

Image Restoration by Inverse Filtering in the Frequency Domain Using Gaussian and Ideal Low Pass Filters

By Nasser Abbasi

Introduction

This report was written during Fall 2004. EECS 207A. UCI.

This is an implementation of a standard algorithm for 2D gray image restoration which is based on a mathematical model of image degradation.

Image restoration attempts to recover, as much as possible, the original image from the degraded image, using knowledge of the process that caused the degradation itself.

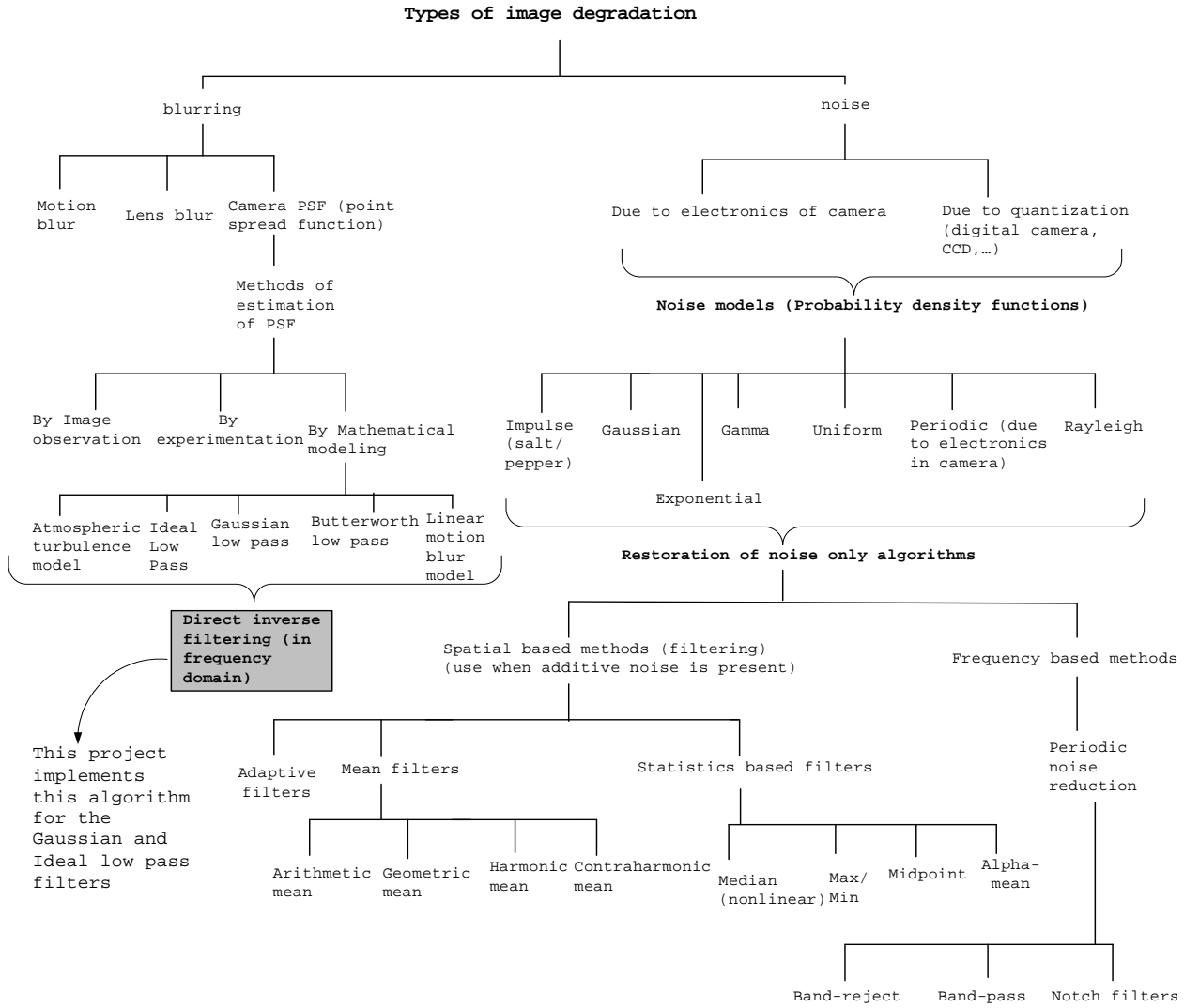
The restoration algorithm used assumes we know the mathematical model of the degradation that we are trying to remove.

Degradation is caused by either blur or by noise. There are algorithms to do restoration in the frequency domain and those that do the restoration directly in the spatial domain.

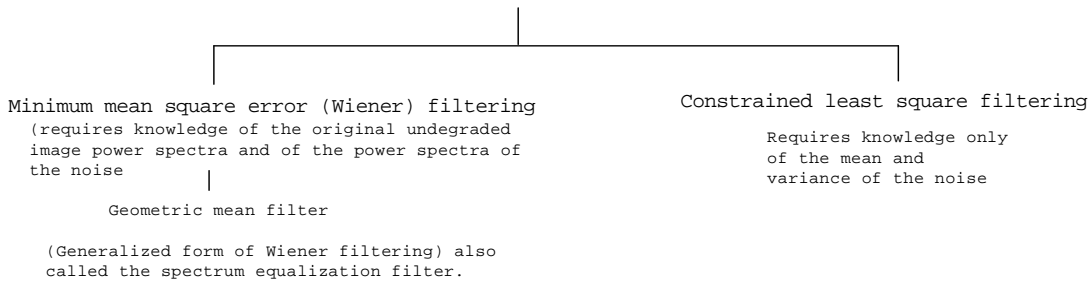
There are algorithms that concentrate on reversing the effects of noise only, those that concentrate on removing the effects of blur, and those that attempt to remove the effect of both noise and blur.

This project is an implementation of an algorithm that works in the frequency domain which inverts the effect of blur only.

To help illustrate this, I have made the following diagram which shows the different algorithms used for image restorations. This diagram shows that the field of image restoration is large and complex and is based on the mathematical modeling of image degradation.



Algorithms that incorporates degradation due to PSF and noise



Geometric transformation algorithms for restoration

These class of algorithms modify the spatial relationships between pixels in the image. Also called Rubber-sheet transformations.

Overview.vsd
by Nasser Abbasi
December 6, 2004

As can be seen from the diagram above, it is important to know that different blur degradations require the use of

different PSF (Point Spread Function) mathematical model.

For example, motion blur would require different PSF than say a degradation caused by ripple distortion or by a normal blur degradation. Even blur degradation itself has many different types (motion, lens, Gaussian, Radial, etc...) and different mathematical function for PSF would be needed for each type to obtain an accurate restoration that matches the original image as best as possible. For instance, degradation by motion blur, would require a mathematical function for PSF which would take an estimate of the linear or radial motion parameters that caused the blur.

Given the above, and to be able to illustrate the direct inversion algorithm, I had to assume a certain PSF model. I have selected to use the 2D Gaussian low pass filter and the 2D ideal low pass filter. Using these filters, I will start by generating a number of degraded images from an original single undegraded image. I will use different standard deviation values for the Gaussian for each generation of a degraded image, and different radius values for the ideal low pass filter.

In this case, the standard deviation used for the Gaussian is nothing but the radius as well. The radius is measured as the number of pixels from the center or the spectrum. In all cases, the 2D spectrum will be centered in the middle of the image.

After degrading the original image, I will then restore these images using the inverse filtering algorithm and compare the result with the original, undistorted image. I will observe how using different radius values affect the restoration quality, and will compare visually each restored image to the original image and comment on the quality of the restoration.

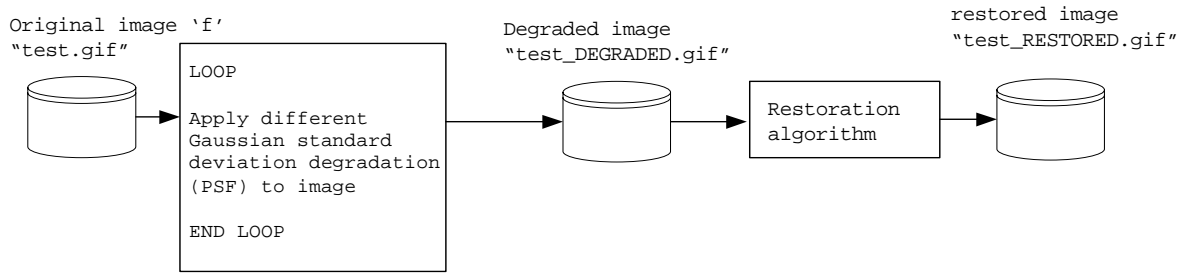
The standard image of Lena will be used throughout this evaluation. It is a gray level, 512 x 512 pixels image that I downloaded from the internet. I have used a gray level image to simplify the implementation, otherwise I would have to perform 2D fourier transformation and inverse transformation on each of the 3 channels. For the purpose of illustrating the algorithm, I did not feel this would not have adding any more value.

The restoration algorithm can be implemented completely in the spatial domain if needed. However, since the algorithm requires performing a convolution operation, and since convolution in the spatial domain is equivalent to multiplication in the frequency domain, we will start by transforming the input image to the frequency domain to take advantage of the speed of the FFT (Fast Fourier Transforms).

In this implementation, an image is read, degraded using the PSF, and then restored using the direct inversion algorithm. Both the degraded and restored images are saved to disk.

The name of the output image file will be the same as the input image file, but with the word `_RESTORED` and `_DEGRADED` appended to the file name. The type of the images outputted will be in the same graphic format as the input image.

This diagram below illustrates the data flow of the program as a black box



A small note on how this report was written

This report was itself written in *Mathematica* in the same file as the program itself. This made it easier to add comments on the program and have these as part of the report itself as the same time. This feature is attractive since it eliminates the need to separate the program from the documentation or the report as would be the case in other systems.

Project Details

Mathematical model of image degradation

The algorithm is based on the following mathematical model of image degradation.

Let the degraded image be called g . Let the camera PSF (point spread function) be called h (This is the degradation function which depends on the system itself, i.e. the camera). Let noise introduced into the degraded image be called η . Let the original, undegraded image be called f .

The degraded image is now given by

$$g(x, y) = h(x, y) \otimes f(x, y) + \eta(x, y)$$

Where the operator \otimes is the convolution operator.

Taking the 2D fourier transform results in

$$G(u, v) = H(u, v) F(u, v) + N(u, v)$$

In this implementation we assume that noise is zero. This results in the following relation

$$G(u, v) = H(u, v) F(u, v)$$

Hence,

$$F(u, v) = \frac{G(u,v)}{H(u,v)}$$

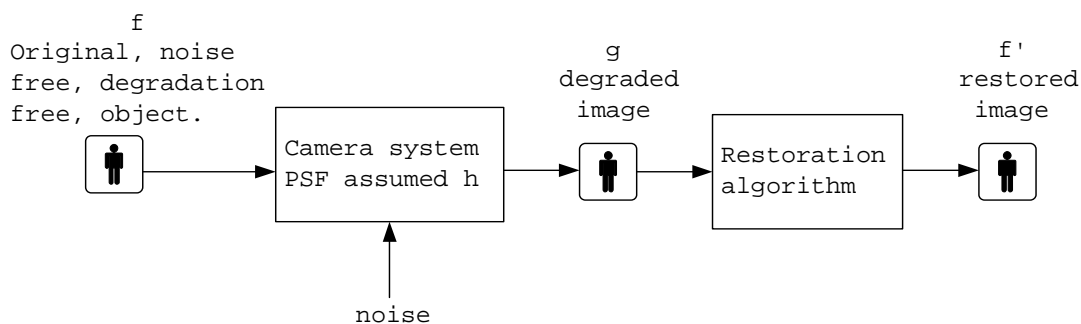
Now, to obtain the restored image, we perform a 2D inverse fourier transform, let the restored image be \hat{f}

$$\hat{f} = \text{Inverse_2D_Fourier_Transform} (F(u,v))$$

Notice that $\hat{f}(x, y)$ (the restored image) will not be exactly the same as $f(x, y)$ for the following reasons

1. We do not know exactly what the camera PSF function is, we will assume here that the PSF is a 2D Gaussian filter with a certain standard deviation or an ideal low pass filter.
2. We did not model the noise in this implementation. Noise is difficult to model and depends on many factors.

The following diagram illustrates this process



PSF model

This is the most important function to select for the restoration, since this is how we assume the degradation has occurred in the first place. The following 2D gaussian filter is assumed for the PSF (this is the PSF expressed in the frequency domain)

$$H(u, v) = e^{-\frac{D^2(u,v)}{2\sigma^2}}$$

For the low pass filter, we use this definition

$H(u, v) = 1$ If $D(u, v) \geq D_0$ else 0 where D_0 is the distance from the center of the spectrum to the point (u,v)

Algorithm steps

The following are the detailed step by step of the algorithm

1. Ask the user for the image file name
2. Read the image to memory to a matrix $f(x, y)$
3. Generate N numbers of $H(u, v)$ (PSF) filters (using different Gaussian standard deviations, or different radius values with a fixed increments). For example, use 3,5,10,20,40 pixels. This results in N PSF matrices $H(u,v)$ call them $H_1(u, v), \dots, H_n(u, v)$
4. Do step 3 for both the Gaussian and the Ideal low pass filters.
5. Multiply the input image $f(x,y)$ by $(-1)^{x+y}$
6. Obtain the 2D fourier transform of the original image, call it $F(u, v)$. Due to step 5, this spectrum is now centered.
7. For each PSF $H(u,v)$, generate 2D Fourier transform of a degraded image using $G_i(u, v) = H_i(u, v) * F(u, v)$

8. For each $G_i(u, v)$ generate the spatial image $g_i(x, v)$ by taking the 2D inverse fourier transform.
9. Take the real part of the image generated in step 8.
10. Multiply the result of step 9 by $(-1)^{x+y}$ to get a centered image.
11. Save each of the degraded images $g_i(x, v)$ to allow more analysis if needed by external programs such as photoshop.
12. Start the restoration pass. For each $G_i(u, v)$ obtain the restored fourier transform $\hat{F}_i(u, v) = \frac{G_i(u,v)}{H_i(u,v)}$
13. For each $\hat{F}_i(u, v)$ apply the 2D IDFT to obtain the restored image $\hat{f}_i(x, y)$
14. Save these images to disk for analysis.
15. Examine visually each of the images $\hat{f}_i(x, y)$ and comment of the quality of restoration by comparing them to the original image $f(x,y)$.

Implementation issues

The main issue to handle in the implementation of the algorithm is the step when we divide $F(u, v) = \frac{G(u,v)}{H(u,v)}$. This is because H can be zero if we use a large standard deviation for the Gaussian. This problem was resolved by explicitly checking for a zero value in the denominator. When this is detected, we set the corresponding value in $F(u,v)$ to 0. This has the effect of setting those high frequencies to zero. Since it is noise that usually occupies the high frequency parts of the image spectrum, this should not cause significant problems in the restoration.

Output, Tables and Results

Conclusions

For small radius values, using the Gaussian low pass filter resulted in an acceptable restoration. The ideal low pass filter however produced no restoration that can be detected.

This can be explained by the fact that an ideal low pass filter is not a causal filter and do not occur naturally.

We notice that for a small radius, more image power will be lost in the degradation process, since image power is mostly concentrated in small circles around the center of the 2D spectrum. As the radius is increased, the effect of the restoration decreased until at about radius 100 pixels, there was no restoration that can be noticed.

This project shows that the choice of restoration PSF is critical. It is not possible to use a generic PSF function to restore different degraded images with without having some knowledge of the cause of the degradation to be able to model a PSF which will best restore the image.

If one is able to estimate the PSF, and can ignore the noise, then this algorithm becomes attractive due to its speed when performed in the frequency domain by utilizing the Fast Fourier Transform and due to its simplicity.

Future work and possible extensions to this project

The following are possible options that this work can be expanded on.

1. Investigate other PSF models such as the Butterworth filter.
2. Investigate how to do restoration with the presence of noise by the use of such filters as minimum mean square error (Wiener) or the constrained least square filtering method. These methods are more mathematically complicated, but

are considered to produce better restoration results.

3. Investigate the restoration of motion blur.

Sample run and output

Here, I show a complete typical run output of this program. This output will use both the Gaussian and Ideal low pass filters.

Start by initialization of the workspace and loading the needed packages

```
Remove["Global`*"];
<< nma.m
<< ImageProcessing`
<< Graphics`Graphics3D`
<< Geometry`Rotations`
```

define the PSF function

```
(* this is actually distance square *)
d[u_, v_, nRow_, nCol_] := Round[N[(u - (nRow/2))^2 + (v - (nCol/2))^2]]

(*This is the Gaussian low pass filter *)

g2D[u_, v_, σ_] := e- $\frac{\text{distanceMatrix}[u,v]}{2\sigma^2}$ 

(*THIS is an ideal low pass filter *)
gIdealLowPass[D0_, u_, v_, nRow_, nCol_] := Module[{response},
  If[Round[N[Sqrt[d[u, v, nRow, nCol]]]] ≤ D0, response = 1, response = 0];
  response
]
```

Display PSF using standard deviation of 6 and 40 pixels just for illustration purposes. We will use $\sigma=6$ here.

```

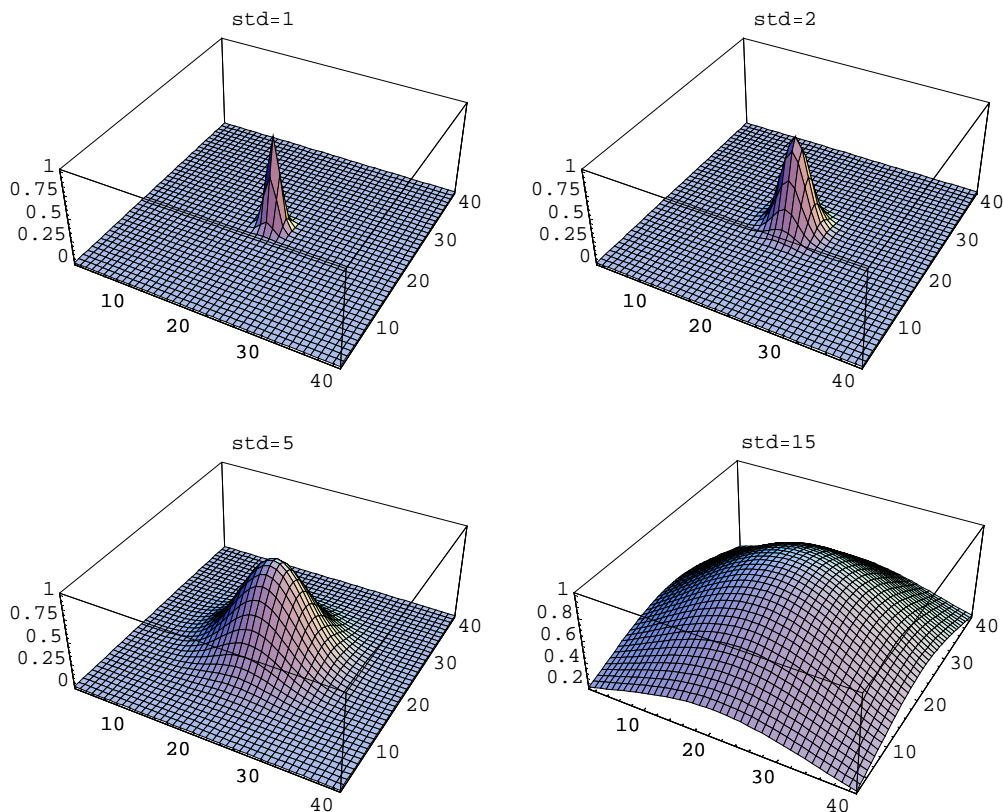
Block[{$DisplayFunction = Identity, nPoints = 20},
  p1 = ListPlot3D[
    Table[N[g2D[u, v, 1]], {u, Round[nRow/2] - nPoints, Round[nRow/2] + nPoints},
      {v, Round[nCol/2] - nPoints, Round[nCol/2] + nPoints}],
    PlotLabel -> "std=1", PlotRange -> All];

  p2 = ListPlot3D[
    Table[N[g2D[u, v, 2]], {u, Round[nRow/2] - nPoints, Round[nRow/2] + nPoints},
      {v, Round[nCol/2] - nPoints, Round[nCol/2] + nPoints}],
    PlotLabel -> "std=2", PlotRange -> All];

  p3 = ListPlot3D[
    Table[N[g2D[u, v, 5]], {u, Round[nRow/2] - nPoints, Round[nRow/2] + nPoints},
      {v, Round[nCol/2] - nPoints, Round[nCol/2] + nPoints}],
    PlotLabel -> "std=5", PlotRange -> All];

  p4 = ListPlot3D[
    Table[N[g2D[u, v, 15]], {u, Round[nRow/2] - nPoints, Round[nRow/2] + nPoints},
      {v, Round[nCol/2] - nPoints, Round[nCol/2] + nPoints}],
    PlotLabel -> "std=15", PlotRange -> All];
]
Show[GraphicsArray[{{p1, p2}, {p3, p4}}]]

```



Now, set the working directory to be the same working directory as this note book to allow easy input of file names for the degraded images.

```

nma`cd
Directory[]

```

Now ask the user for the image file name, and read it to memory

```
(*fileName=Input["Please enter the degraded image file name"];*)
```

sample a 2D continuous gaussian to obtain a discrete version of Gaussian 2D, to use as a filtering window to convolve the image with

```

fileName = "triangle.jpg";
fileName = "lena_gray_blur_gaussian_6";
fileName = "test";
fileName = "lena_gray";

format = "JPG";
img = ImageRead[fileName <> "." <> format];
Show[Graphics[img]];
nColor = ImageDepth[img];
{nRow, nCol} = ImageDimensions[img];
Print["number of colors is " <> ToString[nColor] ];
Print["image size in pixels is " <>
  ToString[nRow] <> " rows x " <> ToString[nCol] <> " cols"];
Print["Other image information " <> ToString[img[[2]]] <>
  " " <> ToString[img[[3]]]];
Print["Displaying the bottom right corner 5x5 image area..."];
f = Chop[N[ img[[1]] ]];
Dimensions[f]
fsaved = f;
Print[MatrixForm[Take[f, {1, 5}, {1, 5}]]];
hist = ImageHistogram[img, {0, 255}];
ShowImageHistogram[hist, PointStyle -> None];

distanceMatrix = Table[d[u, v, nRow, nCol], {u, 1, nRow}, {v, 1, nCol}];

```



```
number of colors is 1
```

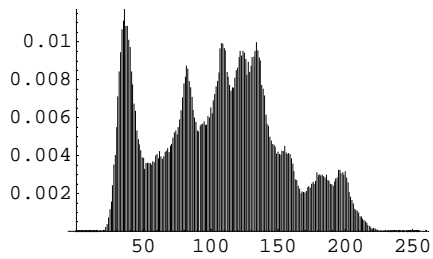
```
image size in pixels is 512 rows x 512 cols
```

```
Other image information PixelInterleave -> None ColorFunction -> GrayLevel
```

```
Displaying the bottom right corner 5x5 image area...
```

```
{512, 512}
```

```
( 31. 32. 42. 35. 41. )  
( 30. 31. 40. 35. 42. )  
( 32. 31. 37. 34. 40. )  
( 35. 34. 37. 34. 38. )  
( 37. 37. 39. 36. 38. )
```



```
(* G E N E R A T E   P S F   *)

(*radius={5,10,15,40,80,140}; *)
radius = { 1, 2, 3, 4, 5, 10, 20, 40, 80};
(*radius={ 180,200,Round[nRow/2]}; *)

(*,30,80,230,Round[nRow/2]};*)
Hgaussian =
  Table[g2D[u, v, radius[[i]], {u, 1, nRow}, {v, 1, nCol}, {i, 1, Length[radius]}];
Hgaussian = Chop[N[Hgaussian]];

Hideal = Table[gIdealLowPass[radius[[i]], u, v, nRow, nCol],
  {u, 1, nRow}, {v, 1, nCol}, {i, 1, Length[radius]}];

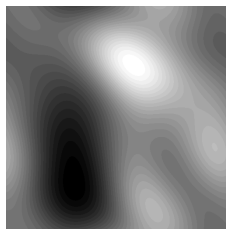
Print["Restoration using Gaussian Low Pass Filter"];
degradeAndRestore[radius, f, Hgaussian, fileName, format];

Restoration using Gaussian Low Pass Filter
```

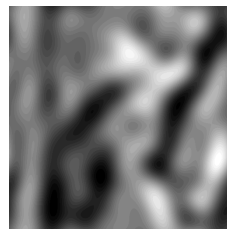
original, r=1



degraded1



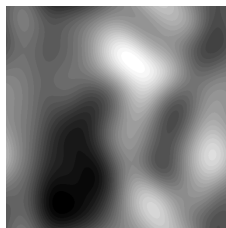
restored



original, r=2

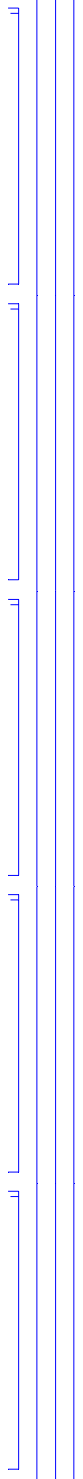


degraded2



restored

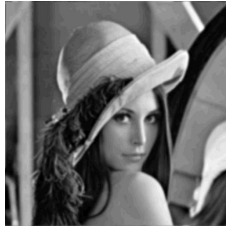




original, r=40



degraded40



restored



original, r=80



degraded80



restored



```
Print["Restoration using ideal Low Pass Filter"];
degradeAndRestore[radius, f, Hideal, fileName, format];

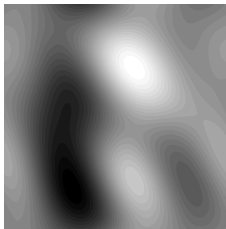
(*restoreOnly[radius, f, H, fileName, format];*)
```

Restoration using ideal Low Pass Filter

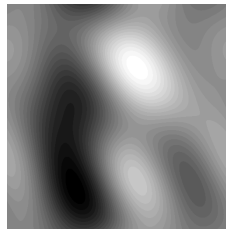
original, r=1



degraded1



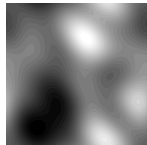
restored



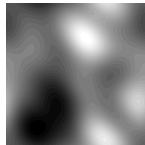
original, r=2



degraded2



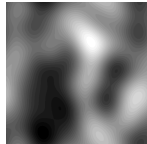
restored



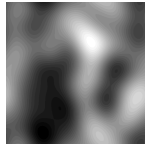
original, r=3



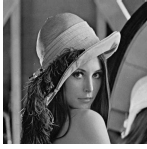
degraded3



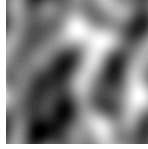
restored



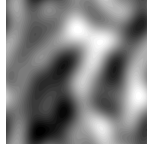
original, r=4



degraded4



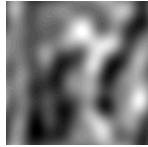
restored



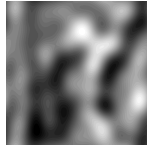
original, r=5



degraded5



restored



original, r=10



degraded10



restored



original, r=20



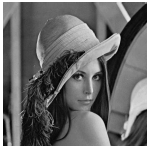
degraded20



restored



original, r=40



degraded40



restored



original, r=80



degraded80



restored



```

degradeAndRestore[radius_, f_, H_, fileName_, format_] := Module[{nH, i},
  nH = Length[radius];
  For[i = 1, i ≤ nH, i = i + 1,
    {
      degradeAndRestore2[radius[[i]], fileName, format, H[[All, All, i]], f];
    }
  ]
];

```

```

restoreOnly[radius_, g_, H_, fileName_, format_] := Module[{nH, i},
  Print["Entering restoreOnly"];
  nH = Length[radius];
  For[i = 1, i ≤ nH, i = i + 1,
    {
      restoreOnly2[radius[[i]], fileName, format, H[[All, All, i]], g];
    }
  ]
]

restoreOnly2[gr_, fileName_, format_, H_, g_] :=
Module[{nRow, nCol, pixs, gName, F, G, gtmp, GG, m, ff, ftmp, h},
  Print["Entering restoreOnly2 "];
  {nRow, nCol} = Dimensions[g];

  gtmp = nma`centerImage[g];
  G = DFT[gtmp];

  h = IDFT[H];
  h = Chop[N[Re[h]]];
  h = nma`centerImage[h];

  (* R E S T O R A T I O N *)

  GG = G;
  (* divide G/H *)
  For[i = 1, i ≤ nRow, i = i + 1,
    {
      For[j = 1, j ≤ nCol, j = j + 1,
        {
          If[H[[i, j]] == 0, GG[[i, j]] = 0, GG[[i, j]] = GG[[i, j]] / H[[i, j]]];
        }
      ]
    }
  ];

  ff = InverseDiscreteFourierTransform[GG];
  ff = Re[ff];
  ff = nma`centerImage[ff];

  Show[Graphics[ ToGrayLevel[ff], ImageSize → {nRow, nCol},
    PlotLabel → "ff, restored image, Gaussian radius=" <> ToString[gr]]];

  pixs = 200;
  theImageSize = {pixs, pixs};
  (*theImageSize= Automatic;*)
  Block[{$DisplayFunction = Identity},
    p0 =
      Show[Graphics[ToGrayLevel[g], ImageSize → {pixs, pixs}, PlotLabel → "f,Max=" <>
        ToString[Floor[Max[ff[[1]]]]] <> " Min=" <> ToString[Floor[Min[f[[1]]]]]];

```

```

p1 =
  Show[Graphics[ToGrayLevel[h]], ImageSize → theImageSize, PlotLabel → "h, PSF"];
gName = "g, Gaussian=" <> ToString[gr];

p2 = Show[Graphics[ToGrayLevel[ff]], ImageSize → theImageSize,
  PlotLabel → "ff, Max=" <> ToString[Floor[Max[ff[[1]]]]] <>
  " Min=" <> ToString[Floor[Min[ff[[1]]]]]];

p01 = Show[Graphics[ToGrayLevel[2 Log[1 + Abs[G]]]],
  ImageSize → theImageSize, PlotLabel → "2 Log[Abs[F]"];

p02 = Show[Graphics[ToGrayLevel[2 Log[1 + Abs[H]]]],
  ImageSize → theImageSize, PlotLabel → "2 Log[Abs[H]"];

p03 = Show[Graphics[ToGrayLevel[2 Log[1 + Abs[GG]]]],
  ImageSize → theImageSize, PlotLabel → "2 Log[Abs[GG]"];

];
Show[GraphicsArray[{p0, p1, p2}], Frame → False];
Show[GraphicsArray[{p01, p02, p03}], Frame → False];

outFileName =
  fileName <> "_RESTORED_gaussian_radius_" <> ToString[gr] <> "." <> format;
ImageWrite[outFileName, ff, format]
(*Print["Written restored image to file "<>outFileName];*)

];

degradeAndRestore2[gr_, fileName_, format_, H_, f_] :=
Module[{nRow, nCol, pixs, gName, F, G, g, GG, m, ff, ftmp, h},

  {nRow, nCol} = Dimensions[f];

  (* D E G R A D E *)

  ftmp = nma`centerImage[f];
  F = DiscreteFourierTransform[ftmp];
  G = H * F;
  g = InverseDiscreteFourierTransform[G];
  g = Re[g];
  g = nma`centerImage[g];

  h = IDFT[H];
  h = Chop[N[Re[h]]];
  h = nma`centerImage[h];

  (* R E S T O R A T I O N *)

```

```

GG = G;
(* divide G/H *)
For[i = 1, i ≤ nRow, i = i + 1,
{
  For[j = 1, j ≤ nCol, j = j + 1,
  {
    If[H[[i, j]] == 0, GG[[i, j]] = 0, GG[[i, j]] = GG[[i, j]] / H[[i, j]]];
  }
]
}
];

ff = InverseDiscreteFourierTransform[GG];
ff = Re[ff];
ff = nma`centerImage[ff];

Block[{$DisplayFunction = Identity},
p0 = Show[Graphics[ToGrayLevel[f]], PlotLabel → "original, r=" <> ToString[gr]];
p1 = Show[Graphics[ToGrayLevel[g]], PlotLabel → "degraded" <> ToString[gr]];
p2 = Show[Graphics[ToGrayLevel[ff]], PlotLabel → "restored"];
];
Show[GraphicsArray[{p0, p1, p2}], Frame → False];

outFileName =
  fileName <> "_RESTORED_gaussian_radius_" <> ToString[gr] <> "." <> format;
(*Print["Creating file ",outFileName];*)
ImageWrite[outFileName, ff, format];

outFileName =
  fileName <> "_DEGRADED_gaussian_radius_" <> ToString[gr] <> "." <> format;
(*Print["Creating file ",outFileName];*)
g = nma`centerImage[g];
g = DFT[g];
g = IDFT[g];
g = Re[g];
g = nma`centerImage[g];
ImageWrite[outFileName, ToGrayLevel[g], format];
]

```

Appendix

References

1. Digital Image Processing, second edition, by Gonzalez and Woods
2. Algorithms for image processing and computer vision, by J.R.Parker
3. Lecture notes, EECS 207A by Professor Meyer, UCI Electrical Engineering Department.