Project 3: Image Enhancement - Spatial vs. Frequency Domain Filters

Steven Young: ECE 572

Due: October 3, 2011

98+10

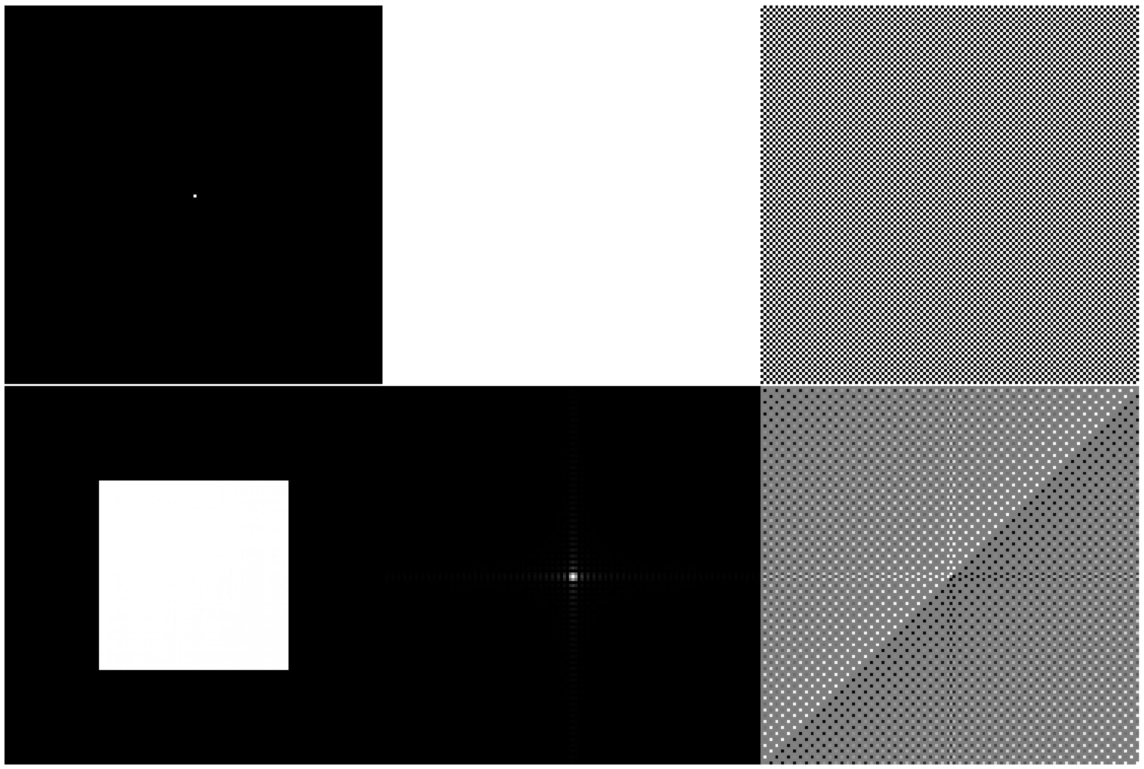
Abstract

The purpose of this project is to explore some simple image enhancement algorithms. This project introduces spatial and frequency domain filters. Thus it involves creating masks, performing convolution, Fourier transforms and inverse Fourier transforms. An in-depth understanding of the Fourier transform is critical to the understanding of this project.

Discussion

Task 1(10 points)

Below you can see the FFT transforms of a single pixel and of a large square on a 128x128 image.



use logtran to show mag image. -1

Figure 1: Original Image, FFT Magnitude and FFT Phase

As you can see, taking the FFT of a single pixel results in a constant magnitude with a phase that alternates between two values in a checkerboard pattern. Taking the FFT of a large square results in an FFT magnitude that is a 2D sinc function.

Below you can see the FFT transforms of some vertical lines on a 128x128 image.

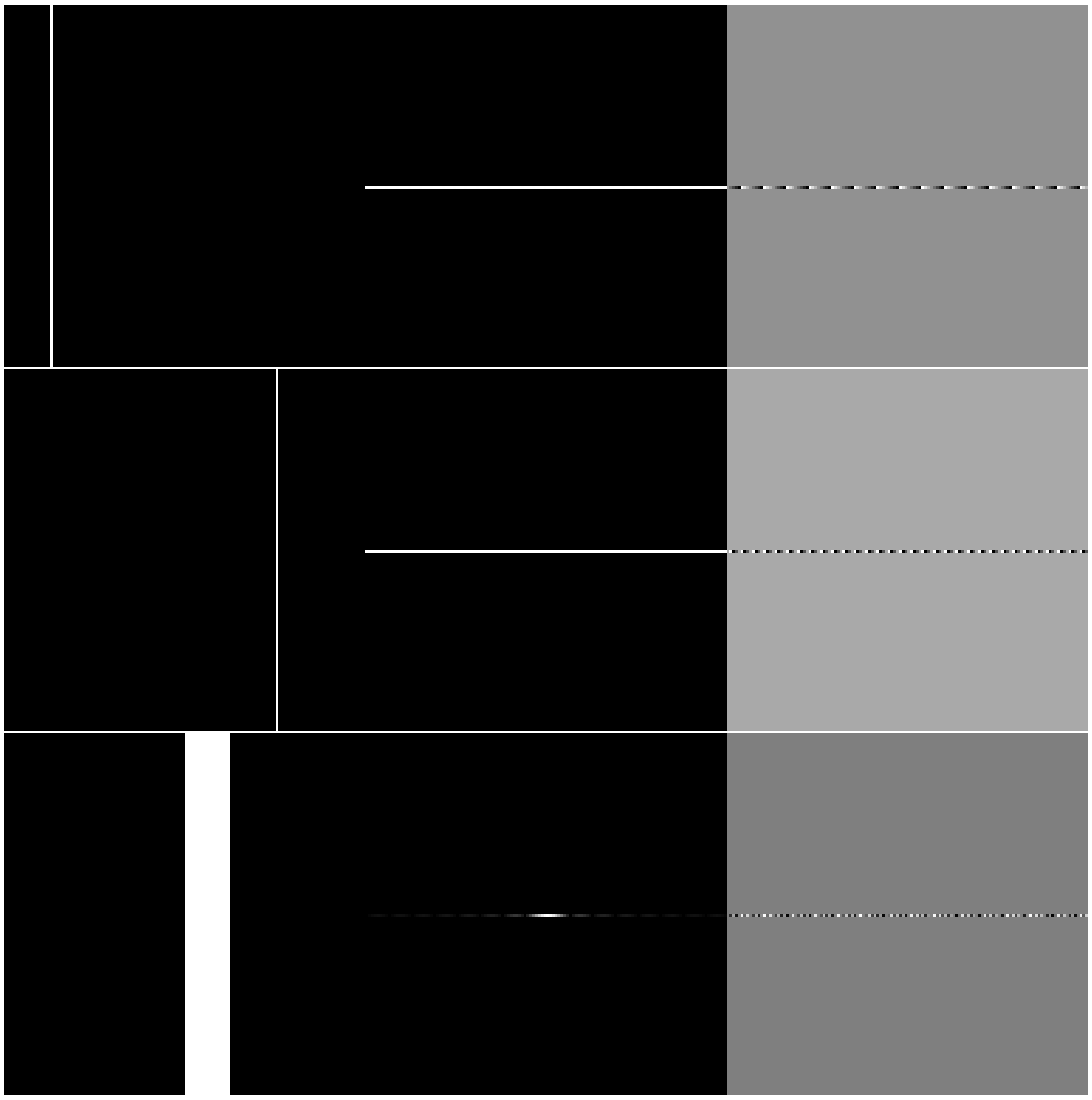


Figure 2: Original Image, FFT Magnitude and FFT Phase

As you can see, shifting the location of the line does not effect the FFT magnitude with only some slight changes in the phase. Changing the width of the line from 1 to something greater than 1 results in a one-dimensional sinc function instead of a constant magnitude line.

With the single pixel image, we expected a constant magnitude and a constant phase of 0. We expected this because this is the FFT of an impulse. However, because of digitization, the phase is not constant 0.

✓

Task 2 (30 points)

For this task we will be working with the following image.

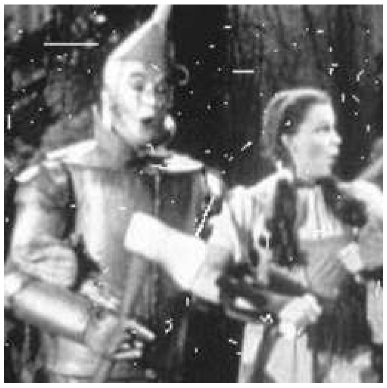
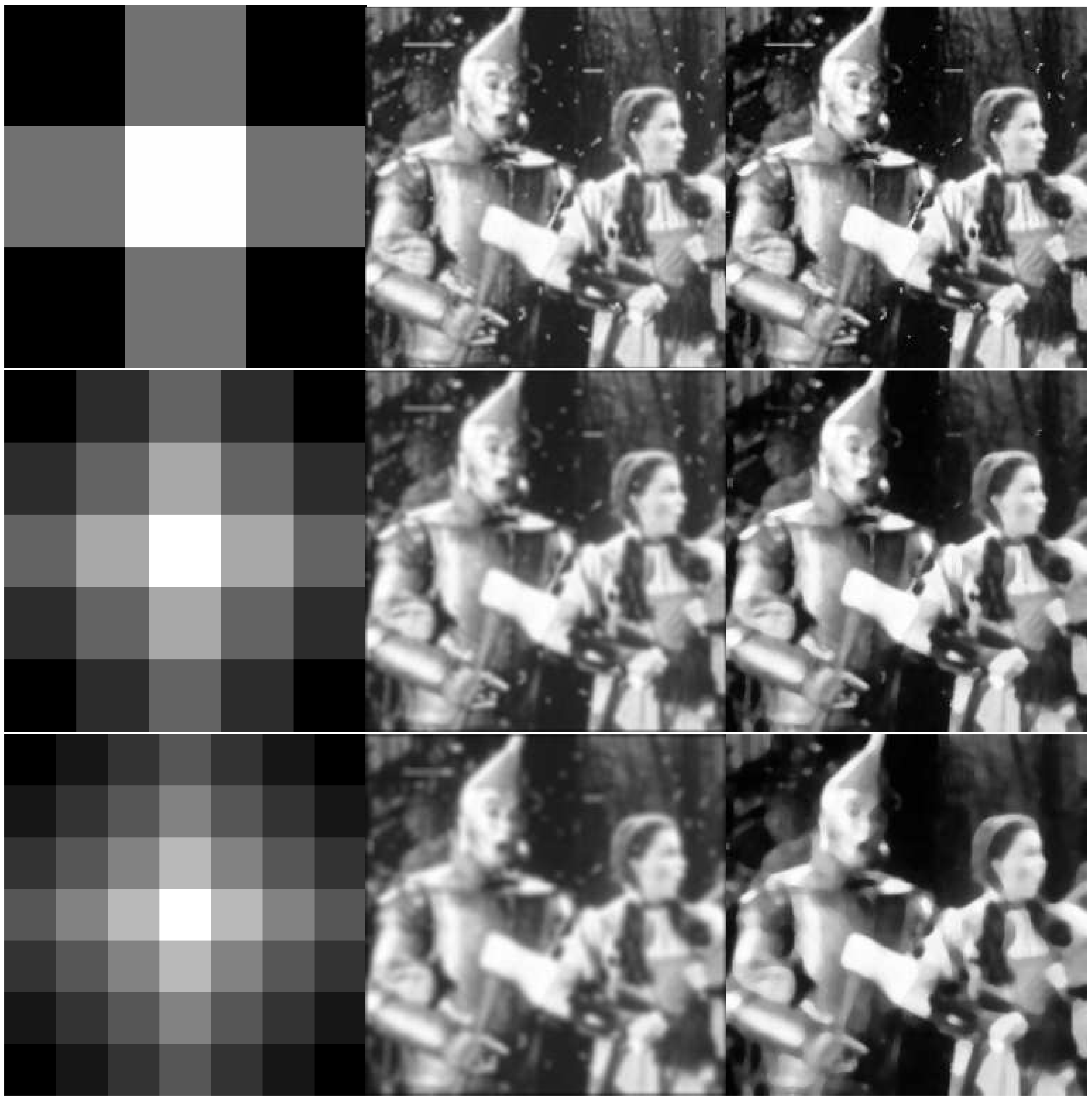


Figure 3: Original Image

Below are the 3x3, 5x5, and 7x7 gaussian kernels with sigma = 1.5



need to show kernel values -1

Figure 4: Guassian Kernel, Gaussian Filtered Image, Median Filtered Image (3x3, 5x5, 7x7)

Subjectively, the Median filter seems to outperform the Gaussian filter with all kernel sizes. The Gaussian filter seems to make the picture fuzzy, and the Median filter seems to make the picture seem unnatural or cartoonish. Subjectively, both methods appear to remove the most artifacts with the smallest degradation with the 5x5 kernel. The Gaussian filter is linear, and thus it becomes fuzzier as the kernel size increases. The Median filter is nonlinear, and thus it does not become fuzzier as the kernel size increases. However, it does become more “cartoonish” looking when the kernel size increases.

Decreasing the cutoff frequency, D_0, in the frequency filter is similar to increasing the size of

✓

the mask with the spatial filter. Below are some spatial and frequency domain comparisons.

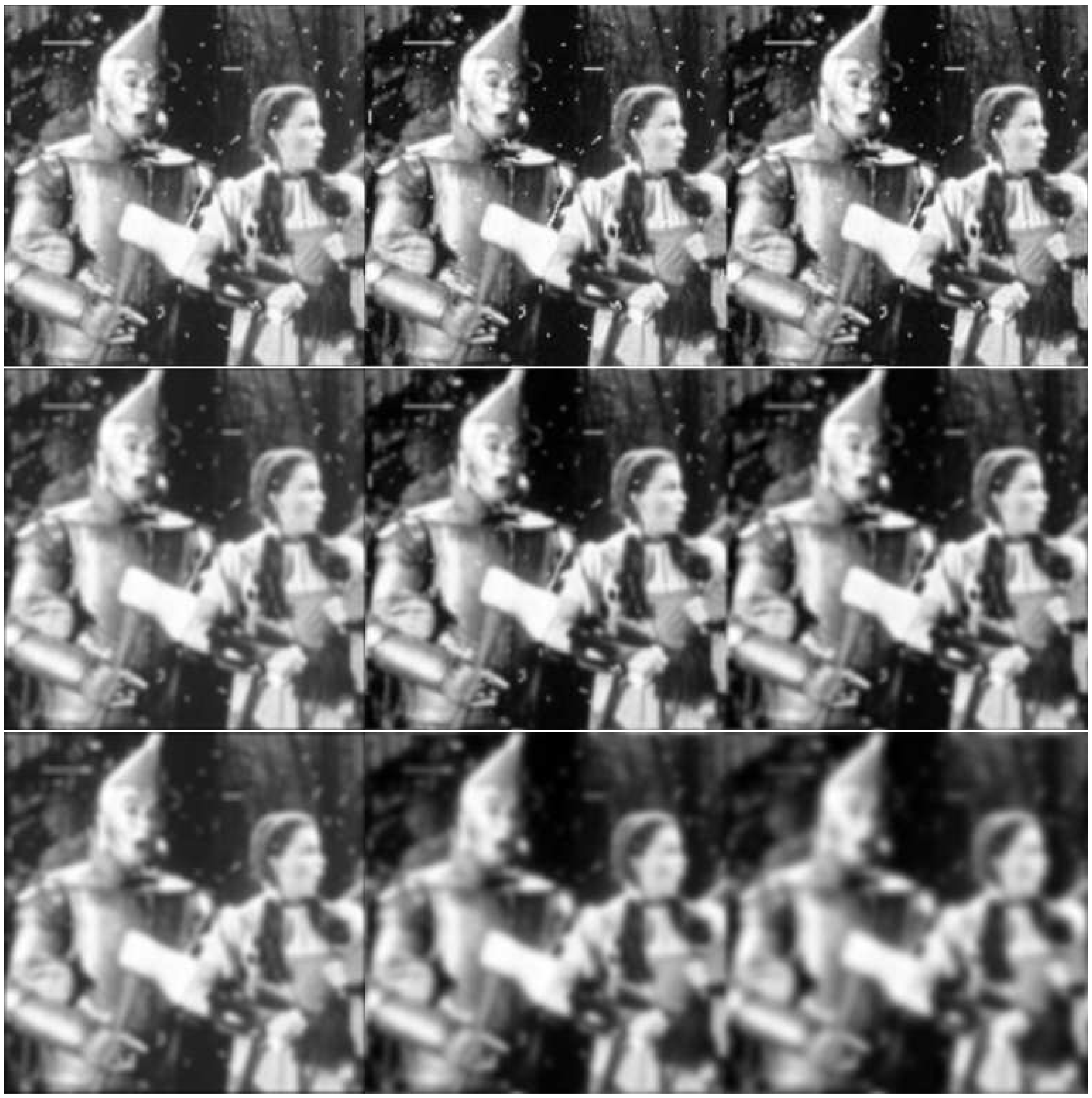


Figure 5: Spatial Guassian Filtered Image, Frequency Gaussian Filtered Image, Frequency Butterworth Filter (order =2)(3x3, D_0 = 64; 5x5, D_0 = 32; 7x7, D_0 = 16)

✓

D_0	Gaussian	Butterworth
64	99.0953	99.4303
32	97.6749	98.1878
16	94.9893	95.5876

Table 1: LP Power Ratios

Based on Tasks 2.1 and 2.2, I believe the same results can be acheived from spatial and frequency filters. However, linear and nonlinear filters cannot be interchanged to acheive the same results.

Task 3 (30 points)

For this task we will be working with the following image.

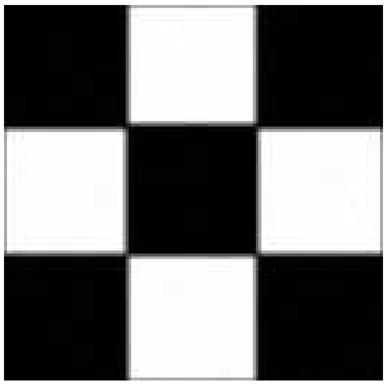


Figure 6: Original Image

Laplaciang and Sobel edge detection results are shown below.

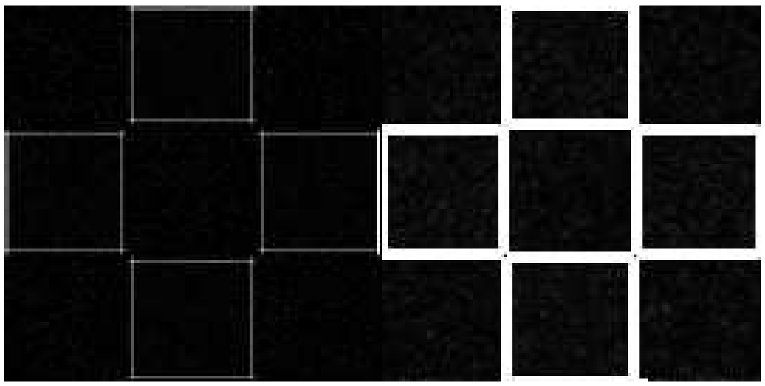


Figure 7: Laplacian and Sobel Edge Detection Results

The Laplacian edge detector seems to have resulted in more precise edges, whereas the Sobel edge detector seems to have wider “edges.” The Laplacian edge detector also only requires one convolution operation while the Sobel edge detector takes two convolution operations.

Below is an unsharp mask with 3x3 and 9x9 kernels respectively.

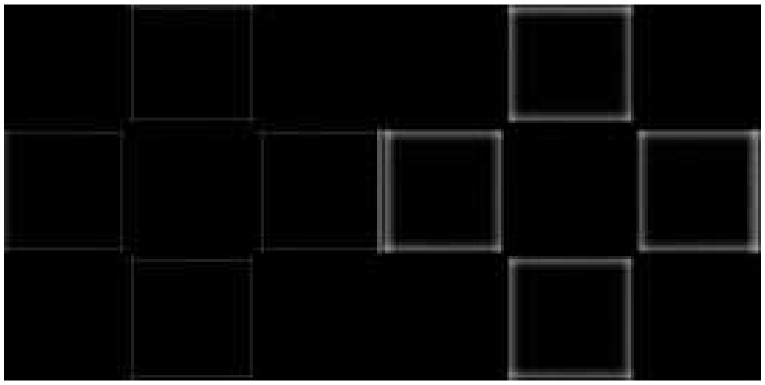


Figure 8: Unsharp Mask (3x3 and 9x9)

As you can see, the larger the kernel the wider this edge becomes. While this make it easier for the human eye to view, it results in a less precise edge.

Below are the Butterworth and Gaussian High Pass Filter results. Butterworth is in the left column and the cutoff frequency, D_0, increases as you go down the column.

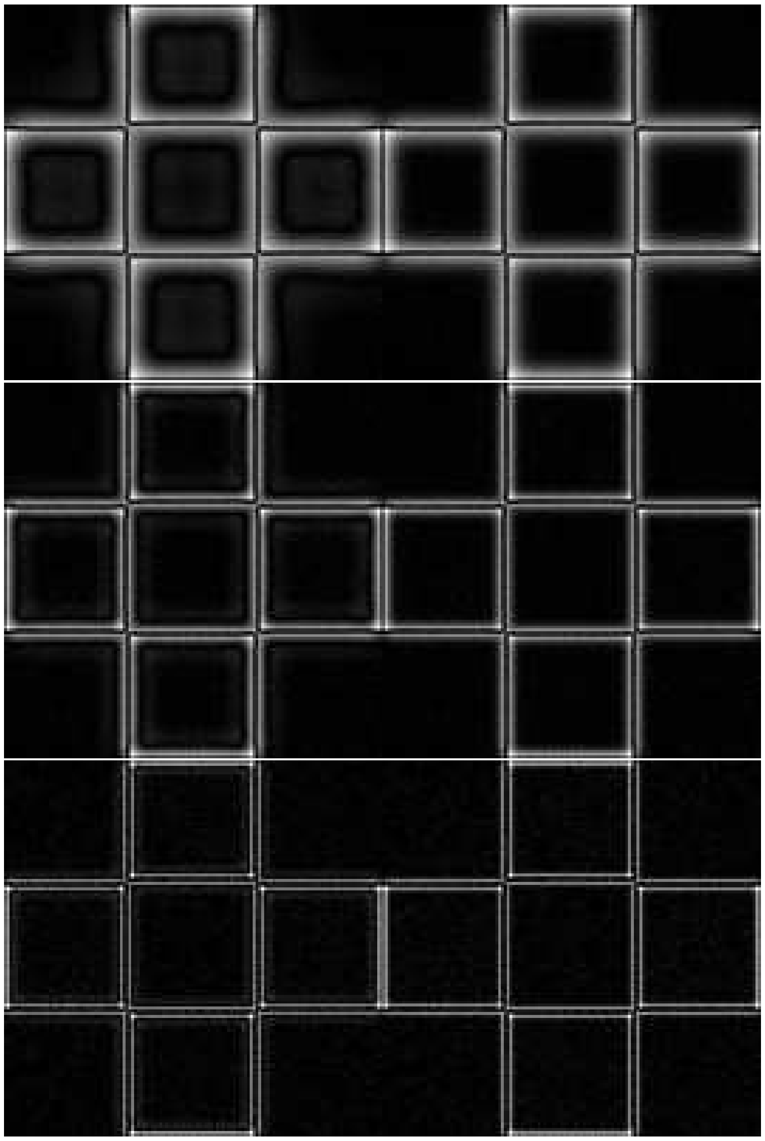


Figure 9: Two Sets w/ Three Kernel Sizes: Butterworth(D_0=(10,20,40), order =2) and Gaussian Edge Detection Results(D_0 = 10,20,40)

As you can see these both produce similar results. The Butterworth seems to have some additional artifacts. The power ratios for these high pass filters are shown below. They are small because the high-frequency content, the edges, are a small portion of the image.

✓

D_0	Butterworth	Gaussian
40	0.536922	0.44061
20	2.38941	1.92622
10	7.11757	5.96034

Table 2: HP Power Ratios

Task 4 (30 points)

The image used for this task is given below.

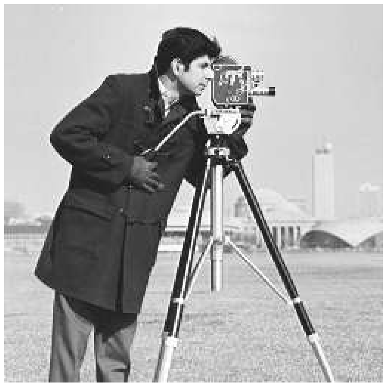


Figure 10: Original Image

Below are the results from the spatial domain and the frequency domian along with their difference image.

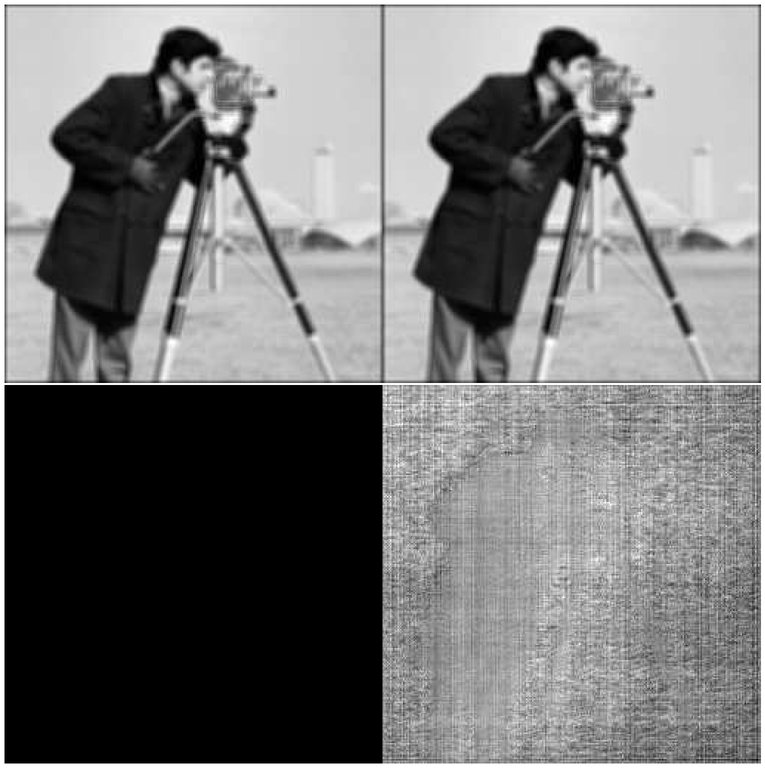


Figure 11: Spatial Average, Frequency Average, Difference, Difference Rescaled

As, you can see the results are **very** similar. The difference image shows that there is some difference, but it is insignificant. Also, since the mask is small, the computation time is shorter for the spatial implementation.

	Spatial	Freq
secs	0.08	1.22

Table 3: Computation Times

Below are the autocorrelation images computed both spatially and in the frequency domain.

✓

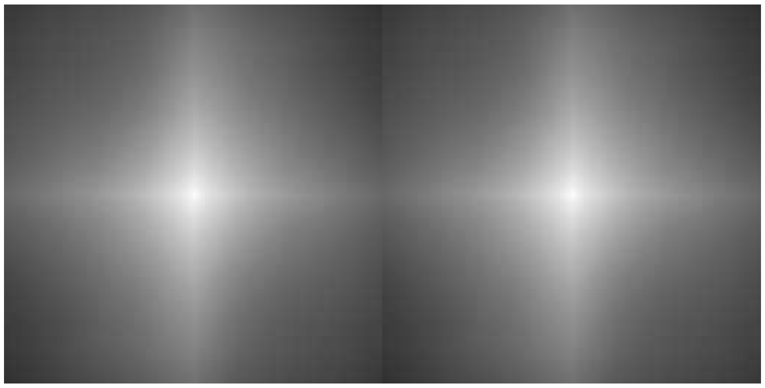


Figure 12: Spatial Autocorrleation and Frequency Autocorrelation

As you can see, the same result is acheived with both methods. However, the frequency method is much faster because of the size of the convolution.

	Spatial	Freq
secs	114.7	0.79

✓

Table 4: Computation Times

Extra Credit (10 points)

For this task, the following image was used to demonstrate Homomorphic filtering.

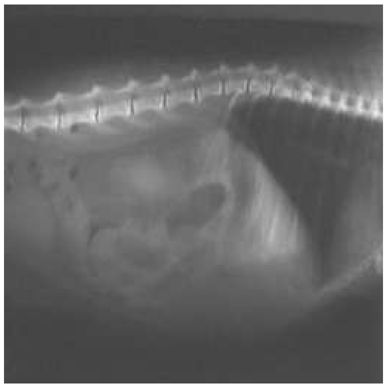
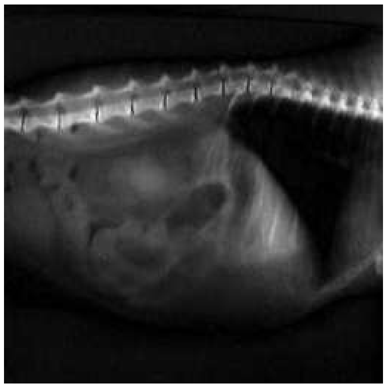


Figure 13: Original Image

First, parameters had to be selected. The cutoff frequency, D_0, was set to 2. Gamma_H was set to 100 and Gamma_L was set to 20. The order of the filter, c, was set to 10. The gamma values are used to truncate the histogram. The order parameter set the shape of the filter. The cutoff parameter sets the minimum frequency. The filtered image is shown below. You can see many more features in the image.



+10

Figure 14: Homomorphic Filtered Image

Discussion

This project reinforced the topics discussed in class. It broadened the students knowledge of the image enhancement techniques beyond what can be gained just by discussion or reading. It is especially important to understand the Fourier transform and its effects upon computation time and how to design filters to accomplish a task.

```cpp
/********************************************************************
 * task1.cpp - Implementation file for task1
 *
 * Author: Steven Young (syoung22@utk.edu), EECS
 *         University of Tennessee
 *
 * Class: ECE 572 Fall 2011
 *
 * Created: 10/01/2011
 *
 * Note:
 *
 ********************************************************************/
#include "square.h"
#include "line.h"
#include "Image.h"
#include "Dip.h"
#include <iostream>
#include <cstdlib>
#include <cmath>             // new: for log()
using namespace std;

#define Usage "./task1 ROWS COLS SQUARE_SIZE LINE_LOC LINE_WIDTH\n"

int main(int argc, char **argv)
{
  Image inimg, outimg, mag, phase, maglog;
  int nr, nc, size_sq, line_loc, line_wid;
  int i, j;

  if (argc < 6) {
    cout << Usage;
    exit(3);
  }

  // Convert Arugments to Integers
  nr = atoi(argv[1]);
  nc = atoi(argv[2]);
  size_sq = atoi(argv[3]);
  line_loc = atoi(argv[4]);
  line_wid = atoi(argv[5]);

  // read in image
  inimg = square(nr, nc, size_sq);
  nr = inimg.getRow();
  nc = inimg.getCol();

  // allocate memory
  mag.createImage(nr, nc);
  phase.createImage(nr, nc);
  outimg.createImage(nr, nc);

  // Fourier transform
  fft(inimg, mag, phase);

  // magnitude after log transformation
  //  maglog = logtran(mag);
  maglog = inimg;           // new: allocate memory
  for (i=0; i<nr; i++)      // new: perform log transformation to compress the dyna
mic range
    for (j=0; j<nc; j++)    // new: you should be able to call the logtran() you de
signed in proj2
      maglog(i,j) = log(1+fabs(mag(i,j)));

  // inverse Fourier transform
  ifft(outimg, mag, phase);

  // output image
  cout << "Write magnitude image ..." << endl;
  writeImage(mag, "../output/squaretestmag.pgm", 1);
  cout << "Write magnitude image AFTER log transformation ..." << endl;
  writeImage(maglog, "../output/squaretestmaglog.pgm", 1);
  cout << "Write phase image ..." << endl;
  writeImage(phase, "../output/squaretestphase.pgm", 1);
  cout << "Write image after inverse Fourier transform ..." << endl;
  writeImage(outimg, "../output/squaretestifft.pgm");

  // read in image
  inimg = line(nr, nc, line_loc, line_wid);
  nr = inimg.getRow();
  nc = inimg.getCol();

  // Fourier transform
  fft(inimg, mag, phase);

  // magnitude after log transformation
  //  maglog = logtran(mag);
  maglog = inimg;           // new: allocate memory
  for (i=0; i<nr; i++)      // new: perform log transformation to compress the dyna
mic range
    for (j=0; j<nc; j++)    // new: you should be able to call the logtran() you de
signed in proj2
      maglog(i,j) = log(1+fabs(mag(i,j)));

  // inverse Fourier transform
  ifft(outimg, mag, phase);

  // output image
  cout << "Write magnitude image ..." << endl;
  writeImage(mag, "../output/linetestmag.pgm", 1);
  cout << "Write magnitude image AFTER log transformation ..." << endl;
  writeImage(maglog, "../output/linetestmaglog.pgm", 1);
  cout << "Write phase image ..." << endl;
  writeImage(phase, "../output/linetestphase.pgm", 1);
  cout << "Write image after inverse Fourier transform ..." << endl;
  writeImage(outimg, "../output/linetestifft.pgm");


  return 0;
}
```

```
/*******************************************************************
 * task2.cpp - Implementation file for task2_1
 *
 * Author: Steven Young (syoung22@utk.edu), EECS
 *         University of Tennessee
 *
 * Class: ECE 572 Fall 2011
 *
 * Created: 10/01/2011
 *
 * Note:
 *
 ********************************************************************/
#include "spatialFilter.h"
#include "Image.h"
#include "Dip.h"
#include <iostream>
#include <cstdlib>
#include <cmath>             // new: for log()
using namespace std;

#define Usage "./task2 INPUT_IMAGE N OUTPUT_IMAGE N_median OUTPUT_median\n"

int main(int argc, char **argv)
{
  Image inimg, outimg;
  int n, n_m;

  if (argc < 6) {
    cout << Usage;
    exit(3);
  }

  // Convert Arugments to Integers
  n = atoi(argv[2]);
  n_m = atoi(argv[4]);

  // read in image
  inimg = readImage(argv[1]);

  // filter image
  outimg = gaussianFilter(inimg, n);

  // output image
  writeImage(outimg, argv[3], 1);

  // median filter
  outimg = medianFilter(inimg, n_m);

  // output median image
  writeImage(outimg, argv[5], 1);

  return 0;
}
```

```
/*********************************************************************
 * task2_2.cpp - Implementation file for task2_2
 *
 * Author: Steven Young (syoung22@utk.edu), EECS
 *         University of Tennessee
 *
 * Class: ECE 572 Fall 2011
 *
 * Created: 10/01/2011
 *
 * Note:
 *
 *********************************************************************/
#include "freqFilter.h"
#include "Image.h"
#include "Dip.h"
//#include <iostream>
#include <cstdlib>
#include <math.h>             // new: for log()
//using namespace std;

#define Usage "./task2_2 INPUT_IMAGE CUTOFF ORDER OUTPUT_butterworth OUTPUT_gaussia
n\n"

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

  Image inimg, outimg;
  int cutoff, order;

  if (argc < 6) {
    cout << Usage;
    exit(3);
  }

  // Convert Arugments to Integers
  cutoff = atoi(argv[2]);
  order = atoi(argv[3]);

  // read in image
  inimg = readImage(argv[1]);

  // filter image
  outimg = butterworthFilter(inimg, cutoff, order);

  // output image
  writeImage(outimg, argv[4], 1);

  // gaussian filter
  outimg = gaussianFilterFreq(inimg, cutoff);

  // output median image
  writeImage(outimg, argv[5], 1);

  return 0;
}
```

```
/*********************************************************************
 * task3_1.cpp - Implementation file for task3_1
 *
 * Author: Steven Young (syoung22@utk.edu), EECS
 *         University of Tennessee
 *
 * Class: ECE 572 Fall 2011
 *
 * Created: 10/01/2011
 *
 * Note:
 *
 *********************************************************************/
#include "spatialFilter.h"
#include "Image.h"
#include "Dip.h"
//#include <iostream>
#include <cstdlib>
#include <math.h>             // new: for log()
//using namespace std;

#define Usage "./task3_1 input_image BLUR_AMOUNT output_unsharp output_sobel output
_laplace\n"

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

  Image inimg, outimg;
  int mask_size, i, j, nr, nc;

  if (argc < 6) {
    cout << Usage;
    exit(3);
  }

  // Convert Arugments to Integers
  mask_size = atoi(argv[2]);

  // read in image
  inimg = readImage(argv[1]);
  nr = inimg.getRow();
  nc = inimg.getCol();

  // filter image
  outimg = unsharp(inimg, mask_size);

  // write image
  writeImage(outimg, argv[3], 0);

  // sobel filter
  outimg = sobelFilt(inimg);

  // output sobel
  writeImage(outimg, argv[4], 0);

  // laplacian filter
  outimg = laplacianFilter(inimg);

  // output laplace
  writeImage(outimg, argv[5], 0);

  return 0;
}
```

```
/*********************************************************************
 * task3_2.cpp - Implementation file for task3_2
 *
 * Author: Steven Young (syoung22@utk.edu), EECS
 *         University of Tennessee
 *
 * Class: ECE 572 Fall 2011
 *
 * Created: 10/01/2011
 *
 * Note:
 *
 ********************************************************************/
#include "freqFilter.h"
#include "Image.h"
#include "Dip.h"
//#include <iostream>
#include <cstdlib>
#include <math.h>             // new: for log()
//using namespace std;

#define Usage "./task3_2 INPUT_IMAGE CUTOFF ORDER OUTPUT_butterworth OUTPUT_gaussia
n\n"

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

  Image inimg, inimg_raw, outimg, outimg_raw;
  int cutoff, order, nr, nc, i, j, sq;

  if (argc < 6) {
    cout << Usage;
    exit(3);
  }

  // Convert Arugments to Integers
  cutoff = atoi(argv[2]);
  order = atoi(argv[3]);

  // read in image
  inimg_raw = readImage(argv[1]);
  nr = inimg_raw.getRow();
  nc = inimg_raw.getCol();

  // pad image for fft
  sq = 256;
  inimg.createImage(sq,sq);

  for(i=0;i<sq;i++){
    for(j=0;j<sq;j++){
      if(i<nr && j<nc){
        inimg(i,j,0)=inimg_raw(i,j,0);
      }else{
        inimg(i,j,0)=0;
      }
    }
  }

  // filter image
  outimg = butterworthFilterHP(inimg, cutoff, order);
  outimg_raw.createImage(nr,nc);

  for(i=0;i<nr;i++){
    for(j=0;j<nc;j++){
      outimg_raw(i,j,0)=outimg(i,j,0);
    }
  }

  // output image
  writeImage(outimg_raw, argv[4], 1);

  // gaussian filter
  outimg = gaussianFilterFreqHP(inimg, cutoff);
  for(i=0;i<nr;i++){
    for(j=0;j<nc;j++){
      outimg_raw(i,j,0)=outimg(i,j,0);
    }
  }

  // output median image
  writeImage(outimg_raw, argv[5], 1);

  return 0;
}
```

```cpp
/********************************************************************
 * task4.cpp - Implementation file for task4
 *
 * Author: Steven Young (syoung22@utk.edu), EECS
 *         University of Tennessee
 *
 * Class: ECE 572 Fall 2011
 *
 * Created: 10/01/2011
 *
 * Note:
 *
 ********************************************************************/
#include "freqFilter.h"
#include "spatialFilter.h"
#include "Image.h"
#include "Dip.h"
#include "conv.h"
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <math.h>             // new: for log()
using namespace std;

#define Usage "./task4 INPUT_IMAGE OUTPUT_spatial OUTPUT_frequency OUTPUT_autospati
al OUTPUT_autofreq\n"

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

  Image inimg, outimg, outimg2, mask, img_flip, diff;
  int i, j, sq, nr, nc;
  clock_t start, end;

  if (argc < 5) {
    cout << Usage;
    exit(3);
  }

  // read in image
  inimg = readImage(argv[1]);

  // spatial filter
  start = clock();
  outimg = averageFilt(inimg, 5);
  end = clock();
  cout << "Spatial Filter(s): " << (end-start)/(double)CLOCKS_PER_SEC << endl;

  // output image
  writeImage(outimg, argv[2], 1);

  // freq filter
  start = clock();
  outimg2 = averageFilterFreq(inimg);
  end = clock();
  cout << "Frequency Filter(s): " << (end-start)/(double)CLOCKS_PER_SEC << endl;

  // output image
  writeImage(outimg2, argv[3], 1);

  // diff image
  diff = outimg2-outimg;

  //output image
  writeImage(diff, "../output/diffImage.pgm", 0);

  // autocorellation spatial
  start = clock();
  nr = inimg.getRow();
  nc = inimg.getCol();

  img_flip.createImage(nr,nc);
  for(i=0;i<nr;i++){
    for(j=0;j<nc;j++){
      img_flip(i,j,0)=inimg(nr-1-i, nc-1-j,0);
    }
  }
  outimg = conv(inimg, img_flip);
  end = clock();
  cout << "Spatial Filter(s): " << (end-start)/(double)CLOCKS_PER_SEC << endl;

  // output image
  writeImage(outimg, argv[4], 1);

  // autocorellation freq
  start = clock();
  outimg = auto_freq(inimg);
  end = clock();
  cout << "Frequency Filter(s): " << (end-start)/(double)CLOCKS_PER_SEC << endl;

  // write image
  writeImage(outimg, argv[5], 1);
  return 0;
}
```

```
/*******************************************************************
 * extraCredit.cpp - Implementation file for extraCredit
 *
 * Author: Steven Young (syoung22@utk.edu), EECS
 *         University of Tennessee
 *
 * Class: ECE 572 Fall 2011
 *
 * Created: 10/01/2011
 *
 * Note:
 *
 *******************************************************************/

#include "freqFilter.h"
#include "Image.h"
#include "Dip.h"
//#include <iostream>
#include <cstdlib>
#include <math.h>             // new: for log()
//using namespace std;

#define Usage "./extraCredit input_image c gamma_l gamma_h cutoff output_image\n"

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

  Image inimg, outimg;
  int  nr, nc, i, j, sq, cutoff;
  float c, gamma_l, gamma_h;

  if (argc < 6) {
    cout << Usage;
    exit(3);
  }

  // Convert Arugments to Integers
  c = atof(argv[2]);
  gamma_l = atof(argv[3]);
  gamma_h = atof(argv[4]);
  cutoff = atoi(argv[5]);

  // read in image
  inimg= readImage(argv[1]);

  // homomorphic filter
  outimg = homomorphic(inimg, c, gamma_l, gamma_h, cutoff);

  // output median image
  writeImage(outimg, argv[6], 1);

  return 0;
}
```

```
/*******************************************************************
 * conv.cpp - Implementation file for convolution function
 *
 * Author: Steven Young (syoung22@utk.edu), EECS
 *         University of Tennessee
 *
 * Class: ECE 572 Fall 2011
 *
 * Created: 10/01/2011
 *
 * Note:
 *
 *******************************************************************/
#include "conv.h"
Image conv(Image input, Image mask){

  Image output;
  int szr, szc, mask_nr, mask_nc, mrad_r, mrad_c;
  int i, j, ii, jj;

  szr = input.getRow();
  szc = input.getCol();

  mask_nr = mask.getRow();
  mask_nc = mask.getCol();

  mrad_r = mask_nr/2;
  mrad_c = mask_nc/2;

  output.createImage(szr, szc);

  for(i=0;i<szr;i++){
    for(j=0;j<szc;j++){
      output(i,j,0)=0;
      for(ii=0;ii<mask_nr;ii++){
        for(jj=0;jj<mask_nc;jj++){
          if((i-mrad_r +ii >= 0) && (i-mrad_r +ii < szc) && (j-mrad_c +jj >= 0) &&
(j-mrad_c +jj < szr)){
            output(i,j,0) += input(i-mrad_r + ii, j-mrad_c +jj,0) * mask(mask_nr-1-
ii, mask_nc-1-jj, 0);
          }
        }
      }
    }
  }

  return output;
}
```

```
/*********************************************************************
 * conv.h - Header file for convolution function
 *
 * Author: Steven Young (syoung22@utk.edu), EECS
 *         University of Tennessee
 *
 * Class: ECE 572 Fall 2011
 *
 * Created: 10/01/2011
 *
 * Note:
 *
 *********************************************************************/

#ifndef LINE_H
#define LINE_H

#include "Image.h"

Image conv(Image, Image);

#endif
```

```
/*******************************************************************
 * square.cpp - Implementation file for square
 *
 * Author: Steven Young (syoung22@utk.edu), EECS
 *         University of Tennessee
 *
 * Class: ECE 572 Fall 2011
 *
 * Created: 10/01/2011
 *
 * Note:
 *
 *******************************************************************/

#include "square.h"


Image square(int szr, int szc, int diam_sq){

  Image output;
  int start_r, start_c, end_r, end_c;
  int i, j;

  output.createImage(szr, szc);

  start_r = szr/2 - diam_sq/2;
  start_c = szc/2 - diam_sq/2;

  end_r = start_r + diam_sq - 1;
  end_c = start_c + diam_sq - 1;

  for(i=0;i<szr;i++){
    for(j=0;j<szc;j++){
      if (i >= start_r && i <= end_r && j >= start_c && j <= end_c){
        output(i,j,0) = 255;
      }else{
        output(i,j,0) = 0;
      }
    }
  }

  return output;
}
```

```
/*********************************************************************
 * square.h - Header file for square
 *
 * Author: Steven Young (syoung22@utk.edu), EECS
 *         University of Tennessee
 *
 * Class: ECE 572 Fall 2011
 *
 * Created: 10/01/2011
 *
 * Note:
 *
 *********************************************************************/
#ifndef SQUARE_H
#define SQUARE_H

#include "Image.h"

Image square(int, int, int);

#endif
```

```
/*******************************************************************
 * line.cpp - Implementation file for line
 *
 * Author: Steven Young (syoung22@utk.edu), EECS
 *         University of Tennessee
 *
 * Class: ECE 572 Fall 2011
 *
 * Created: 10/01/2011
 *
 * Note:
 *
 *******************************************************************/

#include "line.h"


Image line(int szr, int szc, int loc_ln, int wid_ln){

  Image output;
  int start_c, end_c;
  int i, j;

  output.createImage(szr, szc);

  start_c = loc_ln;
  end_c = loc_ln + wid_ln -1;

  for(i=0;i<szr;i++){
    for(j=0;j<szc;j++){
      if (j >= start_c && j <= end_c){
        output(i,j,0) = 255;
      }else{
        output(i,j,0) = 0;
      }
    }
  }

  return output;
}
```

```
/*********************************************************************
 * line.h - Header file for line
 *
 * Author: Steven Young (syoung22@utk.edu), EECS
 *         University of Tennessee
 *
 * Class: ECE 572 Fall 2011
 *
 * Created: 10/01/2011
 *
 * Note:
 *
 *********************************************************************/
#ifndef LINE_H
#define LINE_H

#include "Image.h"

Image line(int, int, int, int);

#endif
```

```
/*********************************************************************
 * spatialFilter.cpp - Implementation file for spatial computations
 *
 * Author: Steven Young (syoung22@utk.edu), EECS
 *         University of Tennessee
 *
 * Class: ECE 572 Fall 2011
 *
 * Created: 10/01/2011
 *
 * Note:
 *
 *********************************************************************/

#include "spatialFilter.h"

Image gaussianFilter(Image inimg, int n){
  Image mask, output;
  int i, j, center;
  float den, sigma;

  // allocate memory
  mask.createImage(n, n);

  sigma = 1.5;
  den = 1/(2*PI*sigma*sigma);
  center = n/2;

  // create mask
  for(i=0;i<n;i++){
    for(j=0;j<n;j++){
      mask(i,j,0) = exp(-(abs(center-i)+abs(center-j))/(2*sigma*sigma)) / den;
    }
  }

  writeImage(mask, "../output/kernel.pgm", 1);

  //convolve image with filter
  output = conv(inimg, mask);

  return output;
}

Image medianFilter(Image inimg, int n){
  Image mask, output;
  int nr, nc, i, j, ii, jj, middle;

  nr = inimg.getRow();
  nc = inimg.getCol();

  output.createImage(nr,nc);

  // create mask
  for(i=0;i<nr;i++){
    for(j=0;j<nc;j++){
      list<int> p_values;
      for(ii=max(0,i-n/2);ii<min(nr,i+n/2+1);ii++){
        for(jj=max(0,j-n/2);jj<min(nc,j+n/2+1);jj++){
          p_values.push_back(inimg(ii,jj,0));
        }
      }
      p_values.sort();
      middle = p_values.size()/2;
      list<int>::iterator it = p_values.begin();
      advance(it,middle);
      output(i,j,0) = *it;
    }
  }

  return output;
}

Image unsharp(Image inimg, int blur_size){
  Image blur, sub_blur;

  //blur image
  blur = averageFilt(inimg, blur_size);

  // Subtract blur
  sub_blur = inimg - blur;

  // add back
  inimg = inimg + (sub_blur*10);

  return sub_blur;
}

Image averageFilt(Image inimg, int size){

  Image mask, outimg;
  int i, j;

  mask.createImage(size,size);
  for(i=0;i<size;i++){
    for(j=0;j<size;j++){
      mask(i,j,0)= 1.0/(size*size);
    }
  }

  outimg = conv(inimg, mask);

  return outimg;
}

Image sobelFilt(Image inimg){

  Image mask, output, output2;
  int i, j;

  mask.createImage(3,3);

  mask(0,0)=-1; mask(0,1)=-2; mask(0,2) = -1;
  mask(1,0)= 0; mask(1,1)= 0; mask(1,2) =  0;
  mask(2,0)= 1; mask(2,1)= 2; mask(2,2) =  1;

  output = conv(inimg, mask);

  for(i=0;i<output.getRow();i++){
    for(j=0;j<output.getCol();j++){
      output(i,j,0) = abs(output(i,j,0));
    }
  }

  mask(0,0)=-1; mask(0,1)= 0; mask(0,2) =  1;
  mask(1,0)=-2; mask(1,1)= 0; mask(1,2) =  2;
  mask(2,0)=-1; mask(2,1)= 0; mask(2,2) =  1;

  output2 = conv(inimg, mask);

  for(i=0;i<output2.getRow();i++){
    for(j=0;j<output2.getCol();j++){
      output2(i,j,0) = abs(output2(i,j,0));
    }
  }
  return output + output2;
}

Image laplacianFilter(Image inimg){

  Image mask, output;

  mask.createImage(3,3);

  mask(0,0)= 0; mask(0,1)=-1; mask(0,2) =  0;
  mask(1,0)=-1; mask(1,1)= 4; mask(1,2) = -1;
  mask(2,0)= 0; mask(2,1)=-1; mask(2,2) =  0;

  output = conv(inimg, mask);

  return output;
}
```

```
/*********************************************************************
 * spatialFilter.h - Header file for spatial computations
 *
 * Author: Steven Young (syoung22@utk.edu), EECS
 *         University of Tennessee
 *
 * Class: ECE 572 Fall 2011
 *
 * Created: 10/01/2011
 *
 * Note:
 *
 *********************************************************************/
#ifndef SPATIALFILTER_H
#define SPATIALFILTER_H

#include "Image.h"
#include <math.h>
#include "Dip.h"
#include "conv.h"
#include <list>
Image gaussianFilter(Image, int);
Image medianFilter(Image, int);
Image unsharp(Image, int);
Image averageFilt(Image, int);
Image sobelFilt(Image);
Image laplacianFilter(Image);
#endif
```

```
/*********************************************************************
 * freqFilter.cpp - Implementation file for frequency computations
 *
 * Author: Steven Young (syoung22@utk.edu), EECS
 *         University of Tennessee
 *
 * Class: ECE 572 Fall 2011
 *
 * Created: 10/01/2011
 *
 * Note:
 *
 *********************************************************************/

#include "freqFilter.h"

Image butterworthFilter(Image inimg, int cutoff, int order){

  Image mag, phase, outimg;
  int nr, nc, i , j;

  nr = inimg.getRow();
  nc = inimg.getCol();

  // allocate memory
  mag.createImage(nr, nc);
  phase.createImage(nr, nc);
  outimg.createImage(nr, nc);

  // get fft
  fft(inimg, mag, phase);
  float total_power = calculatePower(mag);

  // filter
  nr = mag.getRow();
  nc = mag.getCol();

  for(i=0;i<nr;i++){
    for(j=0;j<nc;j++){
      mag(i,j,0) = mag(i,j,0) /(1+pow(d2c(nr,nc,i,j)/cutoff,2*order));
    }
  }
  float power = calculatePower(mag);

  cout << "LP Butterworthn PR: " << 100*power/total_power << endl;

  // get ifft
  ifft(outimg, mag, phase);

  return outimg;
}

Image butterworthFilterHP(Image inimg, int cutoff, int order){

  Image mag, phase, outimg;
  int nr, nc, i , j;

  nr = inimg.getRow();
  nc = inimg.getCol();

  // allocate memory
  mag.createImage(nr, nc);
  phase.createImage(nr, nc);
  outimg.createImage(nr, nc);

  // get fft
  fft(inimg, mag, phase);
  float total_power = calculatePower(mag);

  // filter
  nr = mag.getRow();
  nc = mag.getCol();

  for(i=0;i<nr;i++){
    for(j=0;j<nc;j++){
      mag(i,j,0) = mag(i,j,0) /(1+pow(cutoff/d2c(nr,nc,i,j),2*order));
    }
  }

  float power = calculatePower(mag);

  cout << "HP Butterworthn PR: " << 100*power/total_power << endl;

  // get ifft
  ifft(outimg, mag, phase);

  return outimg;
}


Image gaussianFilterFreq(Image inimg, int cutoff){

  Image mag, phase, outimg;
  int nr, nc, i , j;

  nr = inimg.getRow();
  nc = inimg.getCol();

  // allocate memory
  mag.createImage(nr, nc);
  phase.createImage(nr, nc);
  outimg.createImage(nr, nc);

  // get fft
  fft(inimg, mag, phase);
  float total_power = calculatePower(mag);

  // filter
  nr = mag.getRow();
  nc = mag.getCol();

  for(i=0;i<nr;i++){
    for(j=0;j<nc;j++){
      float dist = d2c(nr,nc,i,j);
      mag(i,j,0) = mag(i,j,0) * exp(-dist*dist/(2*cutoff*cutoff));
    }
  }

  float power = calculatePower(mag);

  cout << "LP Gaussian PR: " << 100*power/total_power << endl;

  // get ifft
  ifft(outimg, mag, phase);

  return outimg;
}

Image gaussianFilterFreqHP(Image inimg, int cutoff){

  Image mag, phase, outimg;
  int nr, nc, i , j;

  nr = inimg.getRow();
  nc = inimg.getCol();

  // allocate memory
  mag.createImage(nr, nc);
  phase.createImage(nr, nc);
  outimg.createImage(nr, nc);


  // get fft
  fft(inimg, mag, phase);
  float total_power = calculatePower(mag);

  // filter
  nr = mag.getRow();
  nc = mag.getCol();

  for(i=0;i<nr;i++){
    for(j=0;j<nc;j++){
      float dist = d2c(nr,nc,i,j);
      mag(i,j,0) = mag(i,j,0) *(1 -  exp(-dist*dist/(2*cutoff*cutoff)));
    }
  }

  float power = calculatePower(mag);

  cout << "HP Gaussian PR: " << 100*power/total_power << endl;

  // get ifft
  ifft(outimg, mag, phase);

  return outimg;
}

Image averageFilterFreq(Image inimg){

  Image mag, phase, outimg, mask, mask_mag, mask_phase, img_pad;
  int nr, nc, i , j;

  nr = inimg.getRow();
  nc = inimg.getCol();

  // create padded mask
  mask.createImage(2*nr,2*nc);
  for(i=0;i<5;i++){
    for(j=0;j<5;j++){
      mask(i,j,0) = 1.0/25.0;
    }
  }

  // created padded image
  img_pad.createImage(2*nr,2*nc);
  for(i=0;i<nr;i++){
    for(j=0;j<nc;j++){
      img_pad(i+2,j+2,0) = inimg(i,j,0);
    }
  }

  // allocate memory
  mag.createImage(2*nr,2*nc);
  phase.createImage(2*nr,2*nc);
  outimg.createImage(2*nr,2*nc);
  mask_mag.createImage(2*nr,2*nc);
  mask_phase.createImage(2*nr,2*nc);

  // get fft
  fft(img_pad, mag, phase);
  fft(mask, mask_mag, mask_phase);

  // filter
  nr = mag.getRow();
  nc = mag.getCol();

  for(i=0;i<nr;i++){
    for(j=0;j<nc;j++){
      mag(i,j,0) = mag(i,j,0) * mask_mag(i,j,0);
      phase(i,j,0) = phase(i,j,0) - mask_phase(i,j,0);
    }
  }

  // get ifft
  ifft(outimg, mag, phase);

  outimg = subImage(outimg,0,0,nr/2-1, nc/2-1);
  return outimg;
}

Image auto_freq(Image inimg){

  Image in_pad, mag, phase, outimg, img_flip, if_pad, magf, phasef, outimg2;
  int nr, nc, i , j;

  nr = inimg.getRow();
  nc = inimg.getCol();

  in_pad.createImage(2*nr, 2*nc);

  for(i=0;i<nr;i++){
    for(j=0;j<nc;j++){
      in_pad(i,j,0)=inimg(i,j,0);
    }
  }

  nr = in_pad.getRow();
  nc = in_pad.getCol();

  // allocate memory
  mag.createImage(nr, nc);
  phase.createImage(nr, nc);
  outimg.createImage(nr, nc);
  magf.createImage(nr, nc);
  phasef.createImage(nr, nc);

  // get fft
  fft(in_pad, mag, phase);

  // filter
  nr = mag.getRow();
  nc = mag.getCol();


  for(i=0;i<nr;i++){
    for(j=0;j<nc;j++){
      mag(i,j,0) = mag(i,j,0) * mag(i,j,0);
      phase(i,j,0) = 0;
    }
  }

  // get ifft
  ifft(outimg, mag, phase);

  outimg2.createImage(nr/2, nc/2);

  // rearrange image
  for(i=0;i<nr/2;i++){
    for(j=0;j<nc/2;j++){
      int x = 3*nr/4+i;
      int y = 3*nc/4+j;
      if(x > nr-1){
        x = x - nr;
      }
      if(y > nc-1){
        y = y - nc;
      }
      outimg2(i,j,0) = outimg(x,y,0);
    }
  }
  return outimg2;
}

float d2c(int nr, int nc, int x, int y){

  float nrf = (float) nr;
  float ncf = (float) nc;

  float xf = (float) x;
  float yf = (float) y;

  return sqrt( ((nrf/2)-xf)*((nrf/2)-xf) + ((ncf/2)-yf)*((ncf/2)-yf));
}

Image homomorphic(Image inimg, float c, float gamma_l, float gamma_h, int cutoff){

  Image mag, phase, outimg;
  int nr, nc, i , j;

  nr = inimg.getRow();
  nc = inimg.getCol();

  // allocate memory
  mag.createImage(nr, nc);
  phase.createImage(nr, nc);
  outimg.createImage(nr, nc);


  // get fft
  fft(inimg, mag, phase);

  // filter
  nr = mag.getRow();
  nc = mag.getCol();

  for(i=0;i<nr;i++){
    for(j=0;j<nc;j++){
      float dist = d2c(nr,nc,i,j);
      mag(i,j,0) = mag(i,j,0) *((gamma_h-gamma_l)*(1 -  exp(-c*dist*dist/(cutoff*cu
toff)))+gamma_l);
    }
  }


  // get ifft
  ifft(outimg, mag, phase);

  return outimg;
}

float calculatePower(Image mag){

  int i, j, nr, nc;
  nr = mag.getRow();
  nc = mag.getCol();

  // Calculate Power
  float p_t = 0;
  for(i=0;i<nr;i++){
    for(j=0;j<nc;j++){
      p_t += mag(i,j,0) * mag(i,j,0);
    }
  }

  return p_t;
}
```

```
/*********************************************************************
 * freqFilter.h - Header file for frequency computations
 *
 * Author: Steven Young (syoung22@utk.edu), EECS
 *         University of Tennessee
 *
 * Class: ECE 572 Fall 2011
 *
 * Created: 10/01/2011
 *
 * Note:
 *
 *********************************************************************/
#ifndef FREQFILTER_H
#define FREQFILTER_H

#include "Image.h"
#include <math.h>
#include "Dip.h"
#include "conv.h"
#include <list>
Image butterworthFilter(Image, int, int);
Image butterworthFilterHP(Image, int, int);
Image gaussianFilterFreq(Image, int);
Image gaussianFilterFreqHP(Image, int);
Image averageFilterFreq(Image);
Image auto_freq(Image);
float d2c(int, int, int, int);
Image homomorphic(Image, float, float, float, int);
float calculatePower(Image);
#endif
```