Purdue University, West Lafayette

Spring 2022
ECE 63700 – Digital Image Processing I

Lab-1
Submission Date- 01/21/2022

Submitted by
Nahian Ibn Hasan
PUID: 0032764564
Email: hasan34@purdue.edu

PURDUE
UNIVERSITY®

FIR Low-pass Filter:

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

$$H(u,v) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} h(m,n)\, e^{-j(um+vn)}$$

$$= \frac{1}{81} \sum_{n=-4}^{4} \sum_{m=-4}^{4} e^{-jum} \cdot e^{-jvn}$$

$$= \frac{1}{81} \sum_{n=-4}^{4} e^{-jvn} \left[e^{j4u} + e^{j3u} + e^{j2u} + e^{ju} + 1 + e^{-ju} + e^{-2ju} + e^{-3ju} + e^{-4ju} \right]$$

$$= \frac{1}{81} \sum_{n=-4}^{4} e^{-jvn} \left[1 + 2\cos(4u) + 2\cos(3u) + 2\cos(2u) + 2\cos(u) \right]$$

$$\therefore H(u,v) = \frac{1}{81} \left[1 + 2\cos(4v) + 2\cos(3v) + 2\cos(2v) + 2\cos(v) \right] \cdot \left[1 + 2\cos(4u) + 2\cos(3u) + 2\cos(2u) + 2\cos(u) \right]$$

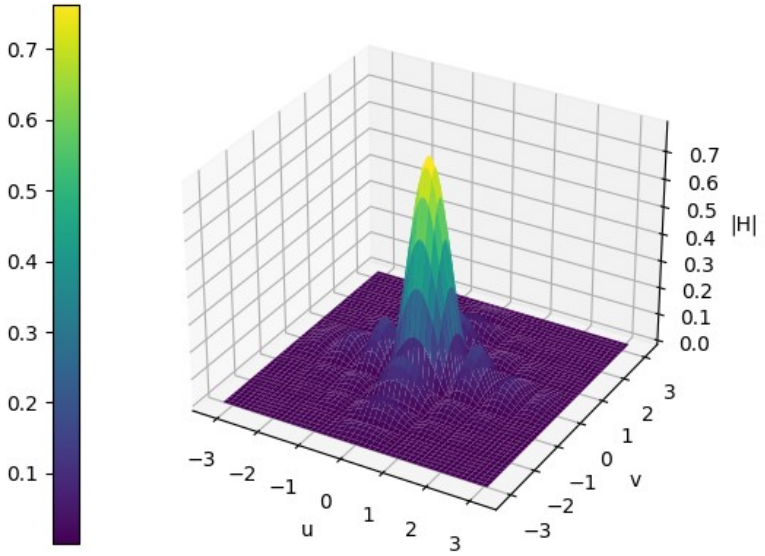
Sharpening Filter:

$$g(m,n) = \delta(m,n) + \lambda \left[\delta(m,n) - h(m,n) \right]$$

$$\boxed{G(u,v) = 1 + \lambda \left(1 - H(u,v) \right)}$$ Ans

Section 3

2. **Plot of FIR Low-Pass Filter:**



3/4. **Color Image and Filtered Image:**

Original Color Image

Filtered Color Image

5. **Code:**

The C code used for this problem is = `FIR_LP_4.c`. The C codes can be found inside the ‘src’ directory. The code is also attached with this report in the Appendix Section. The supplementary python codes for plotting can be found inside the `problems` directory.

FIR Sharpening Filter (Low pass - portion)

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

$$H(u,v) = \sum_{n=\infty}^{\infty} \sum_{m=\infty}^{\infty} h(m,n)\, e^{-j(um+vn)}$$

$$= \frac{1}{25} \sum_{n=-2}^{2} \sum_{m=-2}^{2} e^{-jum} \cdot e^{-jvn}$$

$$= \frac{1}{25} \sum_{n=-2}^{2} e^{-jvn} \left[e^{+j2u} + e^{ju} + 1 + e^{-ju} + e^{-j2u} \right]$$

$$= \frac{1}{25} \sum_{n=-2}^{2} e^{-jvn} \left[1 + 2\cos(2u) + 2\cos(u) \right]$$

$$\therefore H(u,v) = \frac{1}{25} \left[1 + 2\cos(2v) + 2\cos(v) \right] \cdot \left[1 + 2\cos(2u) + 2\cos(u) \right]$$

Sharpening Filter:

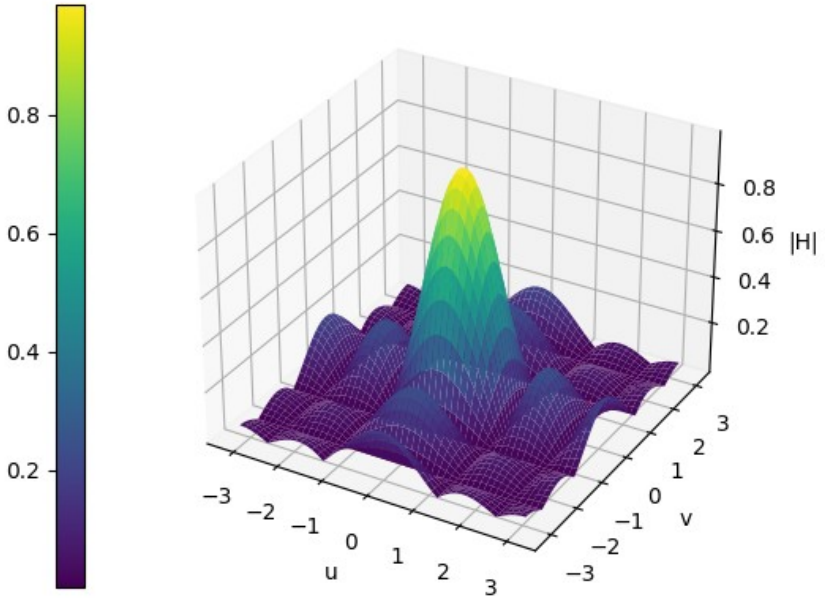
$$g(m,n) = \delta(m,n) + \lambda \left[\delta(m,n) - h(m,n) \right]$$

$$\therefore G(u,v) = 1 + \lambda(1 - H(u,v))$$

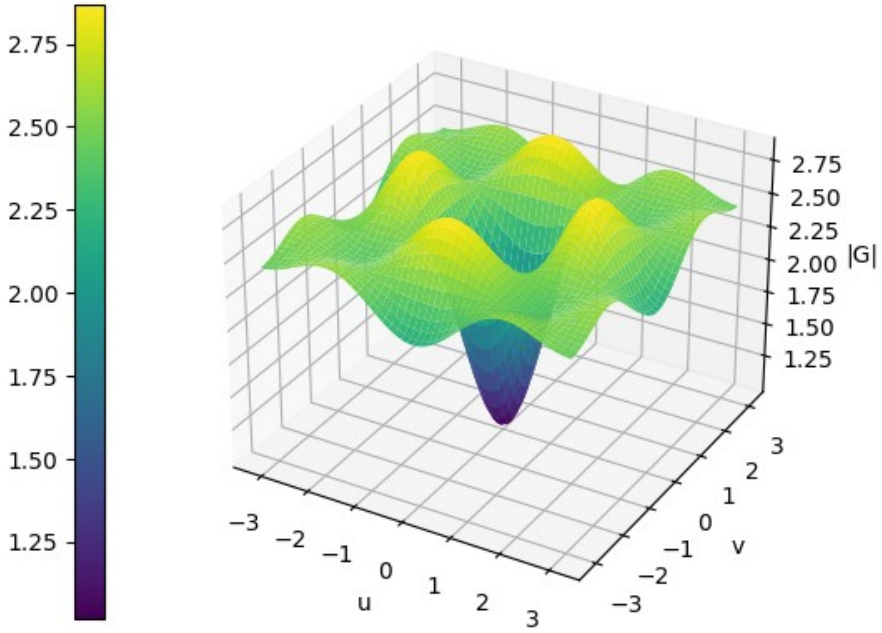
Ans.

Section 4

3. **Plot of FIR Low-Pass Filter (|H|):**



4. **Plot of FIR Sharpening Filter (|G|):**

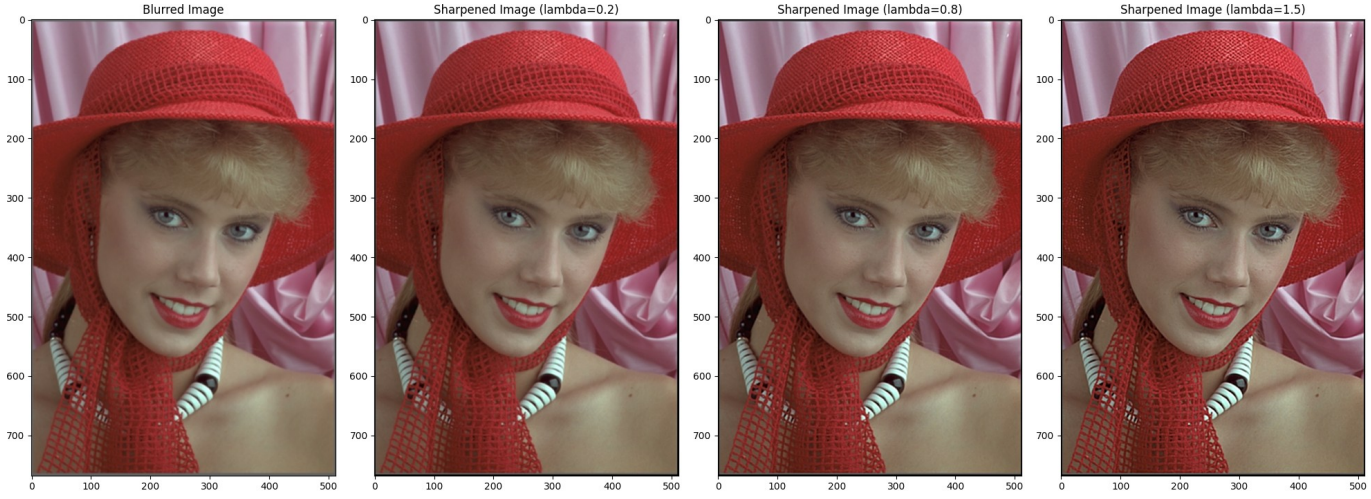


5/6. **Input Color Image and Output Sharpened Image for lambda=1.5:**

Original Blurred Image

Sharpened Image, Lambda=1.5

Comparison among different sharpened image for different lambda:



7.. **C Code:**

The C code used for this problem is = `FIR_Sharp_3.c`. The code can be found inside the `src` directory. The code is also attached with this report in the Appendix Section. The supplementary python codes for plotting can be found inside the `problems` directory.

IIR Filter:

$$Y(m,n) = 0.01\,x(m,n) + 0.9\left[y(m-1,n) + y(m,n-1)\right] - 0.81\,y(m-1,n-1)$$

Transforming in the z domain,

$$Y(z_1,z_2) = 0.01\,X(z_1,z_2) + 0.9\left[Y(z_1,z_2)\,z_1^{-1} + Y(z_1,z_2)\,z_2^{-1}\right] - 0.81\,Y(z_1,z_2)\,z_1^{-1}z_2^{-1}$$

$$\therefore Y(z_1,z_2)\left[1 - 0.9z_1^{-1} - 0.9z_2^{-1} + 0.81\,z_1^{-1}z_2^{-1}\right] = 0.01\,X(z_1,z_2)$$

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

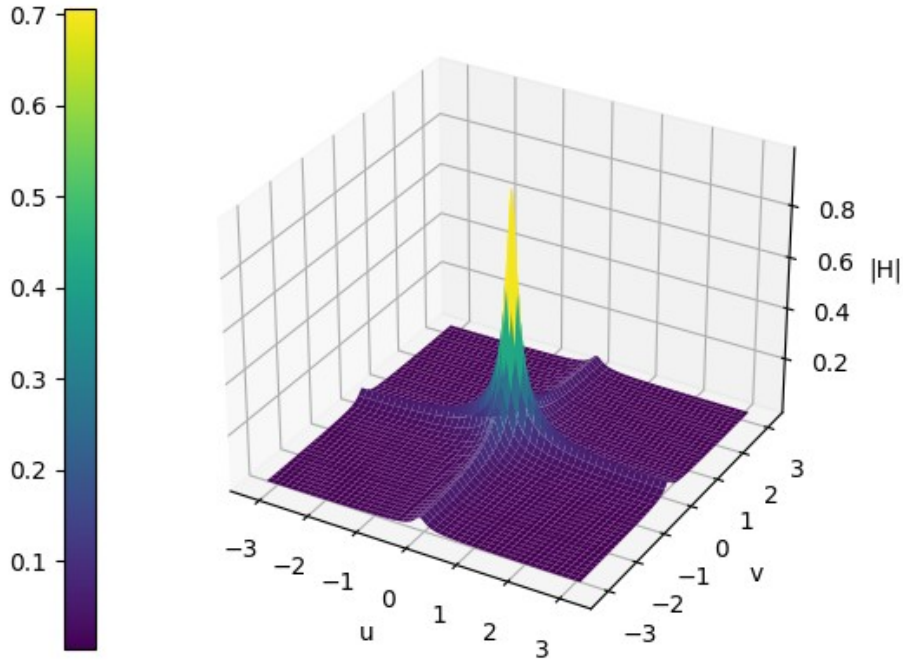
Now in frequency domain, $z_1 = e^{j\omega}$, $z_2 = e^{j\nu}$

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

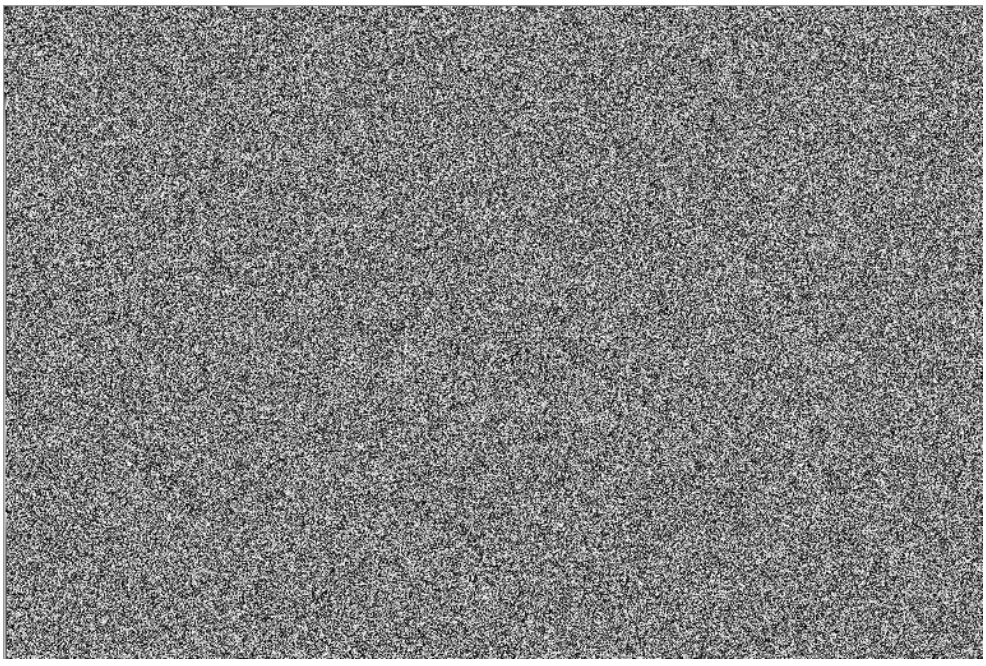
$$\boxed{H(e^{j\omega}, e^{j\nu}) = \frac{0.01}{1 - 0.9\left[e^{-j\omega} + e^{-j\nu}\right] + 0.81\,e^{-j(\omega+\nu)}}}$$

Section 5

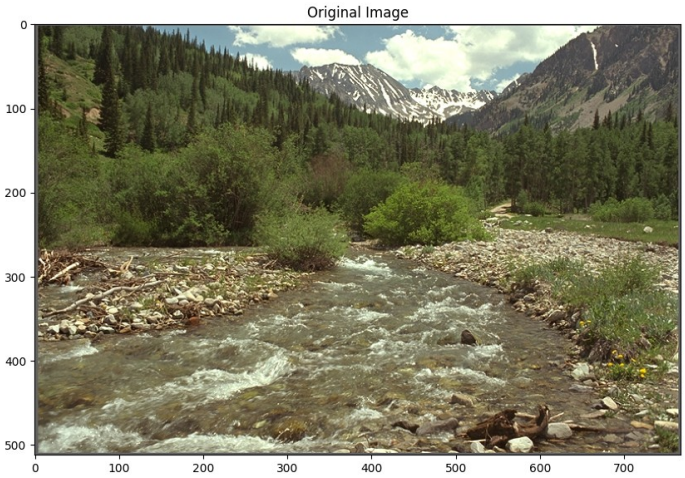
2. **Plot of the IIR Filter magnitude (|H|):**

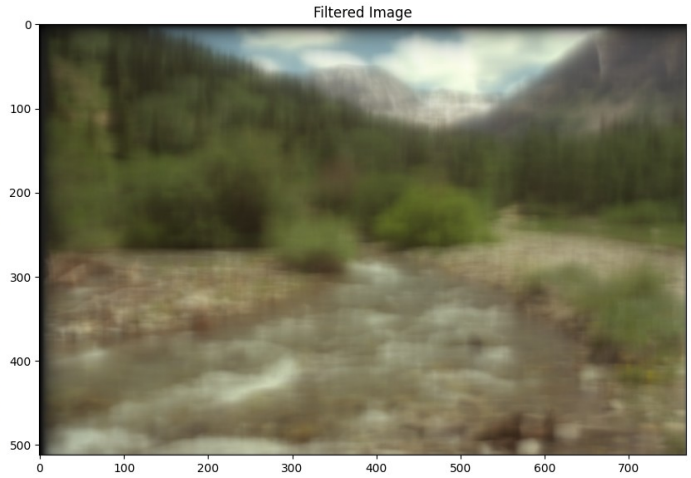


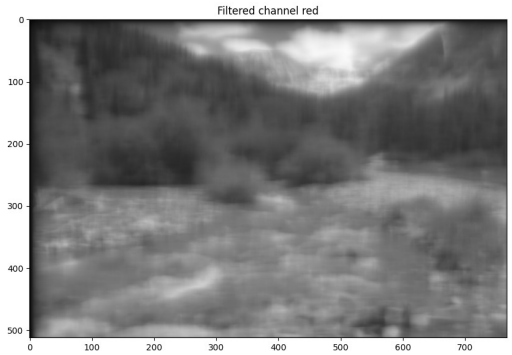
3. **Image of Point Spread Function:**

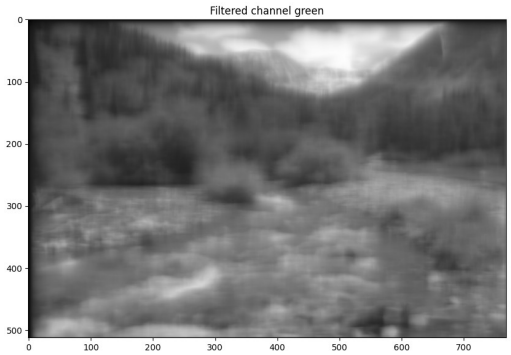


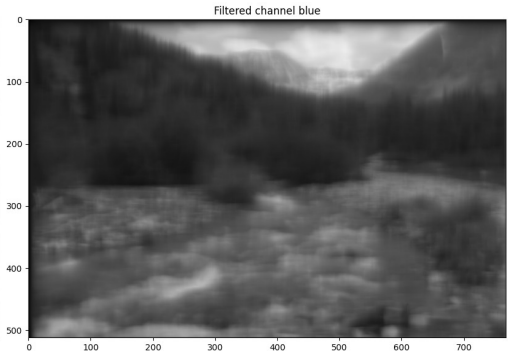
4. **The filtered output color image along with separate channel output:**











5. **C Code:**

The C code used for this problem is = `IIR_3.c`. The code can be found inside the `src` directory. The code is also attached with this report in the Appendix Section. The supplementary python codes for plotting can be found inside the `problems` directory.

Appendix

C Codes

Code of `FIR_LP_4.c`

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

void error(char *name);

int main (int argc, char **argv)
{
        FILE *fp;
        struct TIFF_img input_img, red_img, green_img, blue_img, color_img;
        double **img_r,**img_g,**img_b,**img_2_r,**img_2_g,**img_2_b,temp;
        int32_t i,j,k,l,pixel;

        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 */
        img_r = (double **)get_img(input_img.width,input_img.height,sizeof(double));
        img_2_r = (double **)get_img(input_img.width,input_img.height,sizeof(double));
        fprintf ( stderr,"copy red component to double array\n" );
        for ( i = 0; i < input_img.height; i++ ){
                for ( j = 0; j < input_img.width; j++ ) {
                        img_r[i][j] = input_img.color[0][i][j];
                }
        }
        fprintf ( stderr,"Filter image, red channel\n" );
```

```
for ( i = 4; i < input_img.height-4; i++ ) {
        for ( j = 4; j < input_img.width-4; j++ ) {
                temp = 0;
                for ( k=i-4; k<=i+4; k++ ) {
                        for ( l=j-4; l<=j+4; l++ ) {
                                temp = temp + img_r[k][l];
                        }
                }
                img_2_r[i][j] = temp/81;
        }
}
free_img( (void**)img_r );
fprintf ( stderr,"Fill in boundary pixels -- red channel\n" );
for ( i = 0; i < input_img.height; i++ ) {
        for ( j=0;j<4;j++ ) {
                img_2_r[i][j] = 0;
        }
        for ( j=input_img.width-4;j<=input_img.width-1;j++ ) {
                img_2_r[i][j] = 0;
        }
}
for ( j = 4; j < input_img.width-4; j++ ) {
        for ( i=0;i<4;i++ ) {
                img_2_r[i][j] = 0;
        }
        for ( i=input_img.height-4;i<=input_img.height-1;i++ ) {
                img_2_r[i][j] = 0;
        }
}
/* set up structure for output color image */
/* Note that the type is 'c' rather than 'g' */
get_TIFF ( &red_img, input_img.height, input_img.width, 'g' );
fprintf ( stderr,"copy red component to new images\n" );
for ( i = 0; i < input_img.height; i++ ) {
        for ( j = 0; j < input_img.width; j++ ) {
                pixel = (int32_t)img_2_r[i][j];
                if(pixel>255) {
                        pixel = 255;
                }
                if(pixel<0) {
                        pixel = 0;
                }
                red_img.mono[i][j] = (int32_t)pixel;
        }
}
free_img( (void**)img_2_r );



img_g = (double **)get_img(input_img.width,input_img.height,sizeof(double));
img_2_g = (double **)get_img(input_img.width,input_img.height,sizeof(double));
fprintf ( stderr,"copy green component to double array\n" );
for ( i = 0; i < input_img.height; i++ ){
        for ( j = 0; j < input_img.width; j++ ) {
                img_g[i][j] = input_img.color[1][i][j];
        }
```

```
}
fprintf ( stderr,"Filter image, green channel\n" );
for ( i = 4; i < input_img.height-4; i++ ) {
        for ( j = 4; j < input_img.width-4; j++ ) {
                temp = 0;
                for ( k=i-4; k<=i+4; k++ ) {
                        for ( l=j-4; l<=j+4; l++ ) {
                                temp = temp + img_g[k][l];
                        }
                }
                img_2_g[i][j] = temp/81;
        }
}
free_img( (void**)img_g );
fprintf ( stderr,"Fill in boundary pixels -- green channel\n" );
for ( i = 0; i < input_img.height; i++ ) {
        for ( j=0;j<4;j++ ) {
                img_2_g[i][j] = 0;
        }
        for ( j=input_img.width-4;j<=input_img.width-1;j++ ) {
                img_2_g[i][j] = 0;
        }
}
for ( j = 4; j < input_img.width-4; j++ ) {
        for ( i=0;i<4;i++ ) {
                img_2_g[i][j] = 0;
        }
        for ( i=input_img.height-4;i<=input_img.height-1;i++ ) {
                img_2_g[i][j] = 0;
        }
}
get_TIFF ( &green_img, input_img.height, input_img.width, 'g' );
fprintf ( stderr,"copy green component to new images\n" );
for ( i = 0; i < input_img.height; i++ ){
        for ( j = 0; j < input_img.width; j++ ) {
                pixel = (int32_t)img_2_g[i][j];
                if(pixel>255) {
                        pixel = 255;
                }
                if(pixel<0) {
                        pixel = 0;
                }
                green_img.mono[i][j] = (int32_t)pixel;
        }
}
free_img( (void**)img_2_g );




img_b = (double **)get_img(input_img.width,input_img.height,sizeof(double));
img_2_b = (double **)get_img(input_img.width,input_img.height,sizeof(double));
fprintf ( stderr,"copy blue component to double array\n" );
for ( i = 0; i < input_img.height; i++ ){
```

```
		for ( j = 0; j < input_img.width; j++ ) {
			img_b[i][j] = input_img.color[2][i][j];
		}
}
fprintf ( stderr,"Filter image, blue channel\n" );
for ( i = 4; i < input_img.height-4; i++ ) {
		for ( j = 4; j < input_img.width-4; j++ ) {
			temp = 0;
			for ( k=i-4; k<=i+4; k++ ) {
				for ( l=j-4; l<=j+4; l++ ) {
					temp = temp + img_b[k][l];
				}
			}
			img_2_b[i][j] = temp/81;
		}
}
free_img( (void**)img_b );
fprintf ( stderr,"Fill in boundary pixels -- blue channel\n" );
for ( i = 0; i < input_img.height; i++ ) {
		for ( j=0;j<4;j++ ) {
			img_2_b[i][j] = 0;
		}
		for ( j=input_img.width-4;j<=input_img.width-1;j++ ) {
			img_2_b[i][j] = 0;
		}
}
for ( j = 4; j < input_img.width-4; j++ ) {
		for ( i=0;i<4;i++ ) {
			img_2_b[i][j] = 0;
		}
		for ( i=input_img.height-4;i<=input_img.height-1;i++ ) {
			img_2_b[i][j] = 0;
		}
}
get_TIFF ( &blue_img, input_img.height, input_img.width, 'g' );
fprintf ( stderr,"copy blue component to new images\n" );
for ( i = 0; i < input_img.height; i++ ){
		for ( j = 0; j < input_img.width; j++ ) {
			pixel = (int32_t)img_2_b[i][j];
			if(pixel>255) {
				pixel = 255;
			}
			if(pixel<0) {
				pixel = 0;
			}
			blue_img.mono[i][j] = (int32_t)pixel;
		}
}
free_img( (void**)img_2_b );


get_TIFF ( &color_img, input_img.height, input_img.width, 'c' );
fprintf ( stderr,"constructing filtered color image" );
for ( i = 0; i < input_img.height; i++ ) {
		for ( j = 0; j < input_img.width; j++ ) {
			color_img.color[0][i][j] = red_img.mono[i][j];
```

```
                        color_img.color[1][i][j] = green_img.mono[i][j];
                        color_img.color[2][i][j] = blue_img.mono[i][j];
                }
        }

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

        /* write color image */
        if ( write_TIFF ( fp, &color_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 ( &(red_img) );
        free_TIFF ( &(green_img) );
        free_TIFF ( &(blue_img) );
        free_TIFF ( &(color_img) );



        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);
}
```

Code of `FIR_Sharp_3.c`

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

void error(char *name);

int main (int argc, char **argv)
{
        FILE *fp;
        struct TIFF_img input_img, red_img, green_img, blue_img, color_img;
        double **img_r,**img_g,**img_b,**img_2_r,**img_2_g,**img_2_b,temp,lambda;
        char file_name[50];
        int32_t i,j,k,l,pixel;

        if ( argc != 3 ) 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 );
        }
        fprintf ( stderr, "Lambda =  %s\n", argv[2] );
        sscanf(argv[2],"%lf",&lambda);
        /* Allocate image of double precision floats */
        img_r = (double **)get_img(input_img.width,input_img.height,sizeof(double));
        img_2_r = (double **)get_img(input_img.width,input_img.height,sizeof(double));
        fprintf ( stderr,"copy red component to double array\n" );
        for ( i = 0; i < input_img.height; i++ ){
                for ( j = 0; j < input_img.width; j++ ) {
                        img_r[i][j] = input_img.color[0][i][j];
                }
        }
        fprintf ( stderr,"Filter image, red channel\n" );
        for ( i = 2; i < input_img.height-2; i++ ) {
                for ( j = 2; j < input_img.width-2; j++ ) {
```

```
                temp = 0;
                for ( k=i-2; k<=i+2; k++ ) {
                        for ( l=j-2; l<=j+2; l++ ) {
                                temp = temp + img_r[k][l];
                        }
                }
                temp = img_r[i][j] + lambda*(img_r[i][j] - temp/25);
                img_2_r[i][j] = temp;
        }
}
free_img( (void**)img_r );
fprintf ( stderr,"Fill in boundary pixels -- red channel\n" );
for ( i = 0; i < input_img.height; i++ ) {
        for ( j=0;j<2;j++ ) {
                img_2_r[i][j] = 0;
        }
        for ( j=input_img.width-2;j<=input_img.width-1;j++ ) {
                img_2_r[i][j] = 0;
        }
}
for ( j = 2; j < input_img.width-2; j++ ) {
        for ( i=0;i<2;i++ ) {
                img_2_r[i][j] = 0;
        }
        for ( i=input_img.height-2;i<=input_img.height-1;i++ ) {
                img_2_r[i][j] = 0;
        }
}
/* set up structure for output color image */
/* Note that the type is 'c' rather than 'g' */
get_TIFF ( &red_img, input_img.height, input_img.width, 'g' );
fprintf ( stderr,"copy red component to new images\n" );
for ( i = 0; i < input_img.height; i++ ) {
        for ( j = 0; j < input_img.width; j++ ) {
                pixel = (int32_t)img_2_r[i][j];
                if(pixel>255) {
                        pixel = 255;
                }
                if(pixel<0) {
                        pixel = 0;
                }
                red_img.mono[i][j] = (int32_t)pixel;
        }
}
free_img( (void**)img_2_r );



img_g = (double **)get_img(input_img.width,input_img.height,sizeof(double));
img_2_g = (double **)get_img(input_img.width,input_img.height,sizeof(double));
fprintf ( stderr,"copy green component to double array\n" );
for ( i = 0; i < input_img.height; i++ ){
        for ( j = 0; j < input_img.width; j++ ) {
                img_g[i][j] = input_img.color[1][i][j];
        }
}
```

```
fprintf ( stderr,"Filter image, green channel\n" );
for ( i = 2; i < input_img.height-2; i++ ) {
        for ( j = 2; j < input_img.width-2; j++ ) {
                temp = 0;
                for ( k=i-2; k<=i+2; k++ ) {
                        for ( l=j-2; l<=j+2; l++ ) {
                                temp = temp + img_g[k][l];
                        }
                }
                temp = img_g[i][j] + lambda*(img_g[i][j] - temp/25);
                img_2_g[i][j] = temp;
        }
}
free_img( (void**)img_g );
fprintf ( stderr,"Fill in boundary pixels -- green channel\n" );
for ( i = 0; i < input_img.height; i++ ) {
        for ( j=0;j<2;j++ ) {
                img_2_g[i][j] = 0;
        }
        for ( j=input_img.width-2;j<=input_img.width-1;j++ ) {
                img_2_g[i][j] = 0;
        }
}
for ( j = 2; j < input_img.width-2; j++ ) {
        for ( i=0;i<2;i++ ) {
                img_2_g[i][j] = 0;
        }
        for ( i=input_img.height-2;i<=input_img.height-1;i++ ) {
                img_2_g[i][j] = 0;
        }
}
get_TIFF ( &green_img, input_img.height, input_img.width, 'g' );
fprintf ( stderr,"copy green component to new images\n" );
for ( i = 0; i < input_img.height; i++ ){
        for ( j = 0; j < input_img.width; j++ ) {
                pixel = (int32_t)img_2_g[i][j];
                if(pixel>255) {
                        pixel = 255;
                }
                if(pixel<0) {
                        pixel = 0;
                }
                green_img.mono[i][j] = (int32_t)pixel;
        }
}
free_img( (void**)img_2_g );




img_b = (double **)get_img(input_img.width,input_img.height,sizeof(double));
img_2_b = (double **)get_img(input_img.width,input_img.height,sizeof(double));
fprintf ( stderr,"copy blue component to double array\n" );
for ( i = 0; i < input_img.height; i++ ){
```

```
		for ( j = 0; j < input_img.width; j++ ) {
			img_b[i][j] = input_img.color[2][i][j];
		}
}
fprintf ( stderr,"Filter image, blue channel\n" );
for ( i = 2; i < input_img.height-2; i++ ) {
		for ( j = 2; j < input_img.width-2; j++ ) {
			temp = 0;
			for ( k=i-2; k<=i+2; k++ ) {
				for ( l=j-2; l<=j+2; l++ ) {
					temp = temp + img_b[k][l];
				}
			}
			temp = img_b[i][j] + lambda*(img_b[i][j] - temp/25);
			img_2_b[i][j] = temp;
		}
}
free_img( (void**)img_b );
fprintf ( stderr,"Fill in boundary pixels -- blue channel\n" );
for ( i = 0; i < input_img.height; i++ ) {
		for ( j=0;j<2;j++ ) {
			img_2_b[i][j] = 0;
		}
		for ( j=input_img.width-2;j<=input_img.width-1;j++ ) {
			img_2_b[i][j] = 0;
		}
}
for ( j = 2; j < input_img.width-2; j++ ) {
		for ( i=0;i<2;i++ ) {
			img_2_b[i][j] = 0;
		}
		for ( i=input_img.height-2;i<=input_img.height-1;i++ ) {
			img_2_b[i][j] = 0;
		}
}
get_TIFF ( &blue_img, input_img.height, input_img.width, 'g' );
fprintf ( stderr,"copy blue component to new images\n" );
for ( i = 0; i < input_img.height; i++ ){
		for ( j = 0; j < input_img.width; j++ ) {
			pixel = (int32_t)img_2_b[i][j];
			if(pixel>255) {
				pixel = 255;
			}
			if(pixel<0) {
				pixel = 0;
			}
			blue_img.mono[i][j] = (int32_t)pixel;
		}
}
free_img( (void**)img_2_b );


get_TIFF ( &color_img, input_img.height, input_img.width, 'c' );
fprintf ( stderr,"constructing filtered color image" );
for ( i = 0; i < input_img.height; i++ ) {
		for ( j = 0; j < input_img.width; j++ ) {
```

```
                        color_img.color[0][i][j] = red_img.mono[i][j];
                        color_img.color[1][i][j] = green_img.mono[i][j];
                        color_img.color[2][i][j] = blue_img.mono[i][j];
                }
        }

        /* open color image file */
        strcpy(file_name,"sharpened_image_");
        strcat(file_name,argv[2]);
        strcat(file_name,".tif");
        if ( ( fp = fopen ( file_name, "wb" ) ) == NULL ) {
                fprintf ( stderr, "cannot open file sharpened_image.tif\n");
                exit ( 1 );
        }

        /* write color image */
        if ( write_TIFF ( fp, &color_img ) ) {
                fprintf ( stderr, "error writing TIFF file %s\n", argv[3] );
                exit ( 1 );
        }

        /* close color image file */
        fclose ( fp );

        /* de-allocate space which was used for the images */
        free_TIFF ( &(input_img) );
        free_TIFF ( &(red_img) );
        free_TIFF ( &(green_img) );
        free_TIFF ( &(blue_img) );
        free_TIFF ( &(color_img) );



        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);
}
```

Code of `IIR_3.c`

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

void error(char *name);

int main (int argc, char **argv)
{
        FILE *fp;
        struct TIFF_img input_img, red_img, green_img, blue_img, color_img;
        double **img_r,**img_g,**img_b,**img_2_r,**img_2_g,**img_2_b;
        int32_t i,j,pixel;

        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 */
        img_r = (double **)get_img(input_img.width,input_img.height,sizeof(double));
        img_2_r = (double **)get_img(input_img.width,input_img.height,sizeof(double));
        fprintf ( stderr,"copy red component to double array\n" );
        for ( i = 0; i < input_img.height; i++ ){
                for ( j = 0; j < input_img.width; j++ ) {
                        img_r[i][j] = input_img.color[0][i][j];
                }
        }
        fprintf ( stderr,"Fill in boundary pixels -- red channel\n" );
        for ( i = 0; i < input_img.height; i++ ) {
                img_2_r[i][0] = 0;
                img_2_r[i][input_img.width-1] = 0;
        }
```

```
	for ( j = 1; j < input_img.width-1; j++ ) {
		img_2_r[0][j] = 0;
		img_2_r[input_img.height-1][j] = 0;
	}
	fprintf ( stderr,"Filter image, red channel\n" );
	for ( i = 2; i <= input_img.height-1; i++ ) {
		for ( j = 2; j <= input_img.width-1; j++ ) {
			img_2_r[i][j] = 0.01*img_r[i][j]+0.9*(img_2_r[i-1][j]+img_2_r[i][j-1])-0.81*img_2_r[i-1][j-
1];
		}
	}
	free_img( (void**)img_r );
	/* set up structure for output color image */
	/* Note that the type is 'c' rather than 'g' */
	get_TIFF ( &red_img, input_img.height, input_img.width, 'g' );
	fprintf ( stderr,"copy red component to new images\n" );
	for ( i = 0; i < input_img.height; i++ ) {
		for ( j = 0; j < input_img.width; j++ ) {
			pixel = (int32_t)img_2_r[i][j];
			if(pixel>255) {
				pixel = 255;
			}
			if(pixel<0) {
				pixel = 0;
			}
			red_img.mono[i][j] = (int32_t)pixel;
		}
	}
	free_img( (void**)img_2_r );



	img_g = (double **)get_img(input_img.width,input_img.height,sizeof(double));
	img_2_g = (double **)get_img(input_img.width,input_img.height,sizeof(double));
	fprintf ( stderr,"copy green component to double array\n" );
	for ( i = 0; i < input_img.height; i++ ){
		for ( j = 0; j < input_img.width; j++ ) {
			img_g[i][j] = input_img.color[1][i][j];
		}
	}
	fprintf ( stderr,"Fill in boundary pixels -- green channel\n" );
	for ( i = 0; i < input_img.height; i++ ) {
		img_2_g[i][0] = 0;
		img_2_g[i][input_img.width-1] = 0;
	}
	for ( j = 1; j < input_img.width-1; j++ ) {
		img_2_g[0][j] = 0;
		img_2_g[input_img.height-1][j] = 0;
	}
	fprintf ( stderr,"Filter image, green channel\n" );
	for ( i = 2; i <= input_img.height-1; i++ ) {
		for ( j = 2; j <= input_img.width-1; j++ ) {
			img_2_g[i][j] = 0.01*img_g[i][j]+0.9*(img_2_g[i-1][j]+img_2_g[i][j-1])-0.81*img_2_g[i-1]
[j-1];
		}
	}
```

```
	free_img( (void**)img_g );
	get_TIFF ( &green_img, input_img.height, input_img.width, 'g' );
	fprintf ( stderr,"copy green component to new images\n" );
	for ( i = 0; i < input_img.height; i++ ){
		for ( j = 0; j < input_img.width; j++ ) {
			pixel = (int32_t)img_2_g[i][j];
			if(pixel>255) {
				pixel = 255;
			}
			if(pixel<0) {
				pixel = 0;
			}
			green_img.mono[i][j] = (int32_t)pixel;
		}
	}
	free_img( (void**)img_2_g );



	img_b = (double **)get_img(input_img.width,input_img.height,sizeof(double));
	img_2_b = (double **)get_img(input_img.width,input_img.height,sizeof(double));
	fprintf ( stderr,"copy blue component to double array\n" );
	for ( i = 0; i < input_img.height; i++ ){
		for ( j = 0; j < input_img.width; j++ ) {
			img_b[i][j] = input_img.color[2][i][j];
		}
	}
	fprintf ( stderr,"Fill in boundary pixels -- blue channel\n" );
	for ( i = 0; i < input_img.height; i++ ) {
		img_2_b[i][0] = 0;
		img_2_b[i][input_img.width-1] = 0;
	}
	for ( j = 1; j < input_img.width-1; j++ ) {
		img_2_b[0][j] = 0;
		img_2_b[input_img.height-1][j] = 0;
	}
	fprintf ( stderr,"Filter image, blue channel\n" );
	for ( i = 2; i <= input_img.height-1; i++ ) {
		for ( j = 2; j <= input_img.width-1; j++ ) {
			img_2_b[i][j] = 0.01*img_b[i][j]+0.9*(img_2_b[i-1][j]+img_2_b[i][j-1])-0.81*img_2_b[i-1]
[j-1];
		}
	}
	free_img( (void**)img_b );
	get_TIFF ( &blue_img, input_img.height, input_img.width, 'g' );
	fprintf ( stderr,"copy blue component to new images\n" );
	for ( i = 0; i < input_img.height; i++ ){
		for ( j = 0; j < input_img.width; j++ ) {
			pixel = (int32_t)img_2_b[i][j];
			if(pixel>255) {
				pixel = 255;
			}
			if(pixel<0) {
				pixel = 0;
			}
			blue_img.mono[i][j] = (int32_t)pixel;
```

```
                }
        }
        free_img( (void**)img_2_b );


        get_TIFF ( &color_img, input_img.height, input_img.width, 'c' );
        fprintf ( stderr,"constructing filtered color image" );
        for ( i = 0; i < input_img.height; i++ ) {
                for ( j = 0; j < input_img.width; j++ ) {
                        color_img.color[0][i][j] = red_img.mono[i][j];
                        color_img.color[1][i][j] = green_img.mono[i][j];
                        color_img.color[2][i][j] = blue_img.mono[i][j];
                }
        }

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

        /* write color image */
        if ( write_TIFF ( fp, &color_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 ( &(red_img) );
        free_TIFF ( &(green_img) );
        free_TIFF ( &(blue_img) );
        free_TIFF ( &(color_img) );



        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);
}
```