

# IMAGE DEBLURRING - COMPUTATION OF CONFIDENCE INTERVALS

VIKTORIA TAROUDAKI

tarvic@math.umd.edu

AMSC Program, University of Maryland, College Park

Advisor: Prof.Dianne P. O'Leary

oleary@cs.umd.edu

Professor, Computer Science Department  
and Institute for Advanced Computer Studies  
University of Maryland

Fall Semester 2010

## **Abstract**

Pictures that cameras take can be blurred images of the original object. The restoration- deblurring of the image is not a trivial procedure and it can be very expensive depending on the size of the image. In this project we are trying to efficiently compute confidence intervals for the digital values that represent the image and visualize them so that the viewer can distinguish truth from uncertainty.

# 1 Project Background- Introduction

People always wanted to keep snapshots of their everyday life for reference at a later time or for research and educational purposes. Cavemen drew on the walls of the caves using colors made from nature. Later artists painted their houses, graves and other buildings, objects or paintings with various scenes. More recently, cameras were invented, first engraving, then analog cameras and at last digital cameras. In none of these cases is the object represented exactly in the image. But as technology progresses, the accuracy of the representation increases. Digital cameras give us very good representations of the true image but due to the procedure that the image passes through, blurring occurs. This blurring can be caused by the machine errors in transforming the image into data in the camera and from the background and the way of taking the picture. Having clear images is not a luxury. Sometimes it is a matter of life and death, like in surgeries where the doctor needs to know exactly where to operate, or in weather forecasts.

An image is divided into pixels which have several values which denote the color of that pixel. A grayscale image, which we will use for simplicity, has one value for each pixel, an integer in the interval  $[0, 255]$ . 0 is the black color and 255 is the white color. Blurring occurs when the values of the pixels are distorted. In this project, we will assume that this distortion is caused by a linear transformation from the camera.

We will use the following notation:

Symbol	Size	Explanation
$K$	$m \times n$	Matrix defined through the Point Spread function (PSF) in the case of a linear problem
$X$		Original Clear Image
$x$	$n \times 1$	Vector containing the values corresponding to the pixels of the image $X$
$B$		The blurred image we measure
$b$	$m \times 1$	Vector which contains the values of the pixels of the blurred image $B$
$e$	$m \times 1$	Noise Vector

With the above notation, the model of the blurred image is described by the equation:

$$b = Kx + e.$$

In general, the goal is given the vector  $b$  and the matrix  $K$  and also given a distribution for the noise such that the mean value is 0 and the variance is a nonsingular matrix

$S^2$ , we need to compute confidence intervals (i.e. intervals in which the values of the object we calculate fall into the intervals with a certain confidence) for the quantities  $\varphi_k^* = w_k^T x$  for  $k = 1, 2, \dots, p$ , where  $w_k$  are given vectors. If  $w_k$  is a column from the identity matrix, then we obtain a confidence interval for a single pixel.

Two different types of confidence intervals are of interest.

**Definition 1** *The one-at a time confidence intervals bound each  $\varphi_k^*$  individually probability  $\alpha$  ( $\alpha\%$  confidence).*

$$Pr\{l_k \leq \varphi_k^* \leq u_k\} = \alpha, k = 1, 2, \dots, p$$

**Definition 2** *The simultaneous confidence intervals bound all the  $\varphi_k^*$  simultaneously with a probability greater than  $\alpha$ .*

$$Pr\{l_k \leq \varphi_k^* \leq u_k, k = 1, 2, \dots, p\} \geq \alpha$$

O'Leary and Rust [?] have proven the following theorems.

**Theorem 3** *Suppose that the noise is normally distributed, and that  $\hat{x}$  is an unbiased estimate of the true solution  $x$ . Then, given  $\alpha$  in  $(0, 1)$ , there is a  $100\alpha\%$  probability that the true value of  $w_k^T x$  is contained in the interval  $[l_k, u_k]$  where*

$$l_k = \min_x \{w_k^T x : \|K(x - \hat{x})\|_S^2 = \kappa^2\}$$

and

$$u_k = \max_x \{w_k^T x : \|K(x - \hat{x})\|_S^2 = \kappa^2\},$$

where  $\|K(x - \hat{x})\|_S^2 = [K(x - \hat{x})]^T S^{-2} [K(x - \hat{x})]$  and  $\kappa$  is such that  $\alpha = \int_{-\kappa}^{\kappa} n(x; 0, 1) dx$  where  $n(x; 0, 1)$  is the normal distribution of  $x$  with mean value 0 and variance 1.

Since we know that the true pixel values lie between 0 and 255, we want to use this information to reduce the length of the confidence intervals.

**Theorem 4** *With the same assumptions as before and also assuming that  $x$  is nonnegative and less than 255, as well as that the matrix  $S$  is nonsingular and symmetric, the computation of the lower and upper bounds of the confidence intervals have also been done. In [?] we are given that: "Under the above assumptions, the probability that  $\varphi$  is contained in the interval  $[l_k, u_k]$  is greater than or equal to  $\alpha$  where*

$$l_k = \min\{w_k^T x : \|Kx - b\|_S \leq \mu, 0 \leq x \leq 255\}$$

and

$$u_k = \max\{w_k^T x : \|Kx - b\|_S \leq \mu, 0 \leq x \leq 255\},$$

$rank(K) = q$ ,  $\int_0^{\gamma^2} \chi_q^2(\rho) d\rho = \alpha$ ,  $r_0 = \min_{x \geq 0} \|Kx - b\|_S^2$ ,  $\mu^2 = r_0 + \gamma^2$  and  $\chi_q^2$  is the probability density function for the chi-squared distribution with  $q$  degrees of freedom".

Other tools like those using Chebyshev's Inequality are useful for problems where the noise is not normally distributed, but this is not the main purpose of this project and so it will not be examined here unless there is some more time at the end than expected.

## 2 Approach

### 2.1 Point-Spread Function and Blurring Matrix $K$

In general, the matrix  $K$  can be experimentally measured using point spread functions for each one of the pixels of the original image. An easy way to do this is by constructing an artificial image which contains only one white point (of value 255) in the target pixel, say the  $(i, j)$  pixel of the image  $X$ , or the  $(j - 1) \cdot m + i$  element of the vector  $x$  corresponding to that clear image and black anywhere else (value 0). We consider this as a clear image and we blur it the same way as we would blur the original image (or the vector corresponding to the original). Then, we measure the blurred vector  $b$  which corresponds to the blurred image we take corresponding to the artificial image. This vector  $b$  is going to be the  $(j - 1) \cdot m + i$  column of the matrix  $K$  and it's a vector corresponding to the Point Spread Function of the blur for the  $(i, j)$  pixel of the image  $X$ , or the  $(j - 1) \cdot m + i$  element of the vector  $x$ .

If we know that the blur is spatially invariant, and it usually is as it affects only neighboring points of the source point, then having measured only one column of the blurring matrix  $K$  is enough to determine the whole matrix as the rest of the columns of  $K$  are simply going to be some displacement of that one column.

If the blur is spatially variant, we need to move the source point to all the pixels of the image and measure the blur to compute all the columns of the blurring matrix but we will not consider this case in this project. For more information, someone could consult [?].

For the purposes of this project, the blurring matrix  $K$  was constructed using spatially invariant blur and Gaussian Point Spread Functions. As usually these Point Spread Functions are of much smaller size than the original image. We used Point Spread Functions of size  $5 \times 5$  and some parameters  $s_1 = s_2 = 3$ . So, for  $k, l = 1 \dots 5$ , we get that

$$PSF(k, l) = \exp\left(-\frac{1}{2} \frac{(k - c_1)^2}{s_1^2} - \frac{1}{2} \frac{(l - c_2)^2}{s_2^2}\right)$$

where  $c_1$  and  $c_2$  are the coordinates of the center of the Point Spread Function which

for a Point Spread Function which corresponds to the pixel  $(i, j)$  are equivalent to  $i$  and  $j$  respectively. Appropriate displacement follows.

An example of this procedure is given for an image of size  $5 \times 5$  and a Point Spread Function of size  $3 \times 3$ . (The size of the PSF in this example is different from the one used in the codes of the project).

Let the PSF array be a matrix of the form:

$\times$	$\times$	$\times$
$\times$	$\times$	$\times$
$\times$	$\times$	$\times$

where the red denotes the center of the matrix and the  $\times$  denotes non zero elements. Then the Point Spread Function for the first pixel of the image will affect only the pixels which are immediate neighbors. That means that the blurred image of the artificially made having a white color (255) at the first pixel and black (0) everywhere else will

look like

$\times$	$\times$	0	0	0
$\times$	$\times$	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

and reshaping this, column by column, we get the first column of the blurring matrix  $K$ .

As MatLab is column oriented, we count the pixels column by column, so the second pixel is the one which is in the second row but first column.

So, for the second pixel, the blurred image will look like:

$\times$	$\times$	0	0	0
$\times$	$\times$	0	0	0
0	0	0	0	0
0	0	0	0	0

and reshaping this, column by column, we get the second column of the blurring matrix  $K$ .

If we do this for all the pixels of the image we will end up having the tri-block-diagonal blurring matrix  $K$  shown in figure ??.

## 2.2 Blurring an Image and Constructing Noise

When the blurring matrix  $K$  has been computed, then the clear blurred image, i.e., the blurred image without noise is simply the array which corresponds to the product

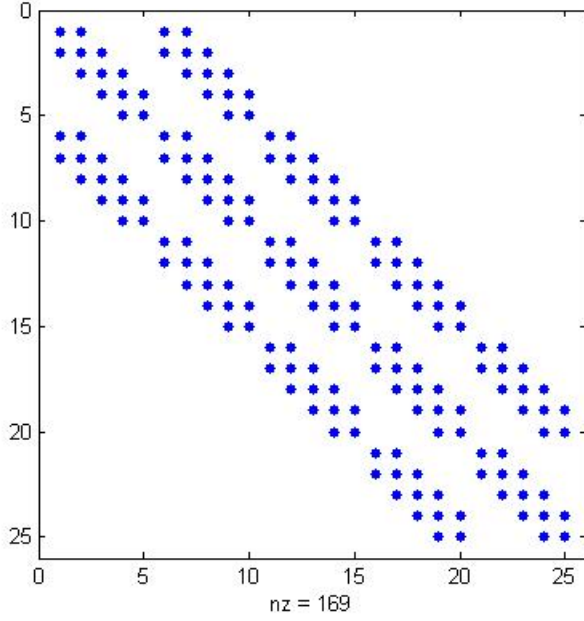


Figure 1: Blurring Matrix  $K$  for an image of size  $5 \times 5$  with Point Spread Functions of size  $3 \times 3$ .

vector:  $Kx$ . But in order to be able to use the theory of the confidence intervals, we need to add noise to the clear blurred image so that we can actually have the standard deviation matrix  $S$  of the noise which also needs to be symmetric and non-singular. To do this, we construct a random vector,  $e$ , of same length as the size of the original image (we use square blurring matrices throughout the project) with elements with mean 0 and without loss of generality, standard deviation a specified number  $sdv$ . The standard deviation matrix in this case is the identity matrix multiplied by the number  $sdv$  which is apparently symmetric and invertible. The noisy blurred image thus corresponds to the sum  $b = Kx + e$ .

### 2.3 Computing the $\mu^2$

By theorem ??, knowing the matrices  $K$  and  $S$  of the blurring function and the standard deviation of the noise as well as the blurred vector  $b$ , we can easily find what the value for  $q = rank(K)$  is. Also using the inverse of the  $\chi^2$  distribution we can find what the  $\gamma^2$  is when  $\int_0^{\gamma^2} \chi_q^2(\rho) d\rho = \alpha$  with  $\alpha$  being the desired probability that defines the confidence intervals. For the purposes of this project, we use  $\alpha = 0.95$ . In addition, we can compute the minimum of a norm:  $r_0 = \min_{x \geq 0} \|Kx - b\|_S^2$  and finally

add those two things up to get  $\mu^2 = r_0 + \gamma^2$ .

The  $\gamma^2$  is computed in MatLab using the command `chi2inv(1-a,q)`. the minimization of the norm, we can use either the `lsqlin` command of MatLab or the `quadprog` which use linear least squares and quadratic programming respectively with the constraints that the image imposes. To do this, the first thing we need to do is to tranform the  $S$ -norm to the 2-norm that MatLab can handle. Thus,

$$\|Kx - b\|_S^2 = (Kx - b)^T S^{-2} (Kx - b) = (S^{-1}(Kx - b))^T (S^{-1}(Kx - b)) = \|S^{-1}Kx - S^{-1}b\|^2$$

The matrices and vectors manipulated by `lsqlin` are the same of the problem. For `quadprog`, we need to modify the matrices and vectors to get the appropriate  $H$  and  $f$  that it takes as input.

## 2.4 Lower and Upper Bounds of Confidence Intervals

So, now with this information, we need to identify the confidence intervals whose bounds are given by the following equations:

$$l_k = \min\{w_k^T x : \|Kx - b\|_S \leq \mu, 0 \leq x \leq 255\}$$

$$u_k = \max\{w_k^T x : \|Kx - b\|_S \leq \mu, 0 \leq x \leq 255\}.$$

O'Leary and Rust ([?]) have proven the following theorem:

**Theorem 5** *The values  $l_k$  and  $u_k$  are defined by the two extreme roots of  $L(\varphi) - \mu^2 = 0$  where  $L(\varphi) = \min_x \{\|Kx - b\|_S^2 : x \geq 0, w_k^T x = \varphi\}$*

The solution of the minimization of a norm problem can be solved in MatLab calling the `lsqlin` or the `quadprog` functions and transforming the data each time as described in the previous section. The main idea here was to make a function of  $\varphi$  that finds the minimum of the norm for all  $x$  with the constraints of the image, i.e.,  $0 \leq x_i \leq 255$  for all  $i$ . The output of this, the minimum norm, would be a function of  $\varphi$  as well which we would like to set equal to the parameter  $\mu$  and find the zeros. This would give us values for  $\varphi$  where  $\varphi = w^T x$ . If, as in this project,  $w$  are the columns of the identity matrix, then for each one of these, we get the value of one pixel. The lower and upper bounds of the confidence intervals will then give us the lower and the upper limits of the value of each one of the pixels with the probability that we used to compute the  $\mu$ .

## 2.5 Sub-images and Sub-matrices

In this project we will use a function  $K$  which will be given by a square matrix. This matrix can be very large depending on the size of the image. To reduce the expense, we need to partition it into sub-matrices and so the whole problem into  $p$  smaller problems which can be solved individually. These sub-problems are much easier to solve. This will be done using parallel computing.

Let's consider again the problem:  $b = Kx$  where  $K$  is the  $n \times n$  PSF matrix,  $x$  is the vector corresponding to the original image and  $b$  the vector corresponding to the blurred image. If we want to make a sub-problem of size  $r \times c$  ( $r$  rows and  $c$  columns on the sub-image defining the sub-problem), we proceed as follows. Using a matrix  $E$  with  $n$  rows and  $rc$  columns, with columns that are unit vectors corresponding to the pixels in the sub-image, and a matrix  $\bar{E}$  that corresponds to the other  $n - rc$  unit vectors we have:

$$E^T b = E^T K x = (E^T K [E \bar{E}]) \left( \begin{bmatrix} E^T \\ \bar{E} \end{bmatrix} x \right) = E^T [\hat{K}_s \hat{K}_t] \begin{bmatrix} x_s \\ x_t \end{bmatrix} = [K_s K_t] \begin{bmatrix} x_s \\ x_t \end{bmatrix} = K_s x_s + K_t$$

where  $x_s$  is the vector corresponding to the sub-image of the original image. Ignoring the second term of the right hand side we have that  $b_s = K_s x_s$  which is a sub-problem we need to solve.  $E$  is a matrix of unit vectors that we choose but in this project we will use the matrix which is part of the identity matrix corresponding to the sub-problem, i.e., if we have  $x_s = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ , then we will take  $E$  to be the first two columns of the identity matrix with size  $n \times n$  so that the matrix  $K_s$  is going to be  $2 \times 2$ .

The methods used in the introduction can now be applied to these smaller problems. We compute the confidence intervals for these. To verify that the computed intervals are correct, we can take a blurred image and deblur it. We can have many different samples of deblurred images. Then we can display these deblurred images and see if the values of the pixels of the deblurred image are in the confidence interval with a probability that is defined by the construction of the confidence interval.

## 3 Implementation

The implementation of the codes will be done in MatLab. There are some already developed MatLab programs that may be used and are included in the Image Processing toolbox. These are mostly input, output and display tools. Some commands of this



type are the `imshow(A)` which displays the image A, the `im2double(A)` which converts the image A from `uint8` to `double` etc. Some of these commands can be replaced by others which are provided by the general MatLab but then, different operations should be done to obtain the same result. For example, the command `double(A)` also transforms a `uint8` image to a `double` image but then the values are from 0 to 255 whereas the values from the `im2double(A)` are from 0 to 1. The problem then is what we use to turn these values back into an image. When we are dealing with a grayscale image, then `imshow(A)` returns white for every value that is greater or equal to 1. That means that we will either use the floating point arithmetic from 0 to 1 to visualize the image or we will need to change accordingly the integer values. Similar outcomes come from the displaying tools of MatLab `image(A)` and `imagesc(A)`. Appropriate values need to be used and they do not always coincide.

The problem involves matrices and vector operations which are easy to be handled using the available linear algebra tools of MatLab. These matrices and vectors can be of very small sizes or of big sizes depending on the original image and its size. Because of that, particular attention was used so that MatLab can control relatively big problems. For this, we had to take into account how operations are done in MatLab and how matrices and vectors are allocated in memory. MatLab cannot handle every size of initial image though. The limit is set by the MatLab memory. So one goal was to minimize as much as possible the variables that are saved but without having to recompute variables or other vectors and matrices all the time so that we reduce the running time and any machine and floating point arithmetic errors that may come up.

The problem can also be divided into smaller in size problems which can be solved more easily. If the blur is spatially invariant, these sub-problems involve the same matrix. These sub-problems can be solved using parallel computing. So even if MatLab's optimization tools may be used, they will need to be modified so that they can also be used in parallelizing the problem. Parallel computing will be useful to handle bigger images in about the same time that MatLab needs for a smaller image or to have results much faster for images of size of the same order in parallel and in MatLab. This will be attempted next semester as the project schedule suggests.

## 4 Databases

A big image database is the USC-SIPI Image Database which belongs to the Signal and Image Processing Institute of the University of Southern California and can be accessed here: <http://sipi.usc.edu/database/>. The database includes grayscale and color images saved in TIFF- format. They are of different sizes,  $256 \times 256$ ,  $512 \times 512$  and  $1024 \times 1024$ . The grayscale pictures have 8 bits/ pixel whereas the color images

have 16 bits/pixel.

Also, a variety of image databases can be found through the Image Processing Place by the link: [http://www.imageprocessingplace.com/root\\_files\\_V3/image\\_databases.htm](http://www.imageprocessingplace.com/root_files_V3/image_databases.htm)

The images that have been used so far for the purpose of the project are grayscale images of various sizes. For validation, it was better to use small images that need less memory allocation and less running time. Bigger images produce huge blurring matrices that need care when used in MatLab. For this reason, the images were of size  $16 \times 16$ . The maximum size of the images that can be used by the code are determined by the memory that MatLab can handle.

An example of a test image is the firework image (figure ??) which was cropped (figure ??) and used at the initial steps of validating the code.

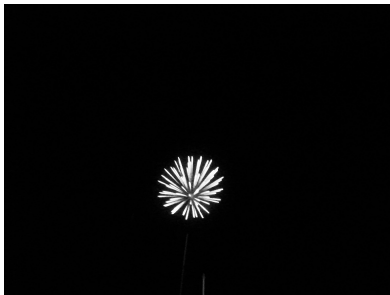


Figure 2: The original  $3648 \times 2736$  image



Figure 3: The cropped  $16 \times 16$  image

## 5 Validation

In order to validate the code, we need to run the program using data which when used, give us a known result. As data, we mean blurred images that we know their origin, i.e., the clear image. In detail, we take some images which are considered as the clear images. The confidence intervals that we compute should give us images that approximate these clear images. Then, we blur these images to obtain the input in our code. Noise is also to be added to the blurred images. Then, deblurring of the images takes place. We also compute the confidence intervals using our code and we count how many samples fall within the intervals. As the same code is going to be used to compute the 95% confidence intervals for all the pixels in the image, it is enough that close to 95% of the pixel values of the same image are between the computed values. If not, we should check every part of the code and the correspondence of the theory as well as some special circumstances which may be developed from the image and affect the outcomes. The images that are to be used are taken from the databases mentioned above or are cropped images from other bigger images that can also be obtained from these databases. For the moment, we use images of size  $16 \times 16$ . Bigger images were also used but the running time indicated that smaller images were better for validation and tests.

Also, to validate the code in general, each of its parts was run independently with input artificially made data so that the expected results are known.

More specifically, to validate the construction of the blurring matrix, the spy command of MatLab was used to visualize the matrix and verify that the structure of the blurring matrix was the anticipated one. In addition, the clear image, the true blurred image and the blurred image infected by noise were shown to see the differences between them and the affect of the blurring matrix. During the construction, the spy command was also used for the Point Spread Function and the Embedded Point Spread Function matrix. To compute the  $\mu^2$  we needed to compute the rank of the blurring matrix, to minimize a norm and find the  $\gamma^2$  of the  $\chi^2$  cumulative distribution function.  $\gamma^2$  can be computed by MatLab using the command `chi2inv` given the rank of the blurring matrix and a probability  $\alpha$ . The result of this can be easily verified using tables of the  $\chi^2$  distribution. Using a known matrix, the verification of the rank of the matrix is easy as we can use a matrix of a desired rank. Finally, the minimization of the norm can be verified using a vector that is simply modified by the product of the matrix we have and the solution vector that we want. The last part of the code is finding the lower and upper bounds of the confidence intervals. This can only be computed when we have constructed and verified all of the above parts of the code. We use a known initial image and at the end we expect to find confidence intervals that contain the

true value of the pixels of the image approximately in 95% of the cases.

Whenever something of the above was not as expected, we went back to the theory to confirm the method used or to find the correct method and looked at each part of the code to verify that the inputs and outputs that we had were the correct and that the theory was implemented correctly. After all the necessary changes, the code was run again with the known data and went over the validation procedure again till the point that the code needed no more changes.

## 6 Testing

A good code should give the right results in a relatively short time without much cost in memory. The time may depend on the size of the image, its format and the way that the values of the pixels are stored. The same affects the cost in memory. Also, for various types of blurring (Point Spread Function) the time for the same image may be different. All of the above are going to be studied and compared. For the moment, the behavior of different sizes of Point Spread Functions has been studied and what is the image that the lower bounds produce as an approximation to the initial image given the blurred image, the blurring matrix  $K$  and the standard deviation of the noise.

In figure ?? there is the figure of the original  $16 \times 16$  cropped fireworks image (left) with its clear blurred image (middle) and the noisy blurred image (right). The blurring occurred using the Gaussian Point Spread Function of size  $5 \times 5$ .

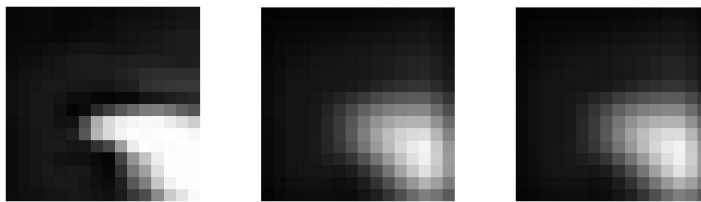


Figure 4: Original image, Clear Blurred Image, Blurred Image with noise

It is not difficult to see that the original image has been faded but still it is clear where the dark or bright color is.

On the contrary, when we use the  $7 \times 7$  image (left) in figure ?? which was blurred using a Gaussian PSF function of size  $5 \times 5$  (middle image) and noise was also added to it (right image), the new distorted image doesn't remind the initial one. The original image had only the middle pixel white and everything else black but the blurred image is more close to a big dark grey square with a black frame.



Figure 5: Original image, Clear Blurred Image, Blurred Image with noise

Nevertheless, the restored image (right image in ??) is a very good approximation of the original one (left).

To further examine this behavior, a smaller test image was constructed. The domino image in figure ?? is an image  $2 \times 2$ . That means that the PSF function of size  $5 \times 5$  would exceed the size of the image. So we constructed several PSF functions of sizes  $1 \times 1$ ,  $2 \times 2$ ,  $3 \times 3$ ,  $4 \times 4$  and  $5 \times 5$ . Only the first 3 are shown below as they illustrate the behavior for all the PSF matrix sizes.

It is obvious that the blurred image is more close to the original when the PSF is of smaller size. In particular, in the first case, the blurred image is the original one and the only alteration comes from the noise which is set to have a standard deviation of 0.01. Also, as long as the PSF function has size less or equal to the size of the original image, the restored image is a good approximation. On the other hand, when the PSF function exceeds the size of the original image, then the restored image does not give us an image like the original one.



Figure 6: Original Image, Recovered Image

## 7 Project Schedule

September	Study the literature, get familiar with the image toolbox and the commands of MatLab for images, write and present project proposal.
October	Study the literature- understand all the aspects of the problem- write code.
November	Write the code and validate it. Prepare and complete Midyear Presentation
Early December	Write midyear report.
Late January	Test various images, begin to exercise on parallel computing.
February	Work on the parallel part of the code.
March	Test various images, validate and correct code if necessary.
April	Validate and correct code if necessary, write final report.
May	Present final report.

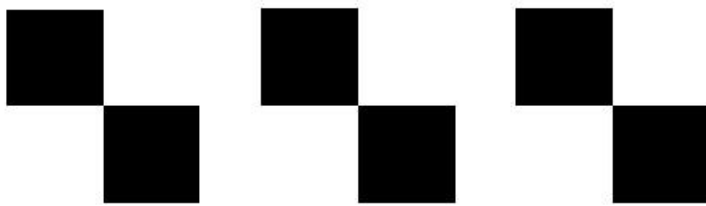


Figure 7: Original image, Clear Blurred Image, Blurred Image with noise



Figure 8: Original Image, Recovered Image



Figure 9: Original image, Clear Blurred Image, Blurred Image with noise



Figure 10: Original Image, Recovered Image





Figure 11: Original image, Clear Blurred Image, Blurred Image with noise



Figure 12: Original Image, Recovered Image

## 8 Milestones

End of September	Having a good understanding of the literature, having used MatLab image toolbox to get familiar with it, having prepared and presented the project proposal.
End of October	Having finished studying the basic literature, having set a database- having started writing the code.
End of November	Having finished the basic writing of the code. (Ideal: Having finished the MatLab part of the code). Having validated it and corrected it if needed. Prepare and Present the work done throughout the semester
Middle of December	Having written the midyear report.
End of January	Having tested several images, getting acquainted with parallel programming.
End of February	Having chosen open MP or MPI and having worked on the parallel computing part of the code. (Ideal: having finished with parallel programming)
End of March	Having tested various images, having validated the code. If in need, having corrected the code (Ideal: by the end of March the code works producing the correct results in little time)
End of April	Having an efficiently working code. Having written the biggest part of the final report.
Middle of May	Having presented the project and turned in code, final report and validation results.

During September, I studied the literature to understand the problem. After meeting with both my advisor and the instructors, I wrote a first draft of the project proposal which was soon finalized to be used for the project proposal presentation. I was also exposed to the Image Processing Toolbox of MatLab (Simple codes for creating and transforming images).

After presenting the project proposal, I continued studying the literature and I started

writing the parts of the code. Till the beginning of November, the code was written and only validation was needed. Validation started at that point and correction of various parts of the code were corrected or optimized for better efficiency of the code.

## References

- [1] Tony F. Chan and Jianhong (Jackie) Shen, *"Image Processing and Analysis"*, SIAM, Philadelphia, 2005
- [2] Martin Hanke, James Nagy and Robert Plemmons, *"Preconditioned Iterative Regularization For Ill-Posed Problems"*, IMA Preprint Series n 1024, 1992
- [3] Per Christian Hansen, James G. Nagy and Dianne P. O'Leary, *"Deblurring Images Matrices, Spectra, and Filtering"*, SIAM, Philadelphia, 2006
- [4] Richard A. Johnson, Gouri K. Bhattacharyya, *"Statistics: Principles and Methods"*, John Wiley & Sons, Inc., 2006
- [5] Jodi L. Mead, Rosemary A Renaut, *"Least squares problems with inequality constraints as quadratic constraints"*, Linear Algebra and its Applications, 432, 2010, p. 1936-1949
- [6] James G. Nagy and Dianne P. O'Leary, *"Restoring Images Degraded By Spatially-Variant Blur"*, SIAM J. Sci. Comput., Vol 19, No 4, 1998, p. 1063-1082
- [7] James G. Nagy and Dianne P. O'Leary, *"Image Restoration through Subimages and Confidence Images"*, Electronic Transactions on Numerical Analysis, 13, 2002, p. 22-37
- [8] Dianne P. O'Leary and Bert W. Rust, *"Confidence Intervals for inequality constrained least squares problems, with applications to ill-posed problems"*, SIAM Journal on Scientific and Statistical Computing, 7, 1986, p. 473-489
- [9] Bert W. Rust and Dianne P. O'Leary, *"Confidence intervals for discrete approximations to ill-posed problems"*, The Journal of Computational and Graphical Statistics, 3, 1994, p. 67-96