Abstract:

The purpose of this project is to study image filtering in spatial domain. It consists of four parts: the first one discusses the spatial filtering of an image using a spatial mask 3x3, and then this mask is used in blurring filter and Laplacian. The second part studies the order statistics filters, specially the median filter. The third part discusses the unsharp masking filter as an example we implement the high-boost filter. The last part discusses anisotropic diffusion filter, and implementation of 2D anisotropic algorithm.
Technical Discussion

PART 1:

- Spatial filtering is performed by convolving the image with a mask or a kernel
- Spatial filters include sharpening, smoothing, edge detection, noise removal, etc.
- In general, linear filtering of an image \( f \) of size \( M \times N \) with filter size \( m \times n \) is given by the expression

\[
g(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) f(x + s, y + t)
\]

\[
a = (m - 1) / 2, \quad b = (n - 1) / 2
\]

- Smoothing Spatial Filters
  - The output of a smoothing spatial filter is simply the average of the pixels contained in the neighborhood of the filter mask.
  - These filters are sometimes called *averaging filters* and also *lowpass filters*
  - Two types of masks of the spatial filter

<table>
<thead>
<tr>
<th>( \frac{1}{9} \times )</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \frac{1}{16} \times )</th>
<th>1</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

- The Laplacian \( \nabla^2 f \)
  - The *Laplacian* of an image is define as
\[ \nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \]

\[ \frac{\partial^2 f}{\partial x^2} = f(x+1, y) - 2f(x, y) + f(x-1, y) \]

\[ \frac{\partial^2 f}{\partial y^2} = f(x, y+1) - 2f(x, y) + f(x, y-1) \]

\[ \nabla^2 f = [f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1)] - 4f(x, y) \]

- Types of masks of the Laplacian filter

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-4</td>
<td>1</td>
<td>1</td>
<td>-8</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>4</td>
<td>-1</td>
<td>-1</td>
<td>8</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

PART 2:

- Order statistics filters are nonlinear spatial filters whose response is based on ordering (ranking) the pixels contained in an area covered by the filter
- The best known example in this category is median filter
- median filter
  - Median filters replace the value of the pixel by the median of the gray levels in the neighborhood of that pixel
PART 3:

- Unsharp Masking consists of subtracting a blurred version of the image from the image itself

\[ f_s(x, y) = f(x, y) - \tilde{f}(x, y) \]

- The best known example in this category is high boost filter

- **high boost filter**
  - A slight generalization of unsharp masking is called high boost filters

\[ f_{hb}(x, y) = A \ f(x, y) - \tilde{f}(x, y) \]

- Types of masks of the high boost filter

<table>
<thead>
<tr>
<th>0</th>
<th>-1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>A + 4</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

PART 4:

- Anisotropic diffusion filter overcome the major drawbacks of conventional spatial filtering, namely the blurring of object boundaries and suppression of fine structural details.

- In any dimension diffusion process can be formulated mathematically as follows:

\[ \frac{\partial}{\partial t} I(\tilde{x}, t) = \text{div}[c(\tilde{x}, t) \ \text{grad} I(\tilde{x}, t)] \]

Where \( I(\tilde{x}, t) \) is the image intensity, \( \tilde{x} \) represents the spatial coordinates, \( t \) is the process ordering parameter, and \( c(\tilde{x}, t) \) is the diffusion function.
\[ c(x, t) = \exp \left( - \left( \frac{\nabla I(x, t)}{k} \right)^2 \right) \]

The image intensity gradient \( \nabla I(x, t) \) can be calculated as follow:

\[ \nabla I(x, y) = \sqrt{(I(x + 1, y) - I(x, y))^2 + (I(x, y + 1) - I(x, y))^2} \]

- The diffusion equation for 2D in digital case is derived as follows:

\[
\frac{\partial}{\partial t} I(x, t) = \text{div}[c(x, t) \cdot \text{grad}(\tilde{x}, t)]
\]

\[= \nabla^T [c(x, t) \cdot \nabla I(x, t)]
\]

\[= \frac{\partial}{\partial x} [c(x, t) \cdot \frac{\partial}{\partial x} I(x, t)] + \frac{\partial}{\partial y} [c(x, t) \cdot \frac{\partial}{\partial y} I(x, t)]
\]

\[\approx \frac{1}{\Delta x^2} [c(x, y, t) \cdot (I(x + \Delta x, y, t) - I(x, y, t)) - c(x - \Delta x, y, t) \cdot (I(x, y, t) - I(x - \Delta x, y, t))] + \frac{1}{\Delta y^2} [c(x, y, t) \cdot (I(x, y + \Delta y, t) - I(x, y, t)) - c(x, y - \Delta y, t) \cdot (I(x, y, t) - I(x, y - \Delta y, t))]
\]

\[= \Phi_{\text{east}} - \Phi_{\text{west}} + \Phi_{\text{north}} + \Phi_{\text{south}}\]
Results
Part 1

A) Spatial filter (blurring & Laplacian filter) (circles image)

- Fig(1-1) shows the original image
- Fig(1-2) shows the blurred image
- Fig(1-3) shows the Laplacian output image
B) Spatial filter (blurring & Laplacian filter)

- Fig(1-4) shows the original image
- Fig(1-5) shows the blurred image
- Fig(1-6) shows the Laplacian output image
Part 2

A) Order statistics filter (Median filter) (lena image)

- Fig(2-1) shows the original image
- Fig(2-2) shows the original image histogram
- Fig(2-3) shows the noisy image ('salt & pepper' noise)
- Fig(2-4) shows the noisy image histogram
- Fig(2-5) shows the output median filter image
- Fig(2-6) shows the output median filter image histogram
B) Order statistics filter (Median filter) (camera image)

- Fig(2-7) shows the original image
- Fig(2-8) shows the original image histogram
- Fig(2-9) shows the noisy image ('salt & pepper' noise)
- Fig(2-10) shows the noisy image histogram
- Fig(2-11) shows the output median filter image
- Fig(2-12) shows the output median filter image histogram
Part 3

A) Unsharp masking filter (High-boost filter)

- Fig(3-1) shows the original image (lena image)
- Fig(3-2) shows output High-boost filter image for A=1.5
- Fig(3-3) shows the original image (montage image)
- Fig(3-4) shows output High-boost filter image for A=1.4
- Fig(3-5) shows the original image (goldhill image)
- Fig(3-6) shows output High-boost filter image for $A=1.4$
- Fig(3-7) shows the original image (horiz image)
- Fig(3-8) shows output High-boost filter image for $A=1.9$
Part 4

Anisotropic diffusion filter

- Fig(4-1) shows the original image (circles image)
- Fig(4-2) shows output of the filter for $k=0.01$
- Fig(4-2) shows output of the filter for $k=0.1$
- Fig(4-2) shows output of the filter for $k=1$
Results Discussion

PART 1:

- As shown from the results the output of a smoothing spatial filter is simply the average of the pixels contained in the neighborhood of the filter mask and the output image become more blurred (e.g. Fig.(1.2))
- The output of a Laplacian filter (e.g. Fig.(1.3)) is the edges and there is no blurred parts

PART 2:

- As shown from the results the median filter can remove the salt & paper noise (e.g. Fig.(2-3)) but make the output image (e.g. Fig.(2-5)) is more blurred than original one (e.g. Fig.(2-1)).
- The histogram of the noised image (e.g. Fig.(2-4)) contains two pulses in the sides and the median filter removes them as shown in the histogram of the output image (e.g. Fig.(2-6))

PART 3:

- As shown from the results Unsharp masking filter (High-boost filter) removes the blurred parts and enhances the edges
- We control the ratio of enhancement by the value A if A=0 High-boost filter will become Laplacian filter and if 1<A<3 the output image (e.g. Fig.(3-2)) will be more clear than original one (e.g. Fig.(3-1)).

PART 4:

- The nonlinear anisotropic filtering can reduce the noise efficiently and sharpen the object boundaries this is the main advantage than the spatial filter.
- Selecting the parameters of the filter is very critical
- The parameter k is control factor as k become smaller all boundary will appear and as k becomes larger some boundary will disappear
- When k=0.01 the output image (e.g. Fig.(4-2)) is more clear when k=1 the output image (e.g. Fig.(4-4)) is more blurred.
Appendix

Program Listings

Part 1(Matlab) //using the readpgm.m in the web site

function FltrIm=SpatialFilter(Im,Mask)
[M N]=size(Im);
[m n]=size(Mask);
a=(m-1)/2;
b=(n-1)/2;
g=zeros(M,N);
for x=1:M,
    for y=1:N,
        for s=-a:a,
            for t=-b:b
                if((x+s)>0 & (y+t)>0 & (x+s)<=M & (y+t)<=N)
                    g(x,y)=g(x,y)+Mask(s+a+1,t+b+1)*Im(x+s,y+t);
                end
            end
        end
    end
end
FltrIm=g;
%
main Program
clear all;
clc;
im=readpgm('E:\Classes\MATLAB_PROGRAM\ECE618\pgm_images\bridge.pgm');
L_max=max(max(im))
L_min=min(min(im))
imshow(im,[0 255]);
%m=(1.0/(3*3.0))*ones(3,3);
m=[0 -1 0;-1 4 -1;0 -1 0];
IMG1=SpatialFilter(im,m);
L_mx=max(max(IMG1))
L_mn=min(min(IMG1))
figure,imshow(IMG1,[L_mn L_mx]);

Part 2(Matlab)

function FltrIm=MedianFilter(Im)
[M N]=size(Im);
a=2;
b=2;
for x=1:M,
    for y=1:N,
        k=1;
        for s=-a:a,
            for t=-b:b
                if((x+s)>0 & (y+t)>0 & (x+s)<=M & (y+t)<=N)
                    g(k)=Im(x+s,y+t);
                else
                    g(k)=0;
                end
                k=k+1;
            end
        end
        FltrIm(x,y)=median(g);
    end
end

%%%%%main Program
clear all;
clc;
im=readpgm('E:\Classes\MATLAB_PROGRAM\ECE618\pgm_images\camera.pgm');
[N M]=size(im);
immax=max(max(im));
L_max=max(max(im))
L_min=min(min(im))
figure(1),imshow(im,[L_min L_max]);
[N M]=size(im);
Vect=reshape(im,N*M,1);
[Pr x]=hist(Vect,[L_min:L_max]);
Pr=Pr/(N*M);
figure(2), plot(x,Pr),grid;
in = immax*imnoise(1.0*im/immax,'salt & pepper',0.02);
L_mx=max(max(in))
L_mn=min(min(in))
figure(3),imshow(in,[L_mn L_mx]);
Vect=reshape(in,N*N,1);
[Pr x]=hist(Vect,[L_mn:L_mx]);
Pr=Pr/(N*N);
figure(4), plot(x,Pr),grid;
IMG1=MedianFilter(in);
L_mx=max(max(IMG1))
L_mn=min(min(IMG1))
figure(5),imshow(IMG1,[L_mn L_mx]);
Vect=reshape(IMG1,N*N,1);
[Pr x]=hist(Vect,[L_mn:L_mx]);
Pr=Pr/(N*N);
figure(6), plot(x,Pr),grid;
function FltrIm=SpatialFilter(Im,Mask)
[M N]=size(Im);
[m n]=size(Mask);
a=(m-1)/2;
b=(n-1)/2;
g=zeros(M,N);
for x=1:M,
    for y=1:N,
        for s=-a:a,
            for t=-b:b
                if((x+s)>0 & (y+t)>0 & (x+s)<=M & (y+t)<=N)
                    g(x,y)=g(x,y)+Mask(s+a+1,t+b+1)*Im(x+s,y+t);
                end
            end
        end
    end
end
FltrIm=g;

%%%%%%%%main Program
clear all;
clc;
im=readpgm('E:\Classes\MATLAB_PROGRAM\ECE618\pgm_images\horiz.pgm');
[N M]=size(im);
L_max=max(max(im));
% im=im*1.0/L_max;
L_max=max(max(im))
L_min=min(min(im))
figure,imshow(im,[0 255]);
A=3;
m=[1 -1 -1; -1 A+8 -1; -1 -1 -1];
IMG1=SpatialFilter(im,m);
L_mx=max(max(IMG1))
L_mn=min(min(IMG1))
figure,imshow(IMG1,[L_mn L_mx]);
dm=im-IMG1;
L_mx=max(max(dm))
L_mn=min(min(dm))
figure,imshow(dm,[L_mn L_mx]);


```cpp
//using the image.c and proto.h in the web site
int _tmain(int argc, _TCHAR* argv[])
{
  char *filename="E:\Classes\MATLAB_PROGRAM\ECE618\pgm_images\circles.pgm";
  Image *image;
  image=ReadPNMImage(filename);
  int i, N,M;
  unsigned char *im;
  N=image->Width;
  M=image->Height;
  im=image->data;
  unsigned char I[260][260];
  int j;

  for(i=0;i<N;i++)
    for(j=0;j<M;j++)
      I[2+i][2+j]=*im++;
    for(i=0;i<M;i++)
      I[0][2+i]=I[2][2+i];
      I[1][2+i]=I[2][2+i];
      I[N+2][2+i]=I[N+1][2+i];
      I[N+3][2+i]=I[N+1][2+i];
  }
  for(i=0;i<N+4;i++)
    I[i][0]=I[i][2];
    I[i][1]=I[i][2];
    I[i][M+2]=I[i][M+1];
    I[i][M+3]=I[i][M+1];
}

double DI[260][260];
int x,y;

double PHIe,PHIw,PHIn,PHIs;
double k=0.1;
double t,dt=0.2;
for(t=0;t<25;t+=dt)
{
  for(x=2;x<N+1;x++)
    for(y=2;y=M+1;y++)
      DI[x][y]=sqrt((double)((I[x][y]-I[x-1][y])*(I[x][y]-I[x-1][y])+"[1][y])+(I[x][y]-I[x][y-1]))*(I[x][y]-I[x][y-1]));
}
"[1][y])+(I[x][y]-I[x][y-1]));
  for(x=2;x<N+1;x++)
    for(y=2;y=M+1;y++)
      PHIe=exp(-
        (abs(DI[x][y]/2+DI[x][y+1]/2)/k)*(abs(DI[x][y]/2+DI[x][y+1]/2)/k))*(I[x+1][y]-I[x][y]);
      PHIw=exp(-
        (abs(DI[x][y]/2+DI[x][y+1]/2)/k)*(abs(DI[x][y]/2+DI[x][y+1]/2)/k))*(I[x][y]-I[x-1][y]);
      PHIn=exp(-
        (abs(DI[x][y]/2+DI[x][y+1]/2)/k)*(abs(DI[x][y]/2+DI[x][y+1]/2)/k))*(I[x][y+1]-I[x][y]);
      PHIs=exp(-
        (abs(DI[x][y]/2+DI[x][y+1]/2)/k)*(abs(DI[x][y]/2+DI[x][y+1]/2)/k))*(I[x][y]-I[x][y-1]);
```
I[x][y] = (int)(I[x][y] + dt*(PHIe-PHIw+PHIn-PHI$I$));
}}

Image *outimage;
outimage = CreateNewImage(image, "#testing Swap");

unsigned char *tempout;
tempout = outimage->data;
for (x=0; x<N; x++) {
    for (y=0; y<M; y++) {
        *tempout++ = I[x+2][y+2];
    }
}

char *outfilename = "E:\Classes\MATLAB_PROGRAM\ECE618\pgm_images\outimg.pgm";
SavePNMImage(outimage, outfilename);

return 0;
}