


Figure 2 DSFT of the Low Pass Filter

3.3 The original image and the filtered image

Figure 3 displays the original and the low pass filtered image. The image gets blurred after going through a low pass filter because the loss of high frequency component.

Img03.tif



lpfilterd.tif

Figure 3: the comparison of the original and filtered image

3.4 Coding List

Lab1_LpFilter.c

This code includes image loading, processing(2-D convolution), and writing.

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

void error(char *name);

int main(int argc, char **argv)
```

```
{
        FILE *fp;
        struct TIFF_img input_img, filterd_img;
        double **img1, **img2, **img3, **oimg1, **oimg2, **oimg3;
        int32_t i, j, p, q;
        double pixel_r, pixel_g, pixel_b;

        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);

        /* check the type of image data */
        if (input_img.TIFF_type != 'c') {
                fprintf(stderr, "error:  image must be 24-bit color\n");
                exit(1);
        }

        /* Allocate image of double precision floats */
        img1 = (double **)get_img(input_img.width, input_img.height, sizeof(double));
        img2 = (double **)get_img(input_img.width, input_img.height, sizeof(double));
        img3 = (double **)get_img(input_img.width, input_img.height, sizeof(double));
        oimg1 = (double **)get_img(input_img.width, input_img.height, sizeof(double));
        oimg2 = (double **)get_img(input_img.width, input_img.height, sizeof(double));
        oimg3 = (double **)get_img(input_img.width, input_img.height, sizeof(double));

        /* copy red/green/blue component to double array */
        for (i = 0; i < input_img.height; i++)
                for (j = 0; j < input_img.width; j++) {
                        img1[i][j] = input_img.color[0][i][j];
                        img2[i][j] = input_img.color[1][i][j];
                        img3[i][j] = input_img.color[2][i][j];
                }


        /* Filter image with the FIR*/
        for (i = 4; i < input_img.height - 4; i++) {
                for (j = 4; j < input_img.width - 4; j++) {
                        pixel_b = 0;
                        pixel_g = 0;
```

```
                pixel_r = 0;
                for (p = 0; p < 9; p++) {
                        for (q = 0; q < 9; q++)
                        {
                                pixel_r = pixel_r + img1[i + (p - 4)][j + (q - 4)]/81;
                                pixel_g = pixel_g + img2[i + (p - 4)][j + (q - 4)]/81;
                                pixel_b = pixel_b + img3[i + (p - 4)][j + (q - 4)]/81;
                        }
                }
                oimg1[i][j] = pixel_r;
                oimg2[i][j] = pixel_g;
                oimg3[i][j] = pixel_b;
        }
}



/* Fill in boundary pixels */
for (i = 0; i < input_img.height; i++)
        for (j = 0; j < input_img.width; j++) {
                if (i<4 || i>input_img.height - 4) {
                        oimg3[i][j] = 0;
                        oimg1[i][j] = 0;
                        oimg2[i][j] = 0;
                }
                if (j<4 || j>input_img.width - 4) {
                        oimg3[i][j] = 0;
                        oimg1[i][j] = 0;
                        oimg2[i][j] = 0;
                }
        }


/* set up structure for output color image */
/* Note that the type is 'c' rather than 'g' */
get_TIFF(&filterd_img, input_img.height, input_img.width, 'c');


/* Illustration: constructing a sample color image -- interchanging the red and green
components from the input color image */
for (i = 0; i < input_img.height; i++)
        for (j = 0; j < input_img.width; j++) {
                if (oimg1[i][j] > 255)
                {
                        oimg1[i][j] = 255;
                }
                if (oimg1[i][j] < 0)
                {
                        oimg1[i][j] = 0;
                }
                if (oimg2[i][j] > 255)
                {
```

```
                        oimg2[i][j] = 255;
                }
                if (oimg2[i][j] < 0)
                {
                        oimg2[i][j] = 0;
                }
                if (oimg3[i][j] > 255)
                {
                        oimg3[i][j] = 255;
                }
                if (oimg3[i][j] < 0)
                {
                        oimg3[i][j] = 0;
                }
                filterd_img.color[0][i][j] = oimg1[i][j];
                filterd_img.color[1][i][j] = oimg2[i][j];
                filterd_img.color[2][i][j] = oimg3[i][j];
        }

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

    /* write color image */
    if (write_TIFF(fp, &filterd_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(&(filterd_img));

    free_img((void**)img1);
    free_img((void**)img2);
    free_img((void**)img3);
    free_img((void**)oimg1);
    free_img((void**)oimg2);
    free_img((void**)oimg3);

    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. FIR Sharping Filtering

In this problem, you 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) be a low pass filter. For our purposes use h(m, n) = 1/25 for $|m| \leqslant 2$ and $|n| \leqslant 2$ and 0 otherwise .

The unsharp mask filter is then given by $g(m, n) = \delta(m, n) + \lambda(\delta(m, n) - h(m, n))$ where λ is a constant greater than zero.

4.1 DSFT of the Low Pass Filter

Figure 4 shows the derivation of the DSFT of the Low Pass Filter

4.2 DSFT of the FIR Sharpening Filter

Figure 4 shows the derivation of the DSFT of the FIR Sharpening Filter

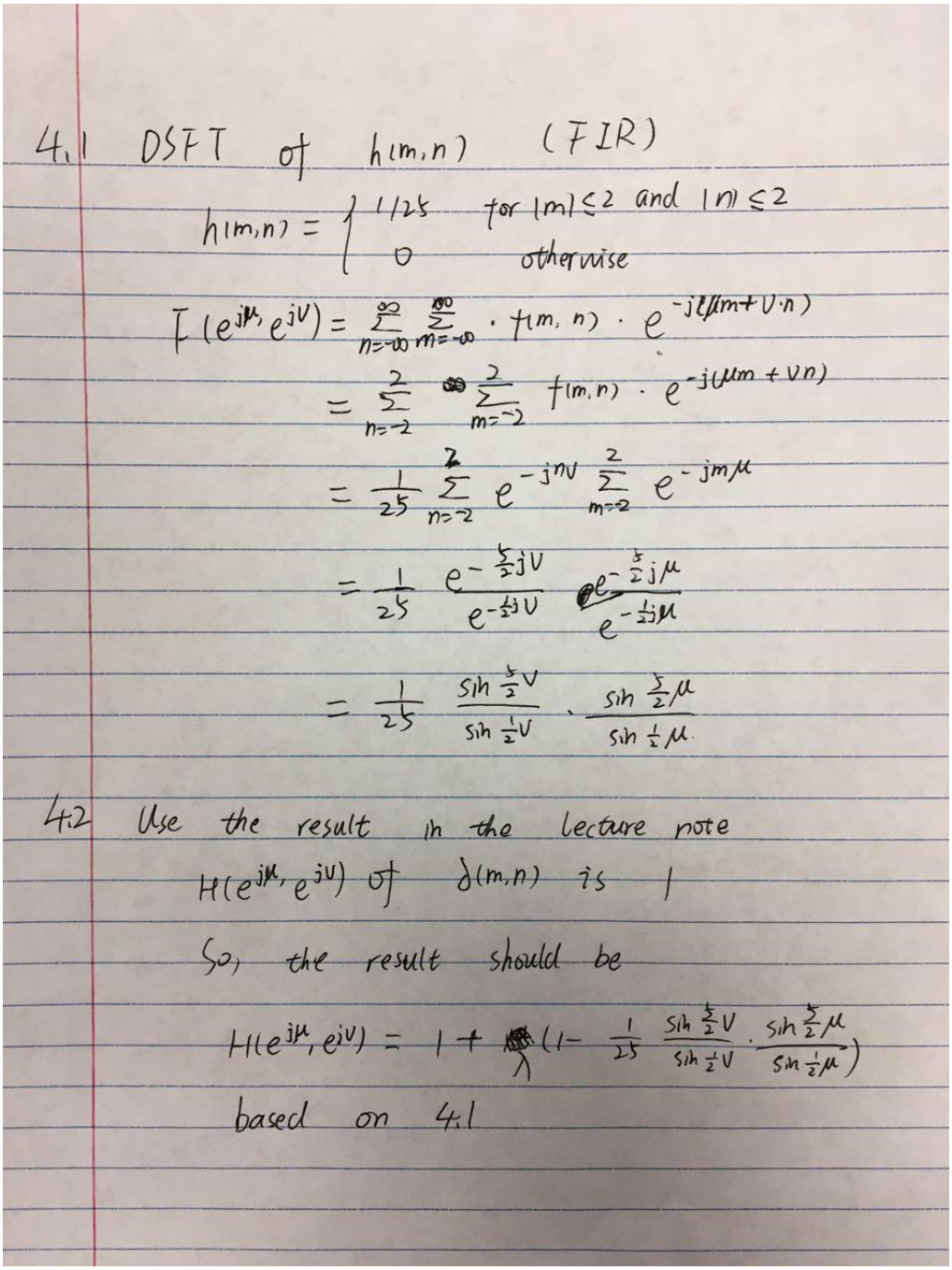


Figure 4: DSFT of the low pass and sharpening filter

4.3 Plot of the DSFT of the LPT

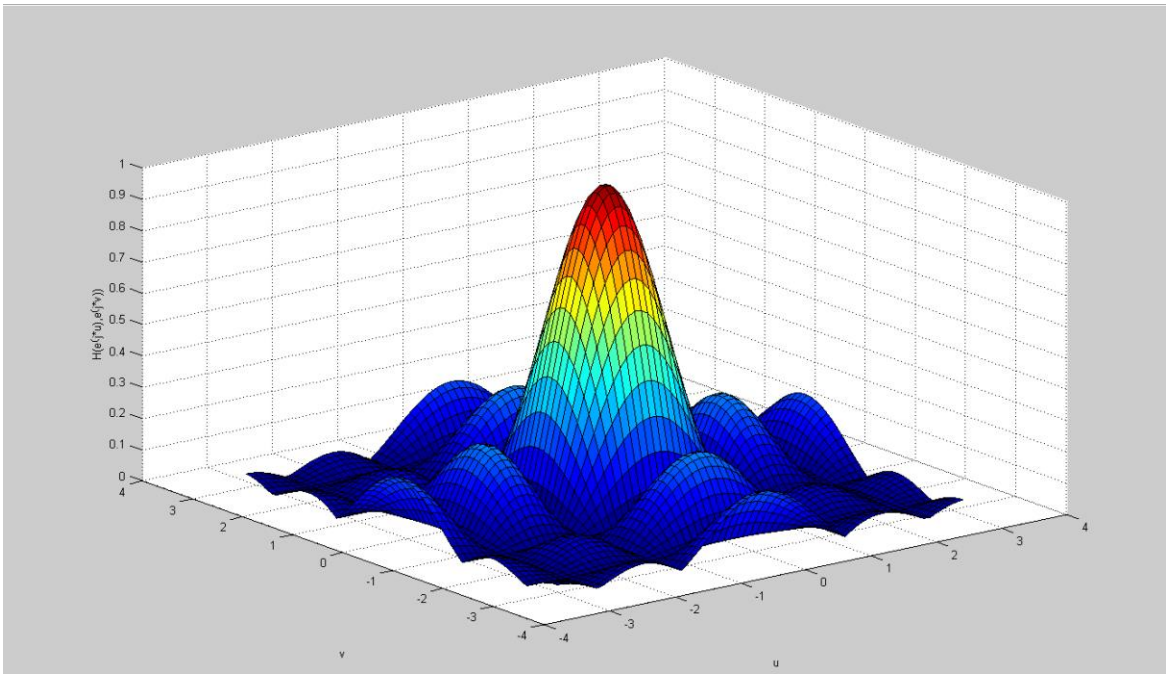


Figure 5: Plot of the DSFT of the LPT

4.4 Plot of the DSFT of the FIR Sharpening Filter

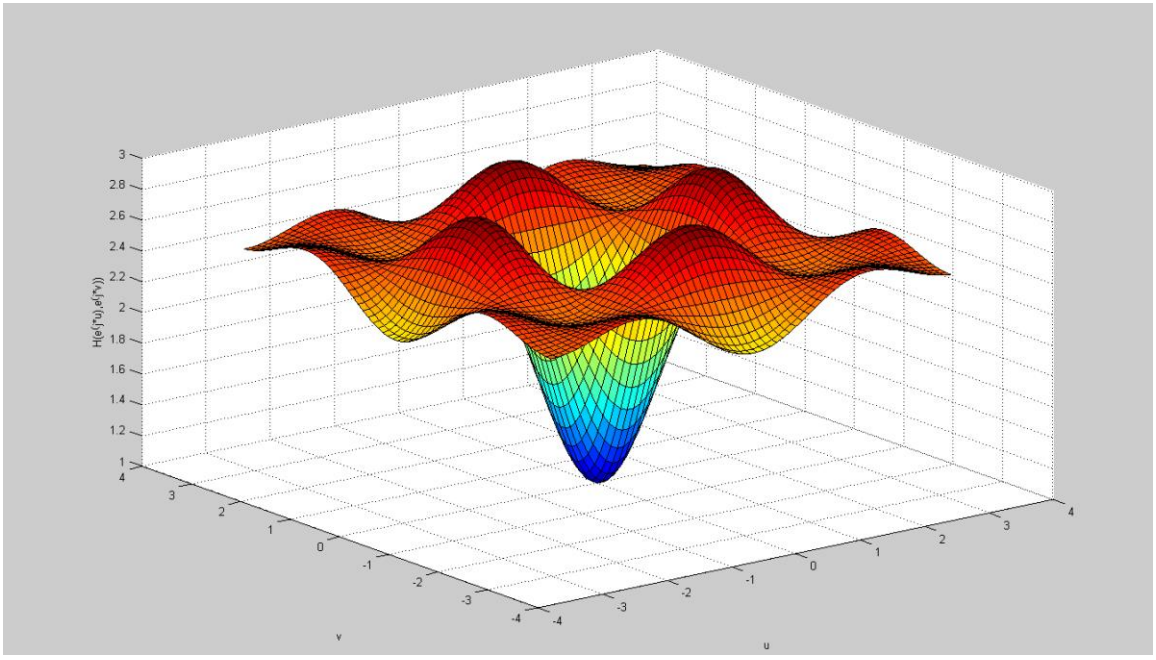


Figure 6: Plot of the DSFT of the FIR Sharpening Filter

4.5 Original image and the filtered image with lambda 1.5

imgblur.tif FIR.tif

Figure 7: the comparison of the original and sharpen image

4.6 Code Listing

FIR.c

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

void error(char *name);

int main(int argc, char **argv)
{
        FILE *fp;
        struct TIFF_img input_img, filterd_img;
        double **img1, **img2, **img3, **oimg1, **oimg2, **oimg3;
        int32_t i, j, p, q;
        double pixel_r, pixel_g, pixel_b;

        double lambda;
```

```
if (argc != 3) error(argv[0]);
lambda = strtod(argv[2], NULL);  // get the value of lambda from client input

/* 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);
}

/* Allocate image of double precision floats */
img1 = (double **)get_img(input_img.width, input_img.height, sizeof(double));
img2 = (double **)get_img(input_img.width, input_img.height, sizeof(double));
img3 = (double **)get_img(input_img.width, input_img.height, sizeof(double));
oimg1 = (double **)get_img(input_img.width, input_img.height, sizeof(double));
oimg2 = (double **)get_img(input_img.width, input_img.height, sizeof(double));
oimg3 = (double **)get_img(input_img.width, input_img.height, sizeof(double));

/* copy red/green/blue component to double array */
for (i = 0; i < input_img.height; i++)
        for (j = 0; j < input_img.width; j++) {
                img1[i][j] = input_img.color[0][i][j];
                img2[i][j] = input_img.color[1][i][j];
                img3[i][j] = input_img.color[2][i][j];
        }


/* Filter image with the Low Pass Filter*/
for (i = 2; i < input_img.height - 2; i++) {
        for (j = 2; j < input_img.width - 2; j++) {
                pixel_b = 0;
                pixel_g = 0;
                pixel_r = 0;
                for (p = 0; p < 5; p++) {
                        for (q = 0; q < 5; q++)
                        {
                                if (p == 2 && q == 2) {
                                        pixel_r = pixel_r + (1 + lambda)*img1[i][j];
```

```
                                        pixel_g = pixel_g + (1 + lambda)*img2[i][j];
                                        pixel_b = pixel_b + (1 + lambda)*img3[i][j];
                                }
                        pixel_r = pixel_r - lambda*img1[i + (p - 2)][j + (q - 2)] / 25;
                        pixel_g = pixel_g - lambda*img2[i + (p - 2)][j + (q - 2)] / 25;
                        pixel_b = pixel_b - lambda*img3[i + (p - 2)][j + (q - 2)] / 25;
                        }
                }
                oimg1[i][j] = pixel_r;
                oimg2[i][j] = pixel_g;
                oimg3[i][j] = pixel_b;
        }
}



/* Fill in boundary pixels */
for (i = 0; i < input_img.height; i++) {
        oimg3[i][0] = 0;
        oimg1[i][0] = 0;
        oimg2[i][0] = 0;
        oimg1[i][input_img.width - 1] = 0;
        oimg2[i][input_img.width - 1] = 0;
        oimg3[i][input_img.width - 1] = 0;
        oimg3[i][1] = 0;
        oimg1[i][1] = 0;
        oimg2[i][1] = 0;
        oimg1[i][input_img.width - 2] = 0;
        oimg2[i][input_img.width - 2] = 0;
        oimg3[i][input_img.width - 2] = 0;
}
for (j = 0; j < input_img.width; j++) {
        oimg1[0][j] = 0;
        oimg2[0][j] = 0;
        oimg3[0][j] = 0;
        oimg1[input_img.height - 1][j] = 0;
        oimg2[input_img.height - 1][j] = 0;
        oimg3[input_img.height - 1][j] = 0;
        oimg1[1][j] = 0;
        oimg2[1][j] = 0;
        oimg3[1][j] = 0;
        oimg1[input_img.height - 2][j] = 0;
        oimg2[input_img.height - 2][j] = 0;
        oimg3[input_img.height - 2][j] = 0;
}

/* set up structure for output color image */
/* Note that the type is 'c' rather than 'g' */
get_TIFF(&filterd_img, input_img.height, input_img.width, 'c');
```

```
	/* Illustration: constructing a sample color image -- interchanging the red and green
components from the input color image */
	for (i = 0; i < input_img.height; i++)
		for (j = 0; j < input_img.width; j++) {
			if (oimg1[i][j] > 255)
			{
				oimg1[i][j] = 255;
			}
			if (oimg1[i][j] < 0)
			{
				oimg1[i][j] = 0;
			}
			if (oimg2[i][j] > 255)
			{
				oimg2[i][j] = 255;
			}
			if (oimg2[i][j] < 0)
			{
				oimg2[i][j] = 0;
			}
			if (oimg3[i][j] > 255)
			{
				oimg3[i][j] = 255;
			}
			if (oimg3[i][j] < 0)
			{
				oimg3[i][j] = 0;
			}
			filterd_img.color[0][i][j] = oimg1[i][j];
			filterd_img.color[1][i][j] = oimg2[i][j];
			filterd_img.color[2][i][j] = oimg3[i][j];
		}

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

	/* write color image */
	if (write_TIFF(fp, &filterd_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(&(filterd_img));
```

```
        free_img((void**)img1);
        free_img((void**)img2);
        free_img((void**)img3);
        free_img((void**)oimg1);
        free_img((void**)oimg2);
        free_img((void**)oimg3);

        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);
}
```

5 IIR Filter

In this problem, you 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 and y(m, n) is the output.

5.1 DSFT of IIR Filter

The Z-Transform of the Filter:

$$Y(Z_1, Z_2) = 0.01X(Z_1, Z_2) + 0.9\left(Z_1^{-1} + Z_2^{-1}\right)Y(Z_1, Z_2) - 0.81Z_1^{-1}Z_2^{-1}Y(Z_1, Z_2)$$

And $Z_1 = e^{j\mu}, Z_2 = e^{j\nu}$

Accordingly $H\left(e^{j\mu}, e^{j\nu}\right) = H(Z_1, Z_2) = \frac{Y(Z_1, Z_2)}{X(Z_1, Z_2)} = \frac{0.01}{1 - 0.9\left(e^{-j\mu} + e^{-j\nu}\right) + 0.81e^{-j(\mu+\nu)}}$

5.2 Plot of the DSFT of the IIR Filter

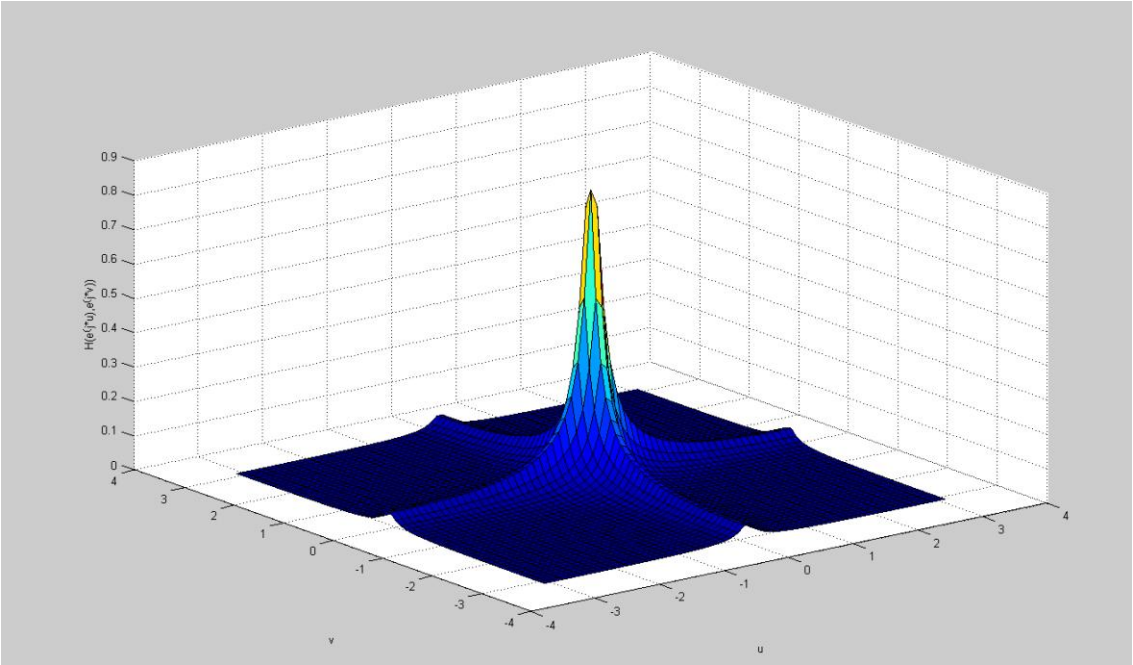


Figure 8: The DSFT of the IIR Filter

5.3 An image of the point spread function

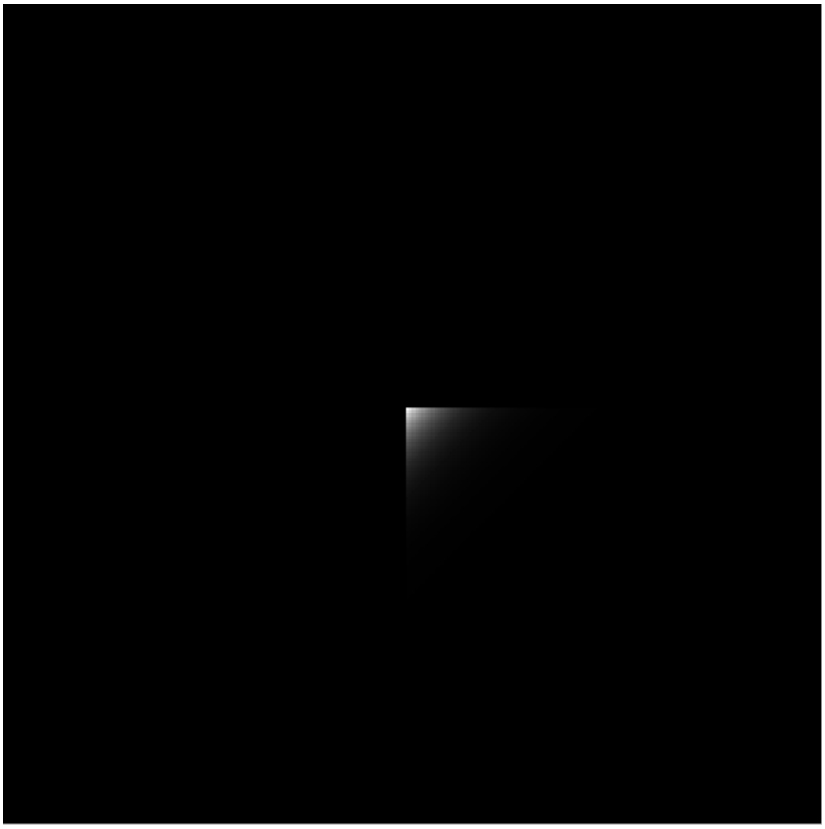


Figure 9: Point Spread Function

5.4 Original Image and the Filtered Image

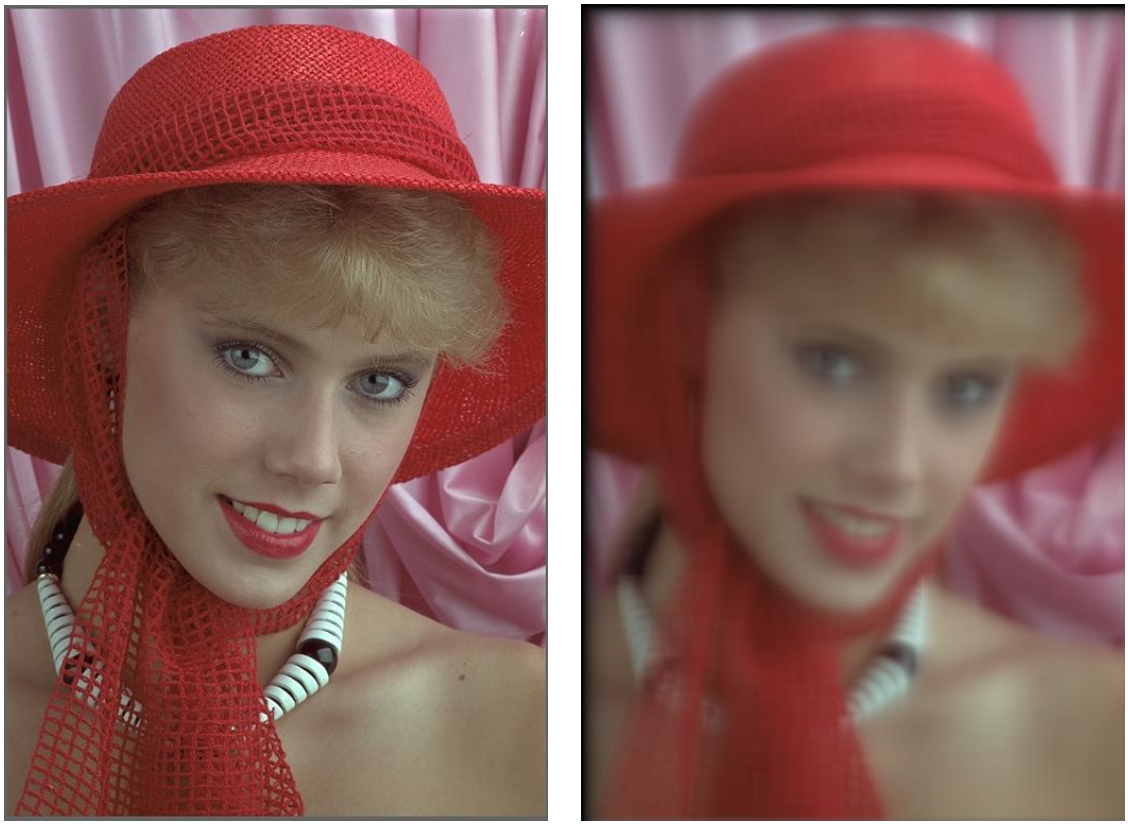


Figure 10: Comparison of the original and IIR filtered image

5.5 Code Listing

IIR.c

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

void error(char *name);

int main(int argc, char **argv)
{
        FILE *fp;
        struct TIFF_img input_img, filterd_img;
        double **img1, **img2, **img3, **oimg1, **oimg2, **oimg3;
        int32_t i, j;
        double pixel_r, pixel_g, pixel_b;
```

```
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);

/* check the type of image data */
if (input_img.TIFF_type != 'c') {
        fprintf(stderr, "error:  image must be 24-bit color\n");
        exit(1);
}

/* Allocate image of double precision floats */
img1 = (double **)get_img(input_img.width, input_img.height, sizeof(double));
img2 = (double **)get_img(input_img.width, input_img.height, sizeof(double));
img3 = (double **)get_img(input_img.width, input_img.height, sizeof(double));
oimg1 = (double **)get_img(input_img.width, input_img.height, sizeof(double));
oimg2 = (double **)get_img(input_img.width, input_img.height, sizeof(double));
oimg3 = (double **)get_img(input_img.width, input_img.height, sizeof(double));

/* copy red/green/blue component to double array */
for (i = 0; i < input_img.height; i++)
        for (j = 0; j < input_img.width; j++) {
                img1[i][j] = input_img.color[0][i][j];
                img2[i][j] = input_img.color[1][i][j];
                img3[i][j] = input_img.color[2][i][j];
        }

for (i = 0; i < input_img.height; i++) {
        for (j = 0; j < input_img.width; j++) {
                pixel_r = 0;
                pixel_g = 0;
                pixel_b = 0;

                pixel_r = pixel_r + 0.01*img1[i][j];
                pixel_g = pixel_g + 0.01*img2[i][j];
                pixel_b = pixel_b + 0.01*img3[i][j];

                if (i > 1 && j > 1) {
                        pixel_r = pixel_r - 0.81*oimg1[i - 1][j - 1];
```

```
                        pixel_g = pixel_g - 0.81*oimg2[i - 1][j - 1];
                        pixel_b = pixel_b - 0.81*oimg3[i - 1][j - 1];
                }
                if (j > 1) {
                        pixel_r = pixel_r + 0.9*oimg1[i][j - 1];
                        pixel_g = pixel_g + 0.9*oimg2[i][j - 1];
                        pixel_b = pixel_b + 0.9*oimg3[i][j - 1];
                }
                if (i > 1) {
                        pixel_r = pixel_r + 0.9*oimg1[i - 1][j];
                        pixel_g = pixel_g + 0.9*oimg2[i - 1][j];
                        pixel_b = pixel_b + 0.9*oimg3[i - 1][j];
                }
                oimg1[i][j] = pixel_r;
                oimg2[i][j] = pixel_g;
                oimg3[i][j] = pixel_b;
        }
    }




    /* set up structure for output color image */
    /* Note that the type is 'c' rather than 'g' */
    get_TIFF(&filterd_img, input_img.height, input_img.width, 'c');


    /* Illustration: constructing a sample color image -- interchanging the red and green
components from the input color image */
    for (i = 0; i < input_img.height; i++)
            for (j = 0; j < input_img.width; j++) {
                if (oimg1[i][j] > 255)
                {
                        oimg1[i][j] = 255;
                }
                if (oimg1[i][j] < 0)
                {
                        oimg1[i][j] = 0;
                }
                if (oimg2[i][j] > 255)
                {
                        oimg2[i][j] = 255;
                }
                if (oimg2[i][j] < 0)
                {
                        oimg2[i][j] = 0;
                }
                if (oimg3[i][j] > 255)
                {
                        oimg3[i][j] = 255;
                }
```

```
                        if (oimg3[i][j] < 0)
                        {
                                oimg3[i][j] = 0;
                        }
                        filterd_img.color[0][i][j] = oimg1[i][j];
                        filterd_img.color[1][i][j] = oimg2[i][j];
                        filterd_img.color[2][i][j] = oimg3[i][j];
                }

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

        /* write color image */
        if (write_TIFF(fp, &filterd_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(&(filterd_img));

        free_img((void**)img1);
        free_img((void**)img2);
        free_img((void**)img3);
        free_img((void**)oimg1);
        free_img((void**)oimg2);
        free_img((void**)oimg3);

        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);
}
```

Matlab Code:

```
u = -pi:0.1:pi;
v = -pi:0.1:pi;
[x,y] = meshgrid(u,v);
%%%%%%%%%%%%% lab3.1%%%%%%%%%%
lowPass =
sin(4.5*x)./sin(0.5*x).*sin(4.5*y)./sin(0.5*y)/8
1;
figure(1)

surf(x,y,abs(lowPass));
xlabel('u');
ylabel('v');
zlabel('H(e^(j*u),e^(j*v))')

%%%%%%%%%%%%%lab4.1%%%%%%%%%%

FIR =
sin(2.5*x)./sin(0.5*x).*sin(2.5*y)./sin(0.5*y)/2
5;
figure(2)

surf(x,y,abs(FIR));
xlabel('u');
ylabel('v');
zlabel('H(e^(j*u),e^(j*v))')

%%%%%%%%%%%%%lab4.2%%%%%%%%%%

FIR_1 = 1 + 1.5*(1 -
sin(2.5*x)./sin(0.5*x).*sin(2.5*y)./sin(0.5*y)/2
5);
figure(3)

surf(x,y,abs(FIR_1));
xlabel('u');
```

```
ylabel('v');
zlabel('H(e^(j*u),e^(j*v))')

%%%%%%%%%%%%lab5.1%%%%%%%%

IIR = 0.01./(1-0.9*exp(-1i*x)-0.9*exp(-
1i*y)+0.81*exp(-1i*(x+y)));
figure(4)

surf(x,y,abs(IIR));
xlabel('u');
ylabel('v');
zlabel('H(e^(j*u),e^(j*v))')

%%%%%%%%%%lab5.2 %%%%%%%%5

image = zeros(256,256);
ref = zeros(256,256);
ref(127,127) = 1;
for x = 1:1:256
    for y = 1:1:256
        image(x,y) = 0.01*ref(x,y);
        if(x>1)
            image(x,y) = image(x,y) +
0.9*image(x-1,y);
        end
        if(y>1)
            image(x,y) = image(x,y) +
0.9*image(x,y-1);
        end
        if(x>1&&y>1)
            image(x,y) = image(x,y) -
0.81*image(x-1,y-1);
        end
    end
end
for x = 1:1:256
```

```
    for y = 1:1:256
        if(image(x,y) > 255)
            image(x,y) = 255;
        end
        if(image(x,y) < 0)
            image(x,y) = 0;
        end
    end
end
figure(5)
imwrite(uint8(255*100*image),'h_out.tif')
imshow(uint8(255*100*image))
```