Fall 2010

Denoising of Natural Images Using the Wavelet Transform

Manish Kumar Singh
San Jose State University

DENOISING OF NATURAL IMAGES USING THE WAVELET TRANSFORM

A Thesis

Presented to

The Faculty of the Department of Electrical Engineering

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Manish Kumar Singh

December 2010

The Designated Thesis Committee Approves the Thesis Titled

DENOISING OF NATURAL IMAGES USING THE WAVELET TRANSFORM

by

Manish Kumar Singh

APPROVED FOR THE DEPARTMENT OF ELECTRICAL ENGINEERING

SAN JOSÉ STATE UNIVERSITY

December 2010

Dr.	Essam Marouf	Department of Electrical Engineering
Dr.	Chang Choo	Department of Electrical Engineering
Dr.	Mallika Keralapura	Department of Electrical Engineering

ABSTRACT

DENOISING OF NATURAL IMAGES USING THE WAVELET TRANSFORM

by Manish Kumar Singh

A new denoising algorithm based on the Haar wavelet transform is proposed. The methodology is based on an algorithm initially developed for image compression using the Tetrolet transform. The Tetrolet transform is an adaptive Haar wavelet transform whose support is tetrominoes, that is, shapes made by connecting four equal sized squares. The proposed algorithm improves denoising performance measured in peak signal-to-noise ratio (PSNR) by 1-2.5 dB over the Haar wavelet transform for images corrupted by additive white Gaussian noise (AWGN) assuming universal hard thresholding. The algorithm is local and works independently on each 4x4 block of the image. It performs equally well when compared with other published Haar wavelet transform-based methods (achieves up to 2 dB better PSNR). The local nature of the algorithm and the simplicity of the Haar wavelet transform computations make the proposed algorithm well suited for efficient hardware implementation.

ACKNOWLEDGEMENTS

I would like to express my gratitude to Dr. Essam Marouf, Department of Electrical Engineering, San José State University, for his generous guidance, encouragement, direction, and support in completing this thesis. He encouraged and helped me in understanding the subject, without which I would not be able to finish this thesis.

I would also like to express my gratitude to Dr. Chang Choo and Dr. Mallika Keralapura, Department of Electrical Engineering, San José State University, for their generous guidance and support in completing this thesis.

I would also like to extend my special thanks to my wife Kirti and my lovely daughter Sanskriti. Without their support and love this thesis would not have been completed.

Last but not least, my thanks to everyone who participated in the subjective image denoising blind test through the web poll.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Images are often corrupted with noise during acquisition, transmission, and retrieval from storage media. Many dots can be spotted in a Photograph taken with a digital camera under low lighting conditions. Figure 1.1 is an example of such a Photograph. Appereance of dots is due to the real signals getting corrupted by noise (unwanted signals). On loss of reception, random black and white snow-like patterns can be seen on television screens, examples of noise picked up by the television. Noise corrupts both images and videos. The purpose of the denoising algorithm is to remove such noise.

Image denoising is needed because a noisy image is not pleasant to view. In addition, some fine details in the image may be confused with the noise or vice-versa. Many image-processing algorithms such as pattern recognition need a clean image to work effectively. Random and uncorrelated noise samples are not compressible. Such concerns underline the importance of denoising in image and video processing.

Images are affected by different types of noise, as discussed in subsection 1.2. The work presented herein focuses on a zero mean additive white Gaussian noise (AWGN). Zero mean does not lose generality, as a non-zero mean can be subtracted to get to a zero mean model. In the case of noise being correlated with the signal, it can be de-correlated prior to using this method to mitigate it. The problem of denoising can be mathematically presented as follows,

$$Y = X + N$$

where Y is the observed noisy image, X is the original image and N is the AWGN noise with variance σ^2.

The objective is to estimate X given Y. A best estimate can be written as the

(a) Clean Boat Image (b) Noisy Boat Image

Figure 1.1. Illustration of Noise in the Image

conditional mean $\hat{X} = E[X \mid Y]$. The difficulty lies in determining the probability density function $\rho(x \mid y)$. The purpose of an image-denoising algorithm is to find a best estimate of X. Though many denoising algorithms have been published, there is scope for improvement.

1.1 Image Denoising versus Image Enhancement

Image denoising is different from image enhancement. As Gonzalez and Woods [1] explain, image enhancement is an objective process, whereas image denoising is a subjective process. Image denoising is a restoration process, where attempts are made to recover an image that has been degraded by using prior knowledge of the degradation process. Image enhancement, on the other hand, involves manipulation of the image characteristics to make it more appealing to the human eye. There is some overlap between the two processes.

1.2 Noise Sources

The block diagram of a digital camera is shown in Figure 1.2. A lens focuses the light from regions of interest onto a sensor. The sensor measures the color and light intensity. An analog-to-digital converter (ADC) converts the image to the digital signal. An image-processing block enhances the image and compensates for some of the deficiencies of the other camera blocks. Memory is present to store the image, while a display may be used to preview it. Some blocks exist for the purpose of user control. Noise is added to the image in the lens, sensor, and ADC as well as in the image processing block itself.

The sensor is made of millions of tiny light-sensitive components. They differ in their physical, electrical, and optical properties, which adds a signal-independent noise (termed as dark current shot noise) to the acquired image. Another component of shot noise is the photon shot noise. This occurs because the number of photons detected varies across different parts of the sensor. Amplification of sensor signals adds amplification noise, which is Gaussian in nature. The ADC adds thermal as well as quantization noise in the digitization process. The image-processing block amplifies part of the noise and adds its own rounding noise. Rounding noise occurs because there are only a finite number of bits to represent the intermediate floating point results during computations [2].

Most denoising algorithms assume zero mean additive white Gaussian noise (AWGN) because it is symmetric, continuous, and has a smooth density distribution. However, many other types of noise exist in practice. Correlated noise with a Guassian distribution is an example. Noise can also have different distributions such as Poisson, Laplacian, or non-additive Salt-and-Pepper noise. Salt-and-Pepper noise is caused by bit errors in image transmission and retrieval as well as in analog-to-digital converters. A scratch in a picture is also a type of noise. Noise can be signal dependent or signal independent. For example, the process of quantization (dividing a continuous signal into discrete levels)

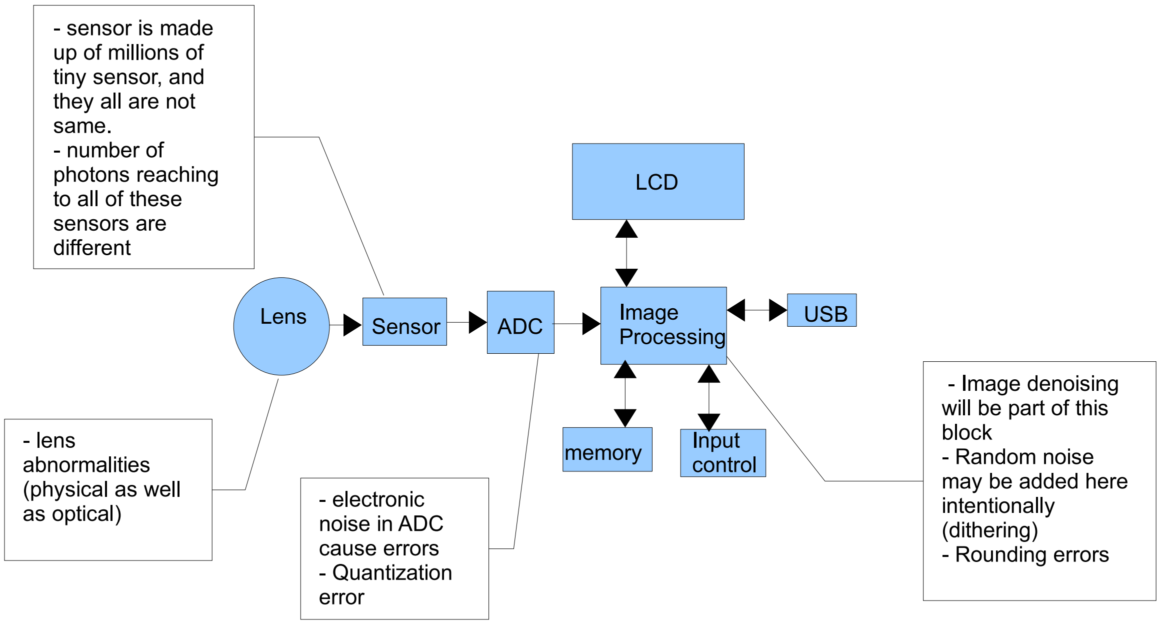


Figure 1.2. Basic Blocks of a Digital Camera and Possible Sources of Noise

adds signal-dependent noise. In digital image processing, a little bit of random noise is deliberately introduced to avoid false contouring or posterization. This is termed dithering. Discretizing a continuously varying shade may make it look isolated, resulting in posterization. The above facts suggest that it is not easy to model all types of practical noise into one model [1]-[2].

This work is also focused on zero mean additive white Gaussian noise (AWGN) due to its generic and simple nature. For correlated noise with a non-zero mean, the zero mean white model can be derived by subtracting the mean after de-correlating the samples.

1.3 Denoising Artifacts

Denoising often adds its own noise to an image. Some of the noise artifacts created by denoising are as follows:

- Blur: attenuation of high spatial frequencies may result in smoothe edges in the image.

- Ringing/Gibbs Phenomenon: truncation of high frequency transform coefficients may lead to oscillations along the edges or ringing distortions in the image.
- Staircase Effect: aliasing of high frequency components may lead to stair-like structures in the image.
- Checkerboard Effect: denoised images may sometimes carry checkerboard structures.
- Wavelet Outliers: these are distinct repeated wavelet-like structures visible in the denoised image and occur in algorithms that work in the wavelet domain.

1.4 The Wavelet Transform in Image Denoising

The goal of image denoising is to remove noise by differentiating it from the signal. The wavelet transform's energy compactness helps greatly in denoising. Energy compactness refers to the fact that most of the signal energy is contained in a few large wavelet coefficients, whereas a small portion of the energy is spread across a large number of small wavelet coefficients. These coefficients represent details as well as high frequency noise in the image. By appropriately thresholding these wavelet coefficients, image denoising is achieved while preserving fine structures in the image.

The other properties of the wavelet transform that help in the image denoising are sparseness, clustering, and correlation between neighboring wavelet coefficients [3]. The wavelet coefficients of natural images are sparse. The histogram of the wavelet coefficients of natural images tends to peak at zero. As they move away from zero, the graph falls sharply. The histogram also shows long tails. Figures 1.3 and 1.4 show examples of such histograms. Wavelet coefficients also tend to occur in clusters. They have very high correlation with the neighboring coefficients across scale and orientation.

All these properties help in differentiating the noise from the signal and enabling its optimal removal.

As Burrus and others [4] have concluded, “The size of the wavelet expansion coefficients $a_{j,k}$ or $d_{j,k}$ drop off rapidly with j and k for a large class of signals. This property is called being an unconditional basis and it is why wavelets are so effective in signal and image compression, denoising, and detection. Here $a_{j,k}$ are average coefficients, $d_{j,k}$ are detailed coefficients, j are scale indices, and k are translation indices.”

Donoho [5]-[6] shows that wavelets are near optimal for compression, denoising, and detection of a wide class of signals.

1.5 Introduction to the Wavelet Transform

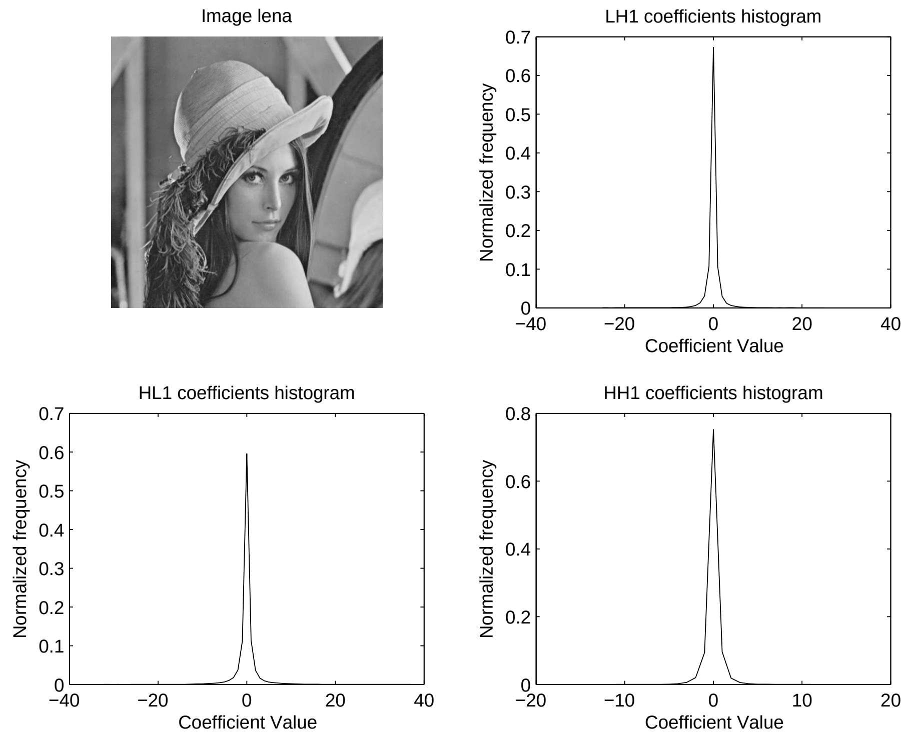
A wave is usually defined as an oscillating function in time or space. Sinusoids are an example. Fourier analysis is a wave analysis. A wavelet is a “small wave” that has its energy concentrated in time and frequency. It provides a tool for the analysis of transient, non-stationary, and time-varying phenomena. It allows simultaneous time and frequency analysis with a flexible mathematical foundation while retaining the oscillating wave-like characteristic. Figure 1.5 shows the difference between a sine wave and a wavelet.

A simple high level introduction to wavelets can be found in the articles by Daubechies et al. [7]-[8].

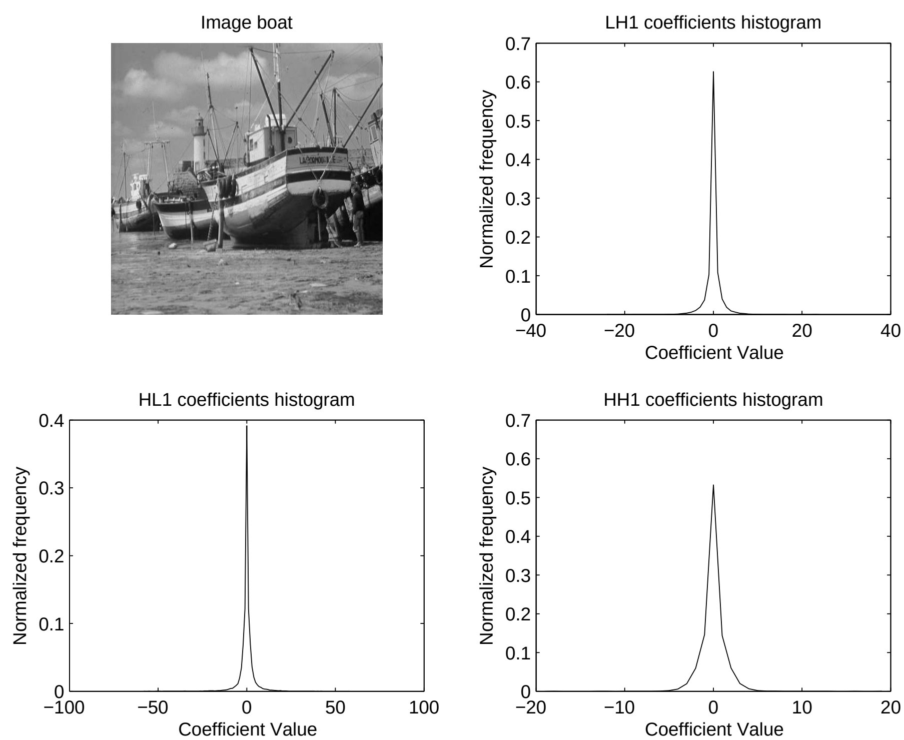
A signal or a function f(t) can often be better analyzed if it is expanded as

$f(t) = \sum_k c_{j0,k}\phi_{j0,k}(t) + \sum_k \sum_{j>j0} d_{j,k}\Psi_{j,k}(t)$

where both j and k are integer indices. $\Psi_{j,k}(t)$ represents the wavelet expansion functions, and $\phi_{j,k}(t)$ represents the scaling functions. They usually form an orthogonal basis. This expansion is termed as wavelet expansion. The term related to the scaling coefficients captures the average or coarse representation of the signal at the scale j0. The

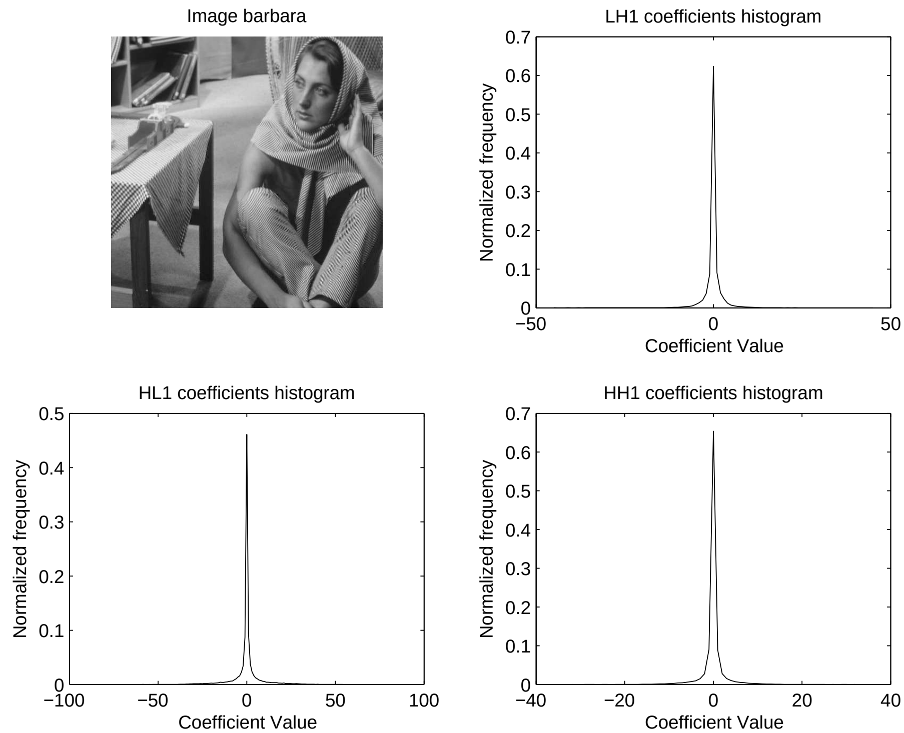


(a) Lena Image

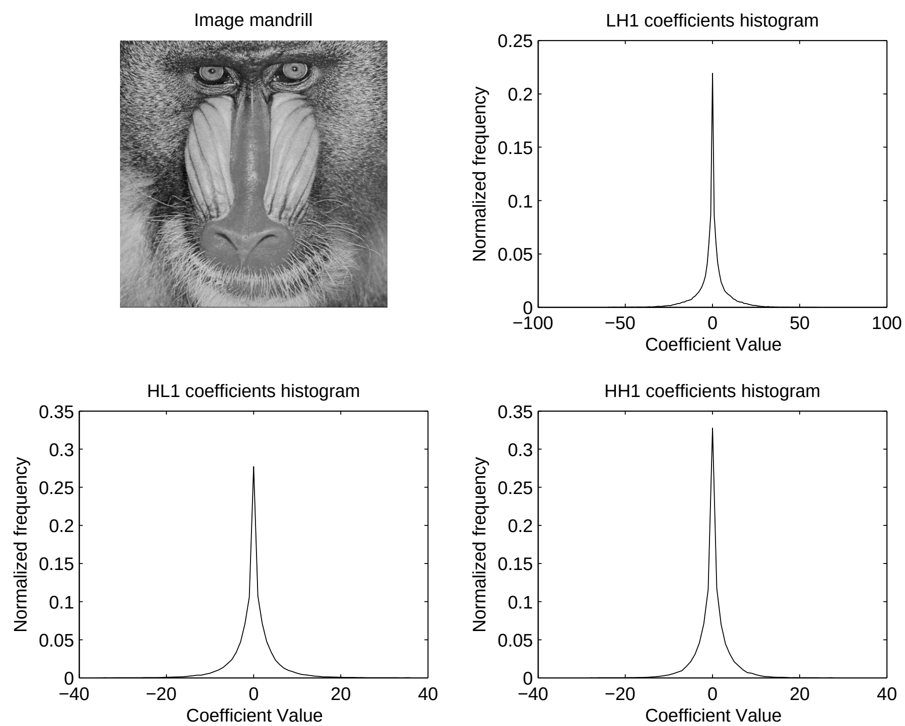


(b) Boat Image

Figure 1.3. Histogram of the Wavelet Coefficients of Natural Images - I



(a) Barbara Image



(b) Mandrill Image

Figure 1.4. Histogram of the Wavelet Coefficients of Natural Images - II

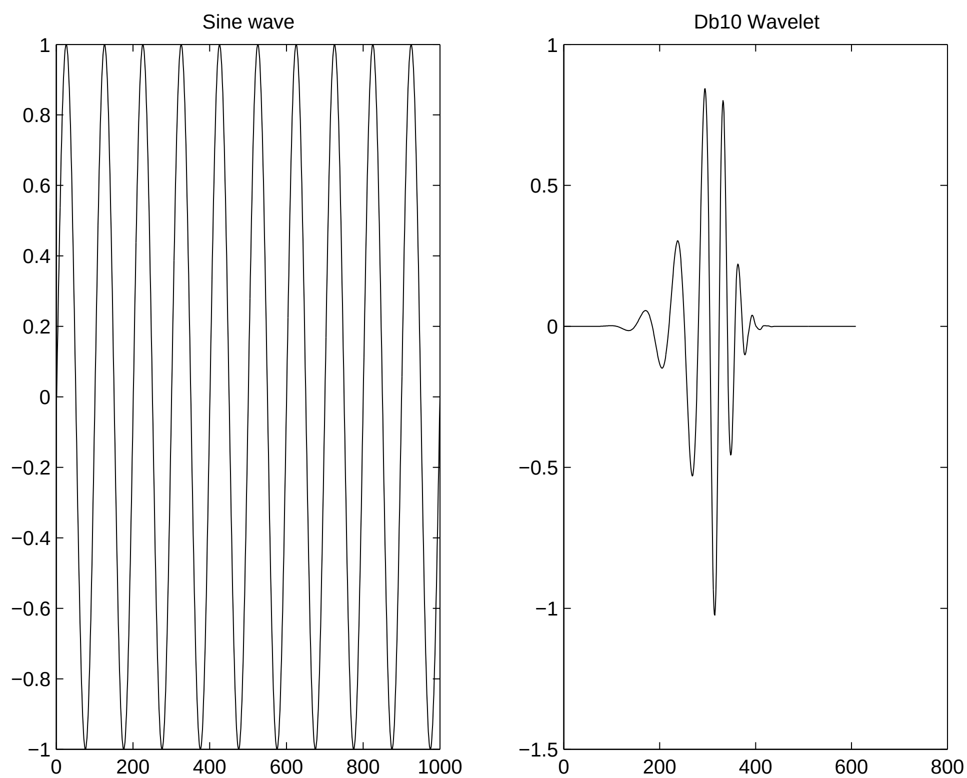


Figure 1.5. Sine Wave versus the Daubechies Db10 Wavelet

term related to the wavelet coefficients captures the details in the signal from scale j0 onwards.

The set of expansion coefficients ($c_{j0,k}$ and $d_{j,k}$) is called the discrete wavelet transform (DWT) of f(t). The above expansion is termed as the inverse transform.

Multi resolution analysis (MRA) and Quadrature mirror filters (QMF) are also important for evaluating the wavelet decomposition. In multi resolution formulation, a single event is decomposed into fine details [9]. A quadrature mirror filter consists of two filters. One gives the average (low pass filter), while the other gives details (high pass filter). These filters are related to each other in such a way as to be able to perfectly reconstruct a signal from the decomposed components [4]. Three levels of multi resolution analysis and synthesis are shown in Figure 1.6. QMF filters achieve perfect reconstruction of the original signal. Decimation operations are not shown in Figure 1.6. Decimation operations when removed, result in more data samples in multi resolution domain. This redundancy helps in denoising.

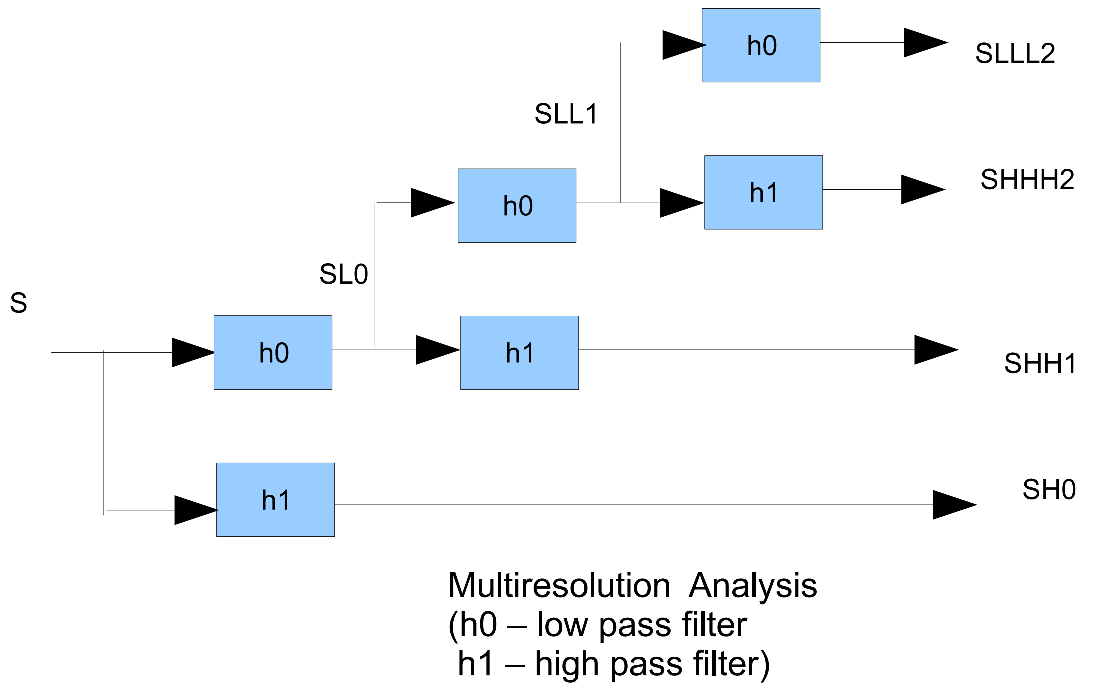
The two dimensional (2D) wavelet transform is an extension of the one dimensional (1D) wavelet transform. To obtain a 2D transform, the 1D transform is first applied across all the rows and then across all the columns at each decomposition level. Four sets of coefficients are generated at each decomposition level: LL as the average, LH as the details across the horizontal direction, HL as the details across the vertical direction and HH as the details across the diagonal direction.

There are other flavors of the wavelet transform such as translation invariant, complex wavelet transform etc., which give better denoising results. The translation invariant wavelet transform (TIWT) performs multi resolution analysis by filtering the shifted coefficients as well as the original ones at each decomposition level. TIWT is shift invariant (also known as time invariant). This approach produces additional wavelet coefficients (possessing different properties) from the same source. This redundancy improves the denoising performance.

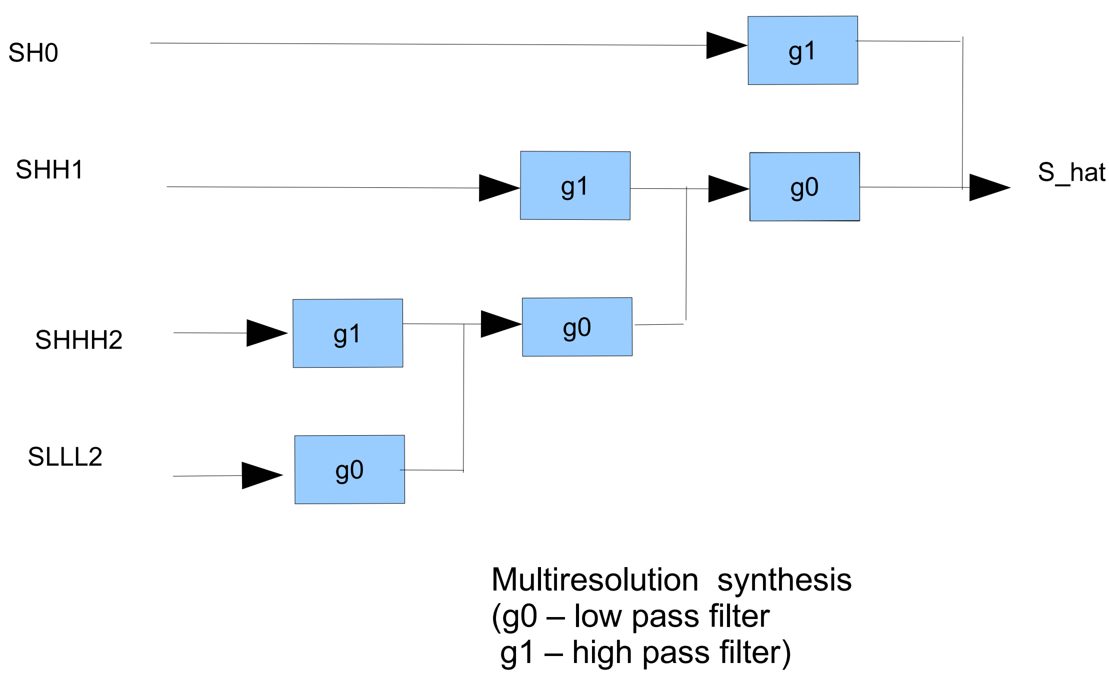
Complex wavelet transforms (CWTs) are a comparatively recent addition to the field of wavelets. A complex number includes some properties that can not be represented by a real number. These properties provide better shift-invariant feature and directional selectivity. However, CWTs with perfect reconstruction and good filter properties are difficult to develop. Dual tree complex wavelets (DT CWTs) were proposed by Kingsbury [10]. DT CWTs have some good properties such as reduced shift sensitivity, good directionality, perfect reconstruction using linear phase filters, explicit phase information, fixed redundancy and effective computation in O(N).

Multi wavelets are wavelets generated by more than one scaling function, while scalar wavelets use only one scaling function. Multi wavelets also improve denoising performance as compared to the scalar wavelet [11].

Wavelet transforms which generate more wavelet coefficients than the size of the input data are termed redundant or over complete. This added redundancy improves the denoising performance.



(a) Analysis



(b) Synthesis

Figure 1.6. Multiresolution Analysis (MRA)

Chapter 2

Survey of Literature

There are many different kinds of image denoising algorithms. They can be broadly classified into two classes:

- Spatial domain filtering
- Transform domain filtering

As evident from the names, spatial domain filtering refers to filtering in the spatial domain, while transform domain filtering refers to filtering in the transform domain. Image denoising algorithms which use wavelet transforms fall into transform domain filtering.

Spatial domain filtering can be further divided on the basis of the type of filter used:

- Linear filters
- Non-Linear filters

An example of a linear filter is the Wiener filter in the spatial domain. An example of a non-linear filter is the median filter. Median filtering is quite useful in getting rid of Salt and Pepper type noise. Spatial filters tend to cause blurring in the denoised image. Transform domain filters tend to cause Gibbs oscillations in the denoised image.

Transform domain filtering can be further divided into three broad classes based on the type of transform used:

- Fourier transform filters
- Wavelet transform filters

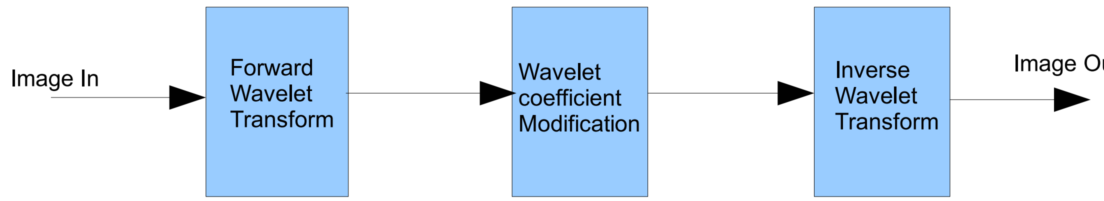


Figure 2.1. Denoising using Wavelet Transform Filtering

- Miscellaneous transform filters such as curvelets, ridgelets etc.

This work is focused on the wavelet transform filtering method. This method is chosen because of all the benefits mentioned in Section 1.4. All wavelet transform denoising algorithms involve the following three steps in general (as shown in Figure 2.1):

- Forward Wavelet Transform: Wavelet coefficients are obtained by applying the wavelet transform.
- Estimation: Clean coefficients are estimated from the noisy ones.
- Inverse Wavelet Transform: A clean image is obtained by applying the inverse wavelet transform.

There are many ways to perform the estimation step. Broadly, they can be classified as:

- Thresholding methods
- Shrinkage methods
- Other approaches

2.1 Thresholding Methods

These methods use a threshold and determine the clean wavelet coefficients based on this threshold. There are two main ways of thresholding the wavelet coefficients, namely the hard thresholding method and the soft thresholding method.

2.1.1 Hard Thresholding Method

If the absolute value of a coefficient is less than a threshold, then it is assumed to be 0, otherwise it is unchanged. Mathematically it is

$\hat{X} = sign(Y)(Y. * (abs(Y) > \lambda))$,

where Y represents the noisy coefficients, λ is the threshold, $\hat{X}$ represents the estimated coefficients.

2.1.2 Soft Thresholding Method

Hard thresholding is discontinuous. This causes ringing / Gibbs effect in the denoised image. To overcome this, Donoho [5] introduced the soft thresholding method.

If the absolute value of a coefficient is less than a threshold λ, then is assumed to be 0, otherwise its value is shrunk by λ. Mathematically it is

$\hat{X} = sign(Y). * ((abs(Y) > \lambda). * (abs(Y) - \lambda))$

This removes the discontinuity, but degrades all the other coefficients which tends to blur the image.

A summary of various thresholding methods used for denoising is given below.

2.1.3 VisuShrink

This is also called as the universal threshold method. A threshold is given by

$T = \sigma\sqrt{2log(M)}$ [5]

where σ^2 is the noise variance and M is the number of samples.

This asymptotically yields a mean square error (MSE) estimate as M tends to infinity. As M increases, we get bigger and bigger threshold, which tends to oversmoothen the image.

2.1.4 SUREShrink

This SUREShrink threshold was developed by Donoho and Johnstone [3]. For each sub-band, the threshold is determined by minimizing Stein's Unbiased Risk Estimate (SURE) for those coefficients. SURE is a method for estimating the loss $\| (\hat{\mu} - \mu)^2 \|$ in an unbiased fashion, where $\hat{\mu}$ is the estimated mean and μ is the real mean.

The threshold is calculated as follows:

$= \arg\min[\sigma^2 - \frac{2.\sigma^2}{n} \#\{k : abs(y_{j,k}) < \lambda\} + \frac{1}{n} \sum(\min(abs(y_{j,k}), \lambda)^2)]$

where n is the number of samples, σ^2 is the nosie variance, $y_{j,k}$ are the noisy samples, λ is the threshold and $\#\{k : abs(y_{j,k}) < \lambda\}$ indicates the number of samples whose value is less than λ. $\arg\min[f(\lambda)]$ indicates the selection of a value for lambda which minimizes the function f [12].

Donoho and Johnstone [3] pointed out that SUREShrink is automatically smoothness adaptive. This implies that the reconstruction is smooth wherever the function is smooth and it jumps wherever there is a jump or discontinuity in the function. This method can generate very sparse wavelet coefficients resulting in an inadequate threshold. So, it is suggested that a hybrid approach be used as an alternative.

2.1.5 BayesShrink

This method is based on the Bayesian mathematical framework. The wavelet coefficients of a natural image are modeled by a Generalized Gaussian Distribution (GGD). This is used to calculate the threshold using a Bayesian framework. S. Grace Chang et al. [13] suggest an approximation and simple formula for the threshold:.

$T = (\sigma_n)^2/\sigma_s$ if σ_s is non-zero. Otherwise it is set to some predetermined maximum value.

$\sigma_s = \sqrt{\max((\sigma_y)^2 - (\sigma_n)^2, 0)}$

$\sigma_y = \frac{1}{N}(\sum(W_n^2))$

The noise variance σ_n is estimated from the HH band as Median$(\mid W_n \mid)/0.6745$, where W_n represents the wavelet coefficients after subtracting the mean.

2.2 Shrinkage Methods

These methods shrink the wavelet coefficients as follows $\hat{x} = \gamma. * Y$ where $0 <= \gamma <= 1$ is the shrinkage factor.

The following methods belong to this category:

2.2.1 Linear MMSE Estimator

Michak et al. [14] proposed the linear Minimum Mean Square Estimation (MMSE) method using a locally estimated variance. Under the assumption of AWGN, an optimal predictor for the clean wavelet coefficient at location k is given by

$\hat{x_k} = y_k * (\sigma_{x,k}^2)/(\sigma_{x,k}^2 + \sigma^2)$

where $\sigma_{x,k}$ is the signal variance estimated at location k and σ is the noise variance, y_k represents the noisy coefficients and $\hat{x_k}$ represents the estimated coefficients.

Two methods were presented for the estimation of the local variance $\sigma_{x,k}$. The first one uses an approximate maximum likelihood (ML) estimator as follows:

$\sigma_{x,k}^2 = \arg\max \prod P(y_j \mid \sigma_{x,k}^2)$

$= max(0, \frac{1}{M}(\sum(y_j^2 - \sigma^2)))$

The second approach uses the maximum a posteriori (MAP) estimator as follows:

$\sigma_{x,k}^2 = \arg\max(\prod(P(y_j \mid \sigma_{x,k}^2)), p(\sigma_{x,k}^2))$

$= max(0, \frac{M}{4\lambda}(-1 + \sqrt{(1 + \frac{8.\lambda}{M^2}).\sum(y_j^2)}) - \sigma^2)$

where $P(\sigma_{x,k}^2) = \lambda.\exp(-\lambda\sigma^2)$ is empirically chosen.

Michak et al. [14] showed that the MAP estimator produces better results compared to the ML estimator. However, in the MAP method, an additional parameter (λ) needs to be estimated. It is suggested that it can be set to the inverse of the standard deviation of the wavelet coefficients that were initially denoised using the ML estimator.

The first method is referred as Michak1 and the second method is referred as Michak2 in the remainder of this text.

2.2.2 Bivariate Shrinkage using Level Dependency

All the above algorithms use a marginal probabilistic model for the wavelet coefficients. However, the wavelet coefficients of natural images exhibit high dependency across scale. For example, there exists a high probability of a large child coefficient if the parent coefficient is large.

Sunder and Selesnick in [15] proposed a bivariate shrinkage function using the MAP estimator and the statistical dependency between a wavelet coefficient and its parent. If w2 is the parent coefficient of w1 (at the same position as w1 but at the next coarser scale), then,

$y_1 = w_1 + n_1$

$y_2 = w_2 + n_2$

y_1 and y_2 are the noisy observations of w_1 and w_2. n_1 and n_2 are the AWGN noise samples.

They can be written as

$Y = W + N$ where $Y = (y_1, y_2), W = (w_1, w_2), N = (n_1, n_2)$

The standard MAP estimator for W given Y is

$\hat{W} = \arg\max \rho_{w|y}(w \mid y)$

$\hat{W} = \arg\max(\rho_{y|w}(y \mid w).P_w(w))$ after applying the Bayesian probability formula.

$= \arg\max(\rho_n(y - w).\rho_w(w))$

Since noise is assumed i.i.d. Gaussian

$\rho_n(y-w) = (1/(2\pi\sigma_n^2)).exp(-(n_1^2+n_2^2)/(2.\sigma_n^2))$

The next step involves the determination of $\rho_w(w)$. Sunder and Selesnick proposed four empirical models, each with its own advantages and disadvantages.

MODEL 1:

$P_w(w) = (\frac{3}{2}\pi\sigma^2).exp(-(\sqrt{3}/\sigma).\sqrt{w1^2+w2^2})$

The estimated coefficients are $\hat{w1} = \frac{(\sqrt{y1^2+y2^2}-\frac{\sqrt{3}\sigma_n^2}{\sigma})+}{\sqrt{y1^2+y2^2}}.y1$

MODEL 2:

$\rho_w(W) = K.exp(-(\alpha\sqrt{w^2+w2^2}+b(\mid w1\mid + \mid w2\mid)))$

Here K is the normalization constant.

The estimated coefficients are

$\hat{w1} = \frac{(R-\sigma_n^2.a)+}{R}.soft(y1,\sigma_n^2.b)$

$R = \sqrt{soft(y1,\sigma_n^2.b)^2 + soft(y2,\sigma_n^2.b)^2}$

$soft(g,t) = sign(g).(\mid g\mid -t)_+$

MODEL 3: In practice, the variance of the wavelet coefficients of natural images are quite different from scale to scale. So, the first model is generalized to include marginal variances.

$\rho(w) = \frac{3}{3\pi\sigma_1\sigma_2}.exp(-\sqrt{3}.\sqrt{(\frac{w1}{\sigma_1})^2+(\frac{w2}{\sigma_2})^2})$

The estimated coefficients are

$\hat{w1}.(1+\frac{\sqrt{3}\sigma_n^2}{\sigma_1^2 r}) = y1$

$\hat{w2}.(1+\frac{\sqrt{3}\sigma_n^2}{\sigma_2^2 r}) = y2$

where

$r = \sqrt{(\frac{\hat{w1}}{\sigma_1})^2+(\frac{\hat{w2}}{\sigma_2})^2}$

These two equations don't have a simple closed form solution, but numerical solutions do exist.

MODEL 4: In practice, the variance of the wavelet coefficients of natural images are quite different from scale to scale. So, the second model is generalized to include marginal variances.

$$\rho(w) = K.exp(-(\sqrt{c1.w_1^2 + c2.w_2^2} + c3. \mid w1 \mid + c4. \mid w4 \mid))$$

where K is the normalization constant.

The estimated coefficients are

$$\hat{w}_1.(1 + \frac{c1.\sigma_n^2}{r}) = soft(y1, c3\sigma_n^2)$$

$$\hat{w}_2.(1 + \frac{c2.\sigma_n^2}{r}) = soft(y2, c4\sigma_n^2)$$

where,

$$r = \sqrt{c1.\hat{w_1}^2 + c2.\hat{w_2}^2}$$

These two equations don't have a simple closed form solution, but numerical solutions do exist.

2.3 Other Approaches

2.3.1 Gaussian Scale Mixtures

Portilla et al. [16] proposed a method for removing noise from digital images based on a statistical model of the coefficients of an over-complete multi-scale oriented basis. Neighborhoods of coefficients at adjacent positions and scales are modeled as the product of two independent random variables: a Gaussian vector and a hidden positive scaler multiplier. The latter modulates the local variance of the coefficients in the neighborhood, and is able to account for the empirically observed correlation between the coefficient amplitudes.

Mathematically, the denoising problem can be written as

$$Y = \sqrt{z}U + W$$

Where U is the zero mean Gaussian random variable, z is the positive scaler multiplier, W is the AWGN and Y refers to the observed coefficients in the neighborhood.

The algorithm can be summarized as follows:

- The image is decomposed into sub-bands (A specialized variant of the steerable pyramid decomposition is used. The representation consists of oriented bandpass bands at 8 orientations and 5 scales, 8 oriented high pass residual sub-bands, and 1 low pass residual sub-band for a total of 49 sub-bands.)

- The following steps (reproduced from [16] for subject completeness) are performed for each sub-band (except for low pass residual):

  - Compute neighborhood noise covariance, C_w, from the image-domain noise covariance.

  - Estimate the noisy neighborhood covariance, C_y.

  - Estimate C_u from C_w and C_y using
    $C_u = C_y - C_w$

  - Compute $\wedge$ and M
    $C_w = SS^T$ and let Q,$\wedge$ be the eigenvector/eigenvalue expansion of the matrix $S^{-1}C_uS^{-T}$.
    M = S.Q

  - For each neighborhood:

    * For each value z in the integration range, compute $E\{x_c \mid y, z\}$ and $p(y \mid z)$ as follows:
      $E\{x_c \mid y, z\} = \sum \frac{zm_{cn}\lambda_n v_n}{z\lambda_n + 1}$
      where m_{ij} represents an element (ith row, jth column) of the matrix M, λ_n are the diagonal element of $\wedge$, v_n the elements of $v = M^{-1}y$.
      $\rho(y \mid z) = \frac{exp(-\frac{1}{2}\sum\frac{v_n^2}{z\lambda_n+1})}{\sqrt{(2\pi)^N|C_w|\prod(z\lambda_n+1)}}$
    * Compute $\rho(z \mid y)$ with $\rho_z(z) = \frac{1}{z}$
      $\rho(z \mid y) = \frac{\rho(y|z)\rho_z(z)}{\int_1^\infty \rho(y|\alpha)\rho_z(\alpha)\,d\alpha}$

* Compute $Ex_c \mid y$ numerically by

  $E\{x_c \mid y\} = \int_1^\infty \rho(z \mid y) E\{x_c \mid y, z\}\, dz$

- The denoised image is reconstructed from the processed sub-bands and the low pass residual.

2.3.2 Non-Local Mean Algorithm

Natural images often have a particular repeated pattern. Baudes et al. [17] used the self-similarities of image structures for denoising. As per their algorithm, a reconstructed pixel is the weighted average of all the pixels in a search window. The search window can be as large as the whole image or even span multiple images in a video sequence. Weights are assigned to pixels on the basis of their similarity with the pixel being reconstructed. While assessing the similarity, the concerned pixel, as well as its neighborhood are taken into consideration. Mathematically, it can be expressed as:

$NL[u](x) = \frac{1}{C(x)} \int \exp - \frac{(G_a * |u(x+.) - u(y+.)|^2)(0)}{h^2} u(y)\, dy$

The integration is carried out over all the pixels in the search window.

$C(x) = \int \exp - \frac{(G_a * |u(x+.) - u(y+.)|^2)(0)}{h^2}\, dz$ is a normalizing constant. G_a is a Gaussian kernel, and h acts as a filtering parameter.

The pseudocode for this algorithm is as follows:

For each pixel x

- We take a window centered in x and size 2t+1 x 2t+1, A(x,t).
- We take a window centered in x and size 2f+1 x 2f+1, W(x,f).
- wmax=0;
- For each pixel y in A(x,t) and y different from x

- We compute the difference between W(x,f) and W(y,f), d(x,y).
- We compute the weight from the distance d(x,y), w(x,y) = exp(- d(x,y) / h);
- If w(x,y) is bigger than wmax then wmax = w(x,y);
- We compute the average, average + = w(x,y) * u(y);
- We carry the sum of the weights, totalweight + = w(x, y);

* We give to x the maximum of the other weights, average += wmax * u(x); totalweight + = wmax;
* We compute the restored value, rest(x) = average / totalweight;

The distance is calculated as follows:

```
function distance(x,y,f) {
 distancetotal = 0 ;
 distance = (u(x) - u(y))^2;
 for k= 1 until f {
    for each i=(i1,i2)
    pair of integer
    numbers such that
    max(|i1|,|i2|) = k {
    distance + =
      ( u(x+i) - u(y+i) )^2;
    }
    aux = distance / (2*k + 1 )^ 2;
    distancetotal + = aux;
 }
```

```
 distancetotal / = f;
}
```

This algorithm is computationally intensive. A faster implementation with improved computation performance was later presented by Wang et al. [18].

2.3.3 Image Denoising using Derotated Complex Wavelet Coefficients

Miller and Kingsbury [19] proposed a denoising method based on statistical modeling of the coefficients of a redundant, oriented, complex multi-scale transform, called the dual tree complex wavelet transform (DT-CWT). They used two models, one for the structural features of the image and the other for the rest of the image. Derotated wavelet coefficients were used to model the structural features, whereas Gaussian Scale Mixture (GSM) models were used for texture and other parts of the image. Both of these models were combined under the Bayesian framework for estimation of the denoised coefficients.

Model 1: $x = \sqrt{z}u$ (to model areas of texture)

Model 2: $x = A.w = \sqrt{z}.A.q$ (to model structural features),

where z is the hidden or GSM multiplier and u is a neighborhood of Gaussian variables with zero mean and covariance C_u, q is a vector of Gaussian distributed random variables with covariance C_q and A is a unitary spatially varying inverse derotation matrix which converts a set of derotated coefficients q to the corresponding DT-CWT (Discrete Time Complex Wavelet Transform) coefficients using the phase of the interpolated parent coefficients.

Chapter 3

Wavelets in Action

The denoising of a one dimensional signal using a moving average filter, a Wiener filter and a simple wavelet thresholding is brought out in Section 3.1. The denoising of the standard "Lena" image using a moving average filter, a Wiener2 filter [20] and two wavelet methods is discussed in Section 3.2. The wavelet approach turns out to be a winner, both visually as well as quantitatively. The effect of different wavelet bases is studied. It is also noted that different wavelets produce slightly different results.

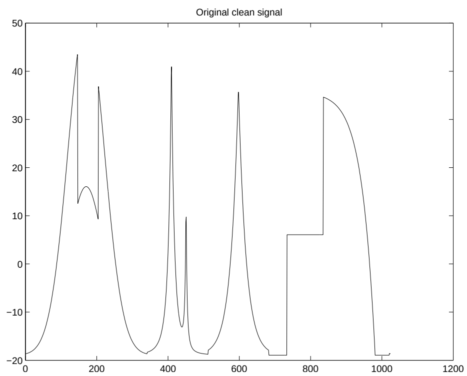
3.1 1D signal Denoising Example

Wavelets do a good job of considerably reducing the noise while preserving the edges, as shown in Figure 3.1. It works well in the smooth areas of the signal, as well as also preserves the edges or structures of the signal. In this section, the average filter, the Wiener filter and the wavelet method are compared. The optimal solution for each method is found by doing multiple iterations. The wavelet method performs very well, both visually as well as quantitatively. It must be noted that the simplest method to threshold wavelet coefficients is used. The denoising performance can be further improved by thresholding the wavelet coefficients using advanced methods.

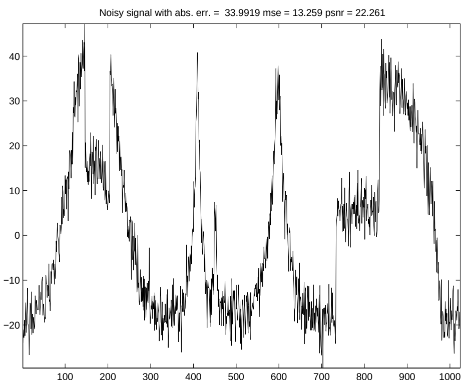
3.1.1 Effect of the Wavelet Basis

The denoising performance of wavelet transform methods is affected by the following:

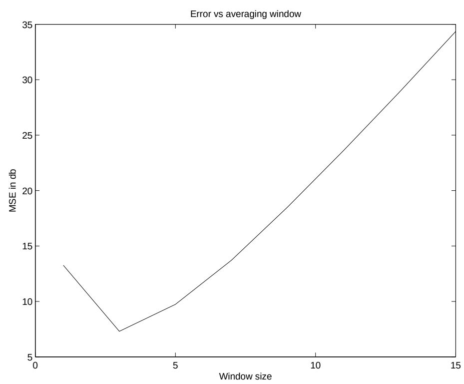
- Wavelet basis
- Number of decompositions



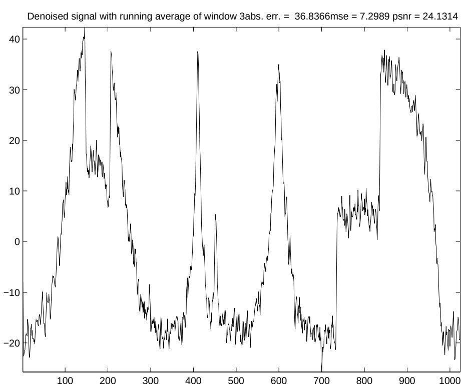
(a) Clean Piece Wise Regular Signal



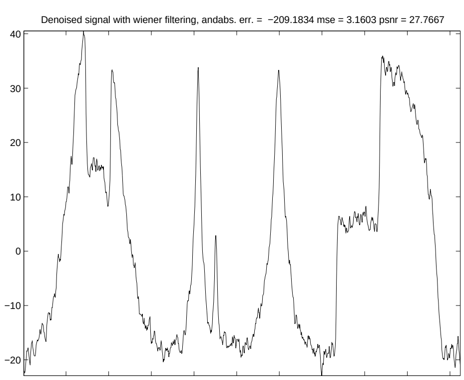
(b) Noisy Signal



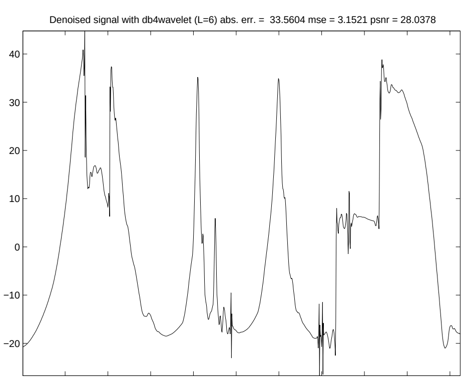
(c) MSE versus Window Size for Averaging Method



(d) Denoised Signal using the Averaging Filter



(e) MSE versus Threshold for the Wavelet Denoising



(f) Denoised Signal with the Wavelet Method

Figure 3.1. Denoising Example 1D Signal (Errors are in dB)

- Transform type (orthogonal, redundant, translation invariant, etc)
- Thresholding method (algorithm to modify or estimate the wavelet coefficients)

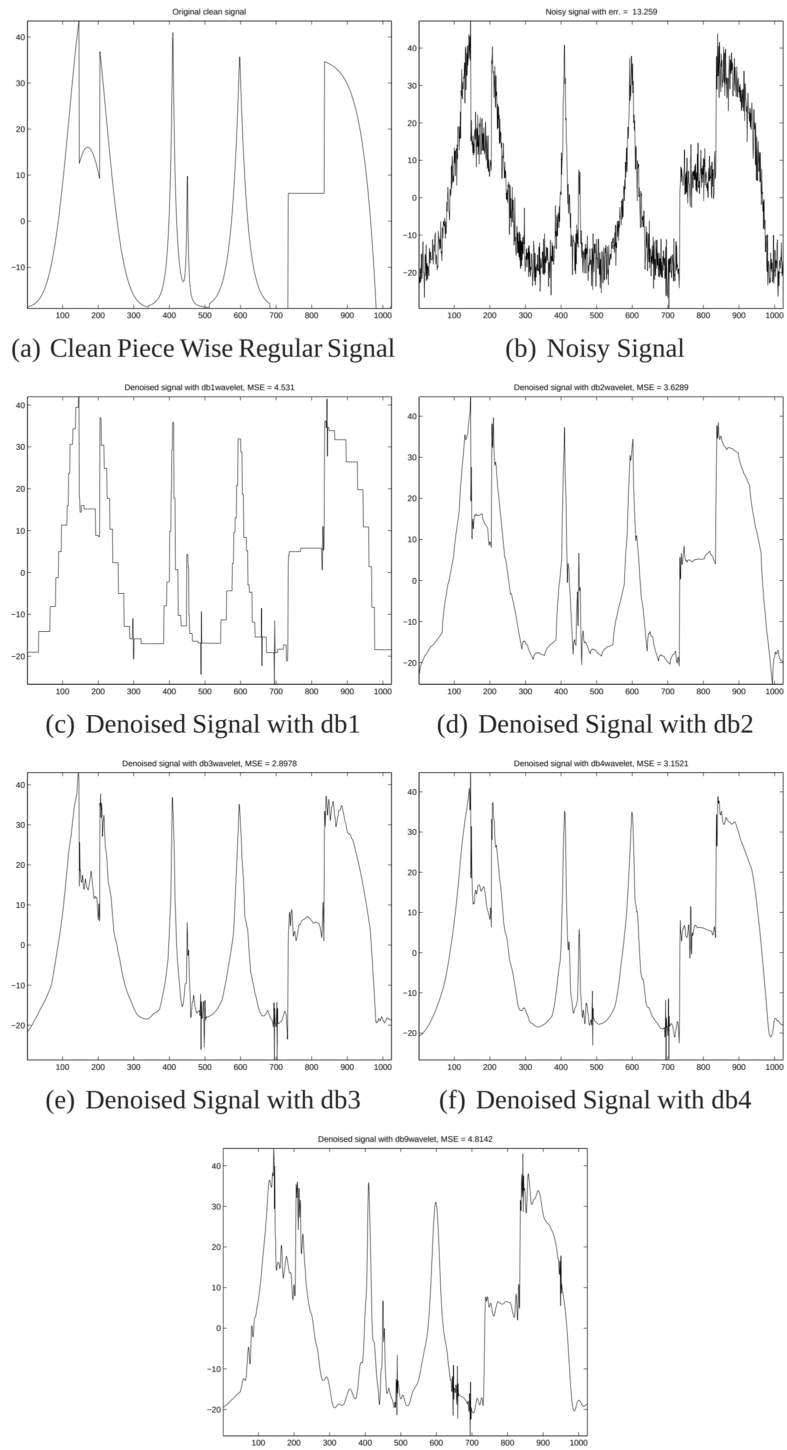
Some wavelet bases are better suited for certain signals when compared to others. Wavelet basis with more number of vanishing moments work better on the smooth parts of the signal. This is due to the fact that a polynomial of order N will not have any detailed coefficients (at all levels of decomposition) if it is decomposed with a wavelet having N or more vanishing moments. So, in this case, all the detailed coefficients will be from the noise, and can be killed. Figures 3.2 and 3.3 show the effect of wavelet bases on denoising performance.

3.2 Natural Image Denoising Example

Results from three different denoising methods - running average, Wiener2 filter [20] and wavelet methods - are compared in Figure 3.4. The performance of the wavelet approach is good, and comparable with that of the Wiener2 filter. The Daubechies wavelet with 10 vanishing moments is used with 2 levels of decomposition. Despite the adoption of the simplest global wavelet thresholding method, the moving average method is outperformed. Improved denoising results can be achieved by using better ways to threshold or estimate the wavelet coefficients. Example result (f) in Figure 3.4 shows that the Portilla method [16] can perform more than 1 dB better. The image also looks much cleaner and sharper. Thus, the wavelet approach does a better job of denoising while not blurring the image.

3.2.1 Effect of the Wavelet Basis

The wavelet basis also plays a role in denoising performance as shown in Figure 3.5, similar to the 1D case. The effect of the wavelet basis on denoising performance in case



(a) Clean Piece Wise Regular Signal

(b) Noisy Signal

(c) Denoised Signal with db1

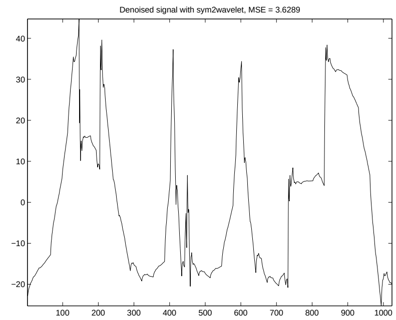
(d) Denoised Signal with db2

(e) Denoised Signal with db3

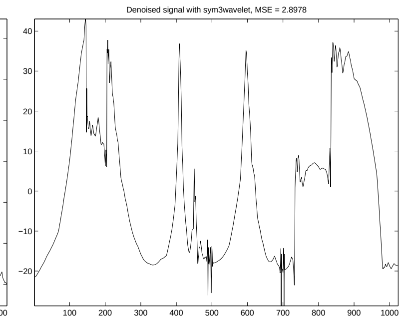
(f) Denoised Signal with db4

(g) Denoised Signal with db9

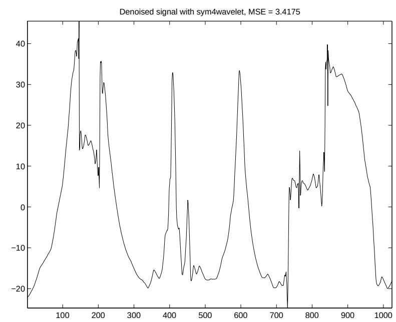
Figure 3.2. Effect of Different Wavelet Bases on 1D Signal Denoising I



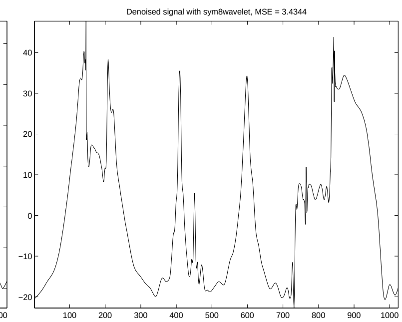
(a) Denoised Signal with sym2



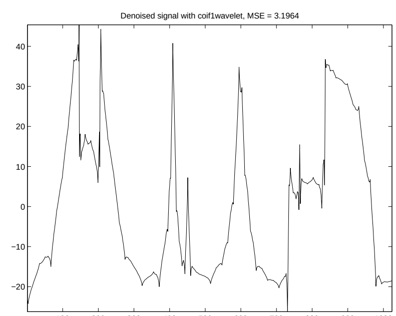
(b) Denoised Signal with sym3



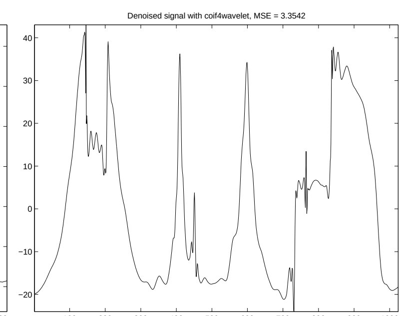
(c) Denoised Signal with sym4



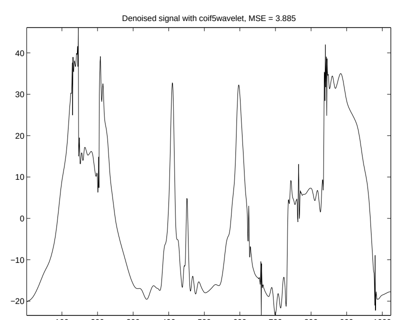
(d) Denoised Signal with sym8



(e) Denoises Signal with coif1



(f) Denoised Signal with Coif4



(g) Denoised Signal with Coif5

Figure 3.3. Effect of Different Wavelet Bases on 1D Signal Denoising II

(a) Original Image

(b) Noisy Image

(c) Denoised Image by Averaging Filter

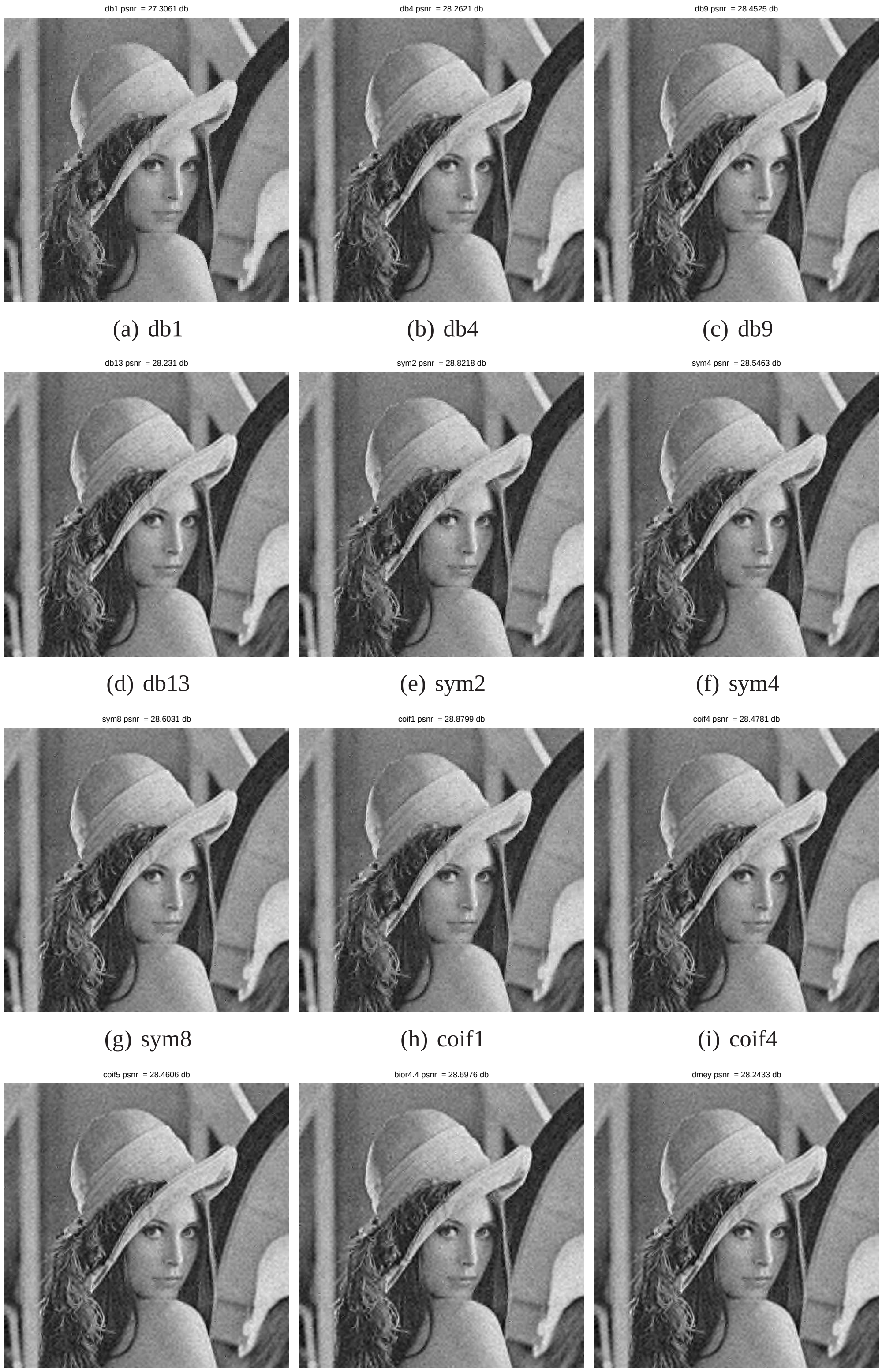
(d) Denoised Image by Wiener2 Filter

(e) Denoised Image by Wavelet Thresholding

(f) Denoised Image by BLS-GSM (Wavelet Method by Portilla et al.)

Figure 3.4. Denoising Example 2-D Image

of natural images is small.



(a) db1 (b) db4 (c) db9

(d) db13 (e) sym2 (f) sym4

(g) sym8 (h) coif1 (i) coif4

(j) coif5 (k) bior4.4 (l) dmey

Figure 3.5. Effect of Different Wavelet Bases on Natural Image Denoising

Chapter 4

Tetrolet Transform Based Denoising

Jens Krommweh [21] proposed a new method for image compression using an adaptive Haar like transform. He called it the Tetrolet transform. It is a simple concept, but quite effective in compression. In the 2D Haar transform, images are divided into 2x2 blocks and the Haar wavelet transform is applied to generate one average and three detailed coefficients. These coefficients capture the detailed information along the horizontal, vertical and diagonal direction. In the Tetrolet transform approach, images are sub-divided into 4x4 blocks. Each 4x4 block is partitioned using tetrominoes. Following this, the Haar transform is applied to generate 4 average coefficients and 12 detailed coefficients. Tetrominoes are the shapes formed by joining four squares such that they connect with each other at least on one edge. See Appendix A for more details about tetrominoes.

The Haar transform is a subset of the Tetrolet transform, with a partition of four 2x2 squares. The Tetrolet transform coefficients are the coefficients generated from the partition that generates the minimum sum of absolute values of all detailed coefficients. In order to recover the image from a Tetrolet transform, it is necessary to store information about the selected partition in each 4x4 block. This additional information offsets some of the advantage achieved by having effective coefficients. However, better compression can be achieved overall when compared with existing compression algorithms which use Haar wavelets.

A new denoising algorithm based on the above concept is proposed. The features of this algorithm are as follows:

- Simplicity: The algorithm is very simple. It does not require complex computations. All computations can be done using adders and shift registers, which are very cost effective for hardware implementations.

- Less Storage: Each 4x4 block is independently denoised. There is no necessity to store the full image or a large piece of the image, as required by other algorithms such as the non-local mean [17]. This makes it well suited for high performance real time applications.
- Redundant Coefficients: It is similar to denoising based on the translation invariant wavelet transform. However, the proposed approach has a higher degree of redundancy. This redundancy helps in achieving better denoising.
- Better Edge Preservation: It is observed that edges are well preserved.
- Scalability: Another variation of the algorithm is possible where the coarsest denoised image is generated using the Haar transform. A finer denoised image is produced when other tetromino partitions are picked and the average of such denoised images is taken.

4.1 Haar Wavelet Transform

The Haar wavelet transform is one of the most simple wavelet transforms. The scaling and wavelet functions for the Haar wavelet transform are defined as follows: $\phi(t)$ = 1 for $0 < t < 1$; 0 otherwise $\psi(t)$ = 1 for $0 < t < 0.5$; -1 for $0.5 < t < 1$; 0 otherwise

Figure 4.1 shows the scaling and wavelet functions at different scales and translation indices. A function can be decomposed into the translated and scaled wavelet function $\psi_{j,k}(t)$. The scaling function $\phi_{j0,k}(t)$ captures the average of the function at scale j0.

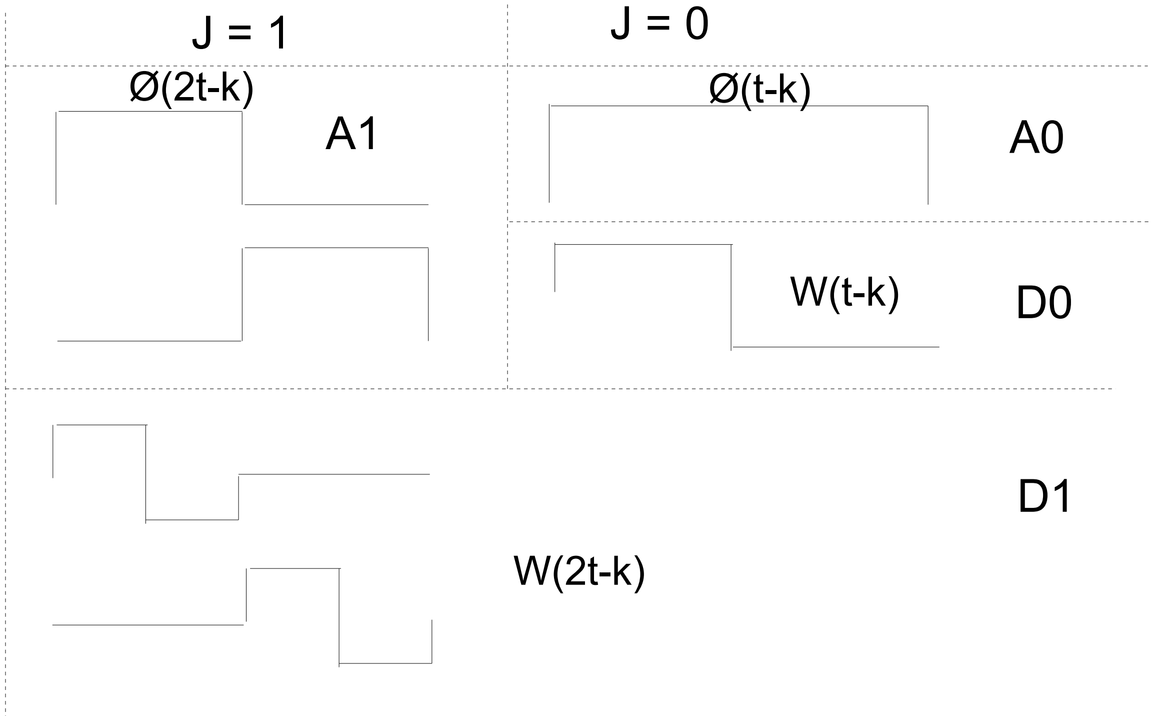


Figure 4.1. Illustration of the Haar Wavelet Transform

4.2 Example of the Tetrolet Transform

To understand the Tetrolet transform, consider the following example compared with the Haar transform to bring out the differences.

Figure 4.2 shows an example of a 4x4 block with a black dot in the center and a white background. Pixel values are from 0 to 255, with 0 being complete black and 255 being complete white. The Tetrolet coefficients are [480 40 0 0; 480 480 0 0; 0 0 0 0; 0 0 0 0] and the Haar coefficients are [370 370 110 -110; 370 370 110 -110; 110 110 -110 110; -110 -110 110 -110]. It can be seen that energy is highly concentrated in the case of the Tetrolet coefficients, while it is spread over all coefficients in the case of the Haar. This energy compactness property is helpful in denoising and compression.

In order to continue further testing, some noise is added. The transform, coefficient

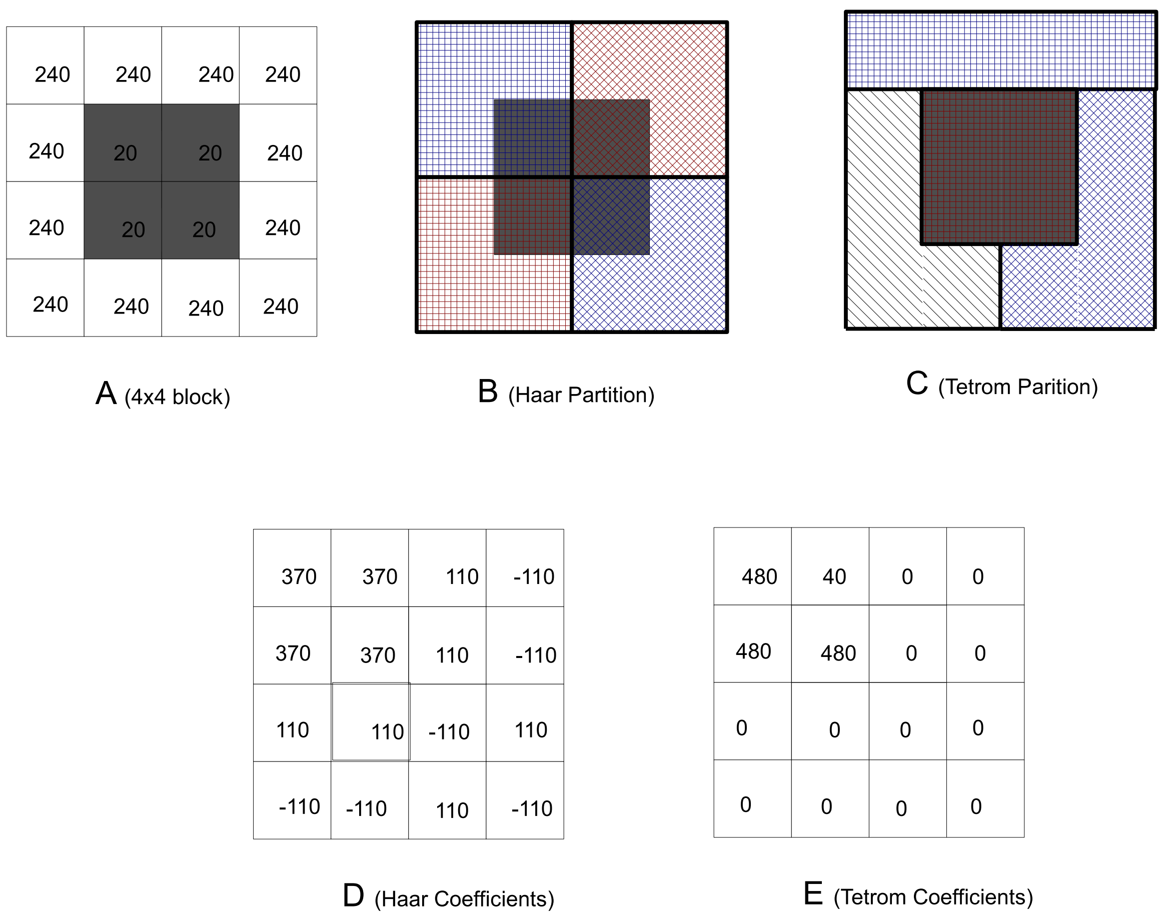


Figure 4.2. Illustration of the Tetrolet Transform Concept (1)

233	222	244	231
215	37	22	272
241	37	17	237
244	239	250	241

F (Noisy samples)

233	222	244	231
215	37	22	272
241	37	17	237
244	239	250	241

G (Haar Denoised samples)
psnr – 26.2 db

227	227	246	246
227	28	28	246
227	28	28	246
243	243	243	243

H (Tetrom Denoised Samples)
psnr – 29.4 db

Figure 4.3. Illustration of the Tetrolet Transform Concept (2)

thresholding and inverse transform are performed again. For this example, a threshold of 30 is used. The noisy samples after adding white noise with a variance of 15 are [233 222 244 231; 215 37 22 272; 241 37 17 237; 244 239 250 241]. Samples recovered by the Haar method are the same as the noisy samples. Since energy is equally distributed among all coefficients, no denoising results from the thresholding of the coefficients. Samples recovered by Tetrolet method are [227 227 246 246; 227 28 28 246; 227 28 28 246; 243 243 243 243]. Peak signal to noise ratio (PSNR), calculated as

$$PSNR(x, y) = 10 * \log_{10} max(max(x), max(y))^2/(\mid x - y \mid)^2$$

for the Haar method is 26.2 dB, while the PSNR from the Tetrolet method is 29.4 dB. More importantly, the features of the block are preserved.

It is found that the direct thresholding of the Tetrolet coefficients does not produce good results for denoising of natural images. An innovative solution which produces good results is proposed. Figures 4.4, 4.5 and 4.6 show the “Lena” image denoised using the Tetrolet transform as compared to the Haar transform. Different thresholding methods are used, as indicated. The improvement obtained is insignificant.

(a) Universal Hard Thresholding (Haar)

(b) Universal Hard Thresholding (Tetrom)

(c) Universal Soft Thresholding (Haar)

(d) Universal Soft Thresholding (Tetrom)

Figure 4.4. Haar versus the Tetrolet Transform Direct (1)

(a) Sure Thresholding (Haar)

(b) Sure Thresholding (Tetrom)

(c) Bayes Thresholding (Haar)

(d) Bayes Thresholding (Tetrom)

Figure 4.5. Haar versus the Tetrolet Transform Direct (2)

(a) Michak1 method (Haar)

(b) Michak1 method (Tetrom)

(c) Michak2 method (Haar)

(d) Michak2 method (Tetrom)

Figure 4.6. Haar versus the Tetrolet Transform Direct (3)

4.3 Histogram Comparison

The effectiveness of the Tetrolet transform in compression is illustrated by the histogram of the coefficients of natural images. Figures 4.7 and 4.8 clearly show that the Tetrolet coefficients produce larger number of zeros. In Figures 4.7 and 4.8, the X axis represents the magnitude of the coefficients, while the Y axis shows the normalized value of the number of coefficients. There are two curves in each histogram. The curve with the higher peak at X=0 corresponds to the Tetrolet transform. This indicates that the Tetrolet transform can be good for image compression.

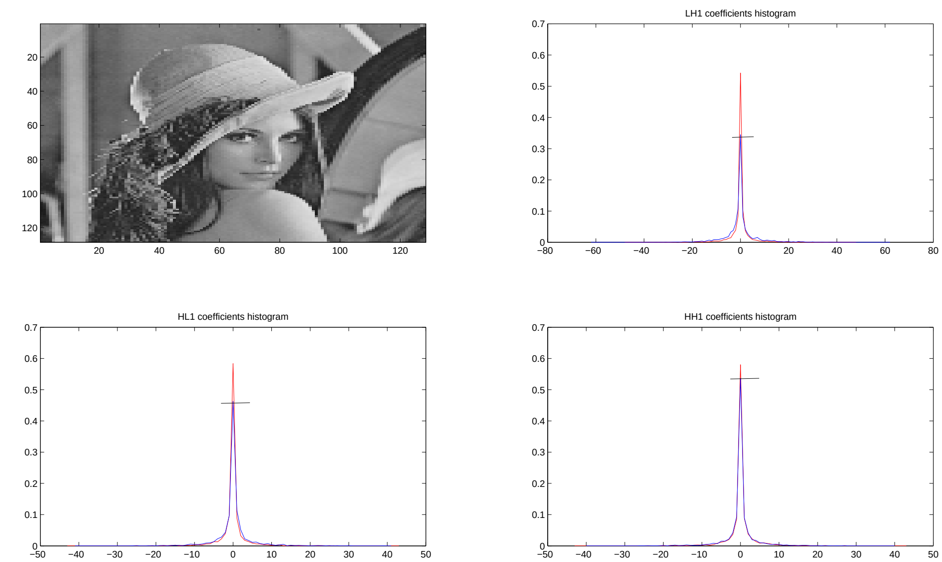
4.4 Tetrolet Transform Based Denoising Algorithm

Direct thresholding of the Tetrolet coefficients does not produce good results. The Tetrolet coefficients are thresholded using different methods in the images in Figures 4.4, 4.5, and 4.6. None of them seem to produce good results. There are 117 different ways to cover a 4x4 block using tetrominoe shapes. This produces a large number of coefficients and the redundancy is exploited in the newly proposed denoising algorithm described below.

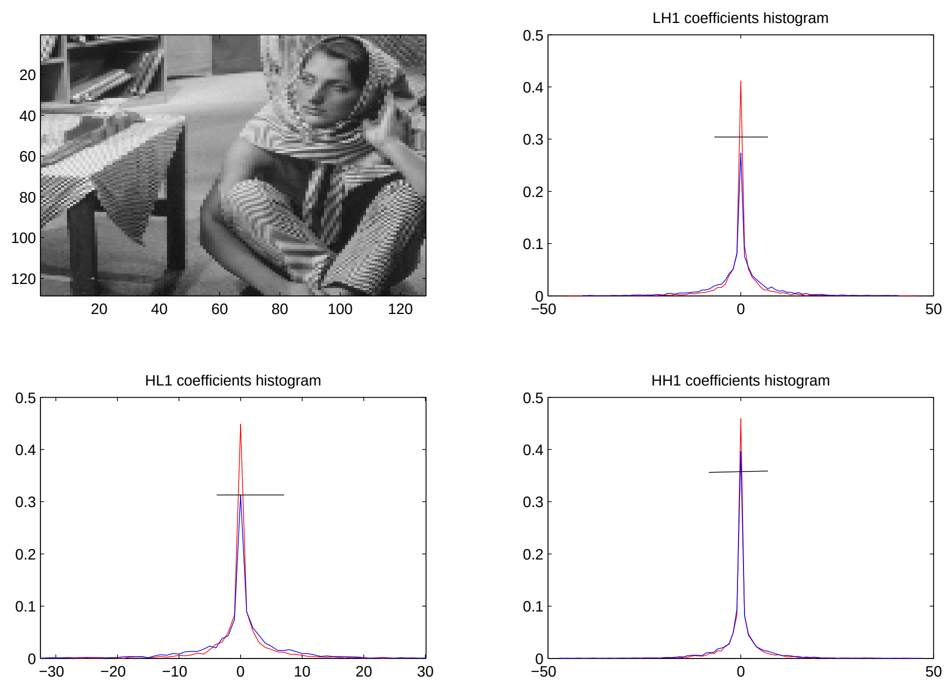
The image is extended if its height and width are not multiples of 4. After denoising, the image is cropped to get the original size. The extended image is divided into 4x4 blocks, and the following steps are performed for each of the blocks:

1. A tetrom configuration which can completely cover the block is picked. There are 117 possible configurations as described in Appendix A. The Haar partition is initially chosen, but it is not necessary to always start with it.

2. The samples of the low pass filter are arranged to minimize their Hamming distance from the corresponding Haar partition. This step is required to remove arbitrary

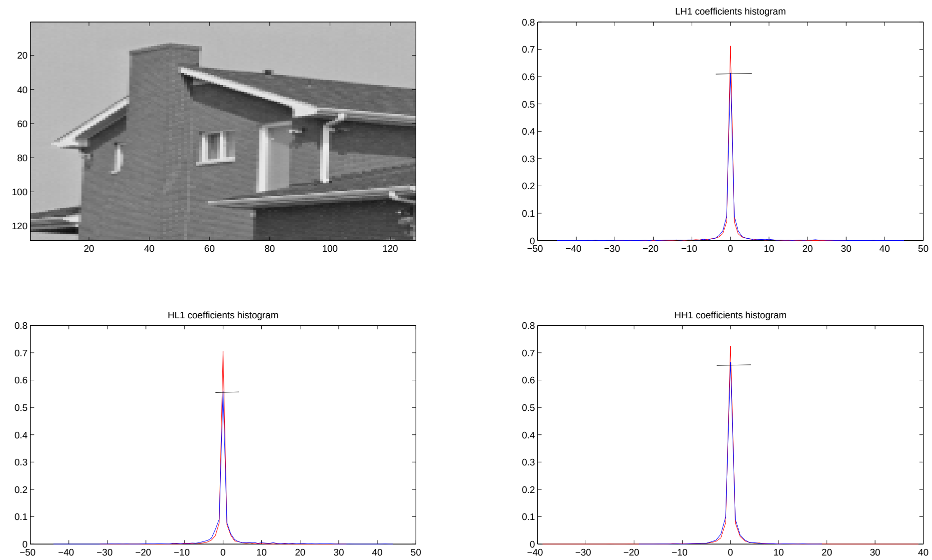


(a) Lena

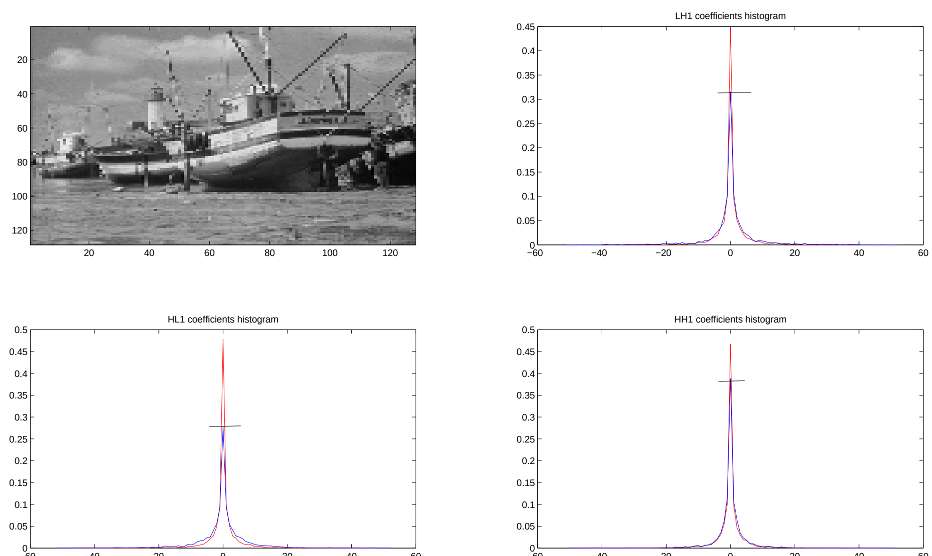


(b) Barbara

Figure 4.7. Histogram of the Tetrolet Coefficients of Natural Images (1)



(a) House



(b) Boat

Figure 4.8. Histogram of the Tetrolet Coefficients of Natural Images (2)

arrangement of samples and prepare average coefficients for the next level of decomposition. Squares of Haar partitions are labeled as 0, 1, 2 and 3. The Hamming distances between the squares of the Haar partition and the 24 different arrangements of the squares of a given tetrominoe partition are computed. The particular arrangement of squares which gives the minimum Hamming distance is chosen, as described by Jens Krommweh [21].

3. The Haar transform of the arranged samples is calculated.

4. The Haar coefficients generated in the above step are thresholded. A scaled version of the universal threshold obtained by the formula $T = \sigma\sqrt{2log(M)} * 0.68$ [5] is used for thresholding. By experiments it is found that the scaled version produces good results. The scale factor is another parameter that can be tuned. Variations are possible here. Any type of thresholding (including soft and hard thresholding methods) can be used. The effect of threshold on denoising performance is discussed in the performance section 5.

5. An inverse Tetrolet transform from the thresholded coefficients is done to get a sample of the recovered pixels.

6. Steps 2 through 5 are iterated after picking another way to partition the 4x4 block. There are 117 possible ways to partition (see Appendix A).

7. The average of all the collected samples is taken.

8. Pixels produced by the above method are the denoised version of the noisy pixels.

The algorithm can be summarized by the following pseudo code

```
// Extend the image so that the width and length of the image
// are multiples of 4. Divide the image into 4x4 blocks.

for each 4x4 block of the image
I4x4_hat = 0;          %

for partition=1 to 117 % all possible ways to fill 4x4
                       % region from tetrominoe shapes

I4x4_coeff = Haar Transform with selected partition (I4x4);

% Hard thresholding method is shown here,
% Other variations are possible like soft thresholding etc.
I4x4_coeff_thresholded = I4x4_coeff.*(abs(I4x4_coeff > T));
% T is the threshold value

I4x4_hat += Inverse Transform (I4x4_coeff_thresholded);
end
I4x4_hat = I4x4_hat/117; % I4x4_hat is the recovered block

% optional wavelet filtering with higher smooth wavelet
% to smooth out the picture. In the proposed algorithm,
% one level of wavelet decomposition with db3 and Hard
% thresholding has been used to denoise final image with
% 1/8th of original threshold.
```

```
end
```

The final division operation can be implemented using shifts if the number of partitions is a power of two. It is shown in the performance section that the later iterations do not improve the image quality by much. Dropping them from consideration improves the speed with very little or no cost to the image quality.

Chapter 5

Performance

Four standard test images (Lena, Barbara, House, and Boat) are corrupted with white noise and then denoised using various methods, including the one proposed by us. The result is compared based on the performance criteria listed in Section 5.1. The random noise added to the image is varied in steps of 5 with the standard deviation ranging from 10 to 30. Smaller images of size 128x128 are used for faster run times in the calculation of the PSNR performance table and Figures 5.1, 5.2, and 5.3. The performance table is generated from an average of 10 random runs. Bigger images of size 512x512 are used for visual comparison. In all the experiments, the starting random seed is fixed at 1001 in order to ensure that results can be replicated. Fixing the seed does not affect the overall behavior or the result. The following methods are compared.

- Universal Hard Thresholding Method by Donoho (referred as VisuHard)
- Universal Soft Thresholding Method by Donoho [5] (referred as VisuSoft)
- SURE Shrink Method by Donoho and Johnstone [3] (referred as Sure)
- Bayes Shrink Method by Chang et al. [13] (referred as Bayes)
- Linear MMSE Estimator Method 1 by Michak et al. [14] (referred as Michak1)
- Linear MMSE Estimator Method 2 by Michak et al. [14] (referred as Michak2)
- Gaussian Scale Mixture Method by Portilla et al. [16] (referred as BLS-GSM)
- Redundant Haar Transform Method [22] (referred as Redundant Haar)
- Method proposed by us (referred to as Tetrom)

We have not included the Non Local mean algorithm by Buades et al. [17]. Though this is one of the latest algorithms and has good performance, it is very intensive in terms of computational complexity as well as memory requirement. This is due to its non-local nature. Further, the algorithm is not wavelet based. Because of these reasons, this algorithm is not in the same category as the others that are being compared above.

5.1 Performance Criteria

Different algorithms are compared based on the following criteria:

- Quantitative comparison - Different algorithms are compared based on the PSNR of the denoised image. The PSNR is calculated as
  $PSNR(x, y) = 10 * \log_{10} max(max(x), max(y))^2/(\mid x - y \mid)^2$,
  where x and y are the clean and estimated samples respectively. Higher PSNR indicates better denoising performance.

- Visual comparison and subjective analysis - Denoised images were subject to a poll where people were asked to pick the three least noisy images, and rank them as first, second and third choice.

- Residual analysis - The noise obtained after subtracting the denoised image from the noisy image is visually inspected for features from the original image. Ideally this should be white noise with no visible image features.

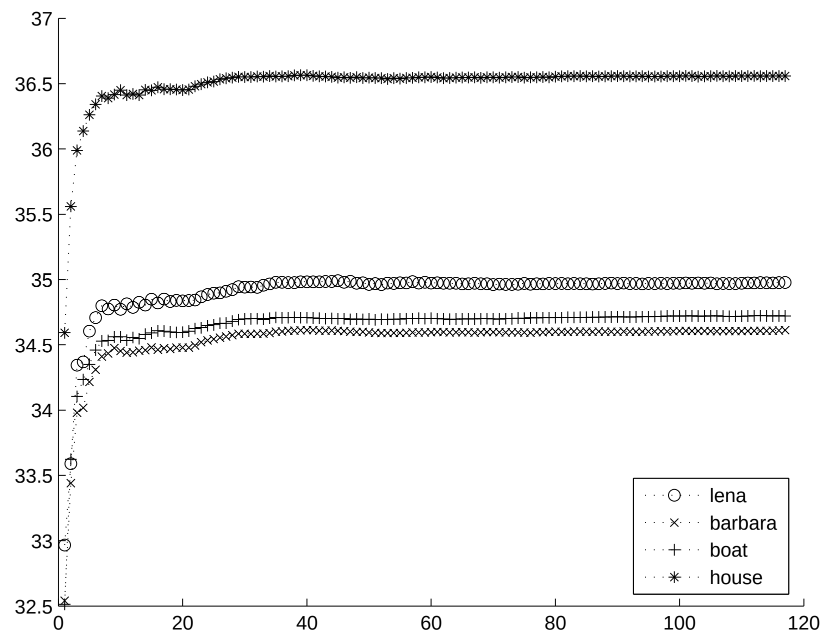
5.2 Comparison with Haar Wavelet Transform and Universal Thresholding

The PSNR values of the denoised image are plotted against the number of tetrominoe partitions being averaged in Graphs 5.1, 5.2, and 5.3. The PSNR values are plotted along the Y-axis and the number of partitions that are being averaged are plotted along the

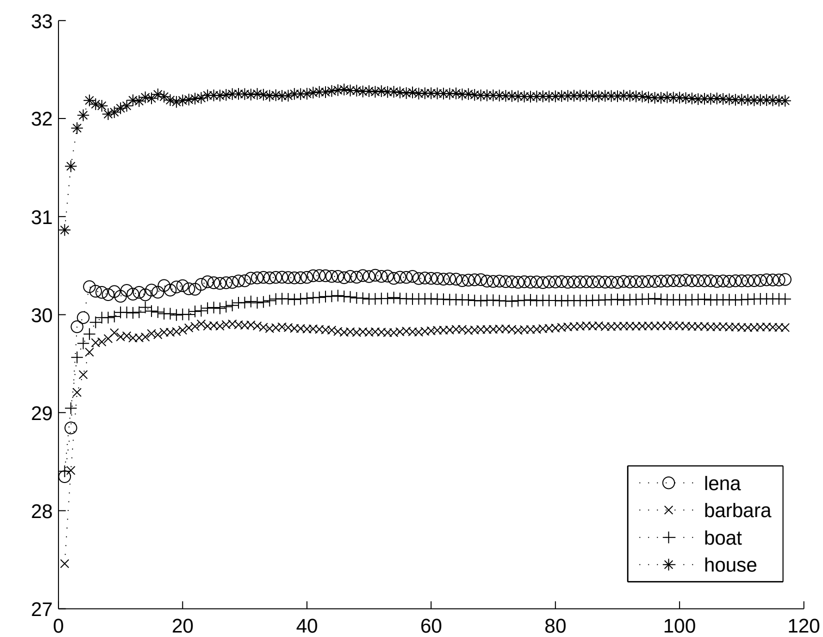
X-axis. X=1 corresponds to the Haar wavelet transform and universal thresholding method.

It can be seen that redundancy improves the denoising performance by a factor of thousand. Denoising performance improves as more and more tetrominoe partitions are averaged. Performance improves rapidly at the start and saturates around a mean after a while. There are two reasons for this:

- Duplication in the generated coefficients is the primary reason. Figure 5.4 shows the duplication in the coefficients generated by selecting different tetrominoe partitions.
- The nature of the problem also contributes to this observation, as explained below. In the tetrolet transform based denoising, a 4x4 block is tiled with tetrominoes followed by the application of the Haar wavelet transform. The Haar wavelet coefficients obtained are thresholded. Samples are obtained via an inverse wavelet transform. This way, many samples are obtained for a pixel value. The assumption is that these samples would be distributed around the true value and, by taking the average of all values, denoising would result. If samples randomly drawn from a normal distribution are averaged, then, the average would rapidly approach the mean. The convergence towards the mean would slow down, as can be seen in Figure 5.5. It shows the average values of samples which are normally distributed around a mean value of 65. The average is plotted on the Y-axis and the number of samples that are being averaged is plotted on the X-axis. It can be seen that the result quickly converges to about 65 by just adding a few samples. Later samples do not add much value.

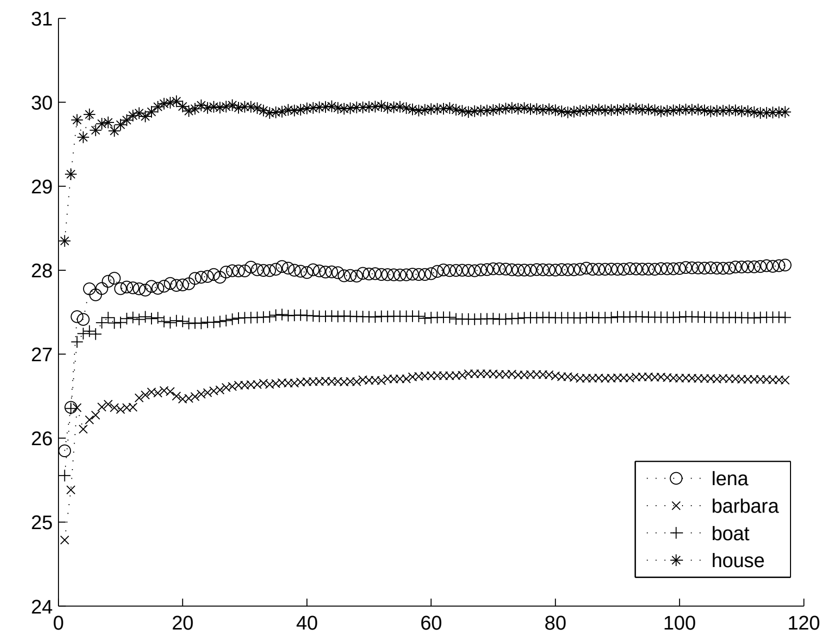


(a) Sigma = 5

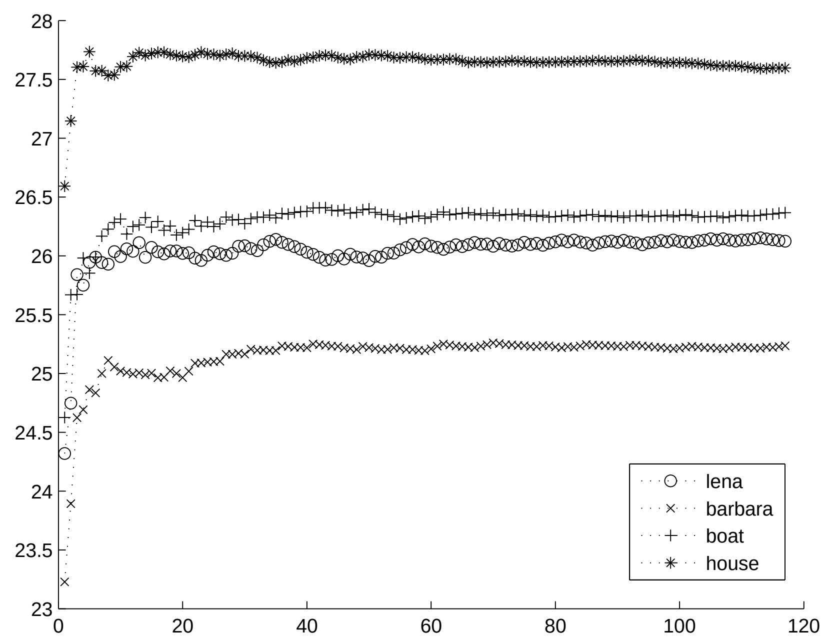


(b) Sigma = 10

Figure 5.1. PSNR versus Number of Tetrominoes Partitions being Averaged (1)

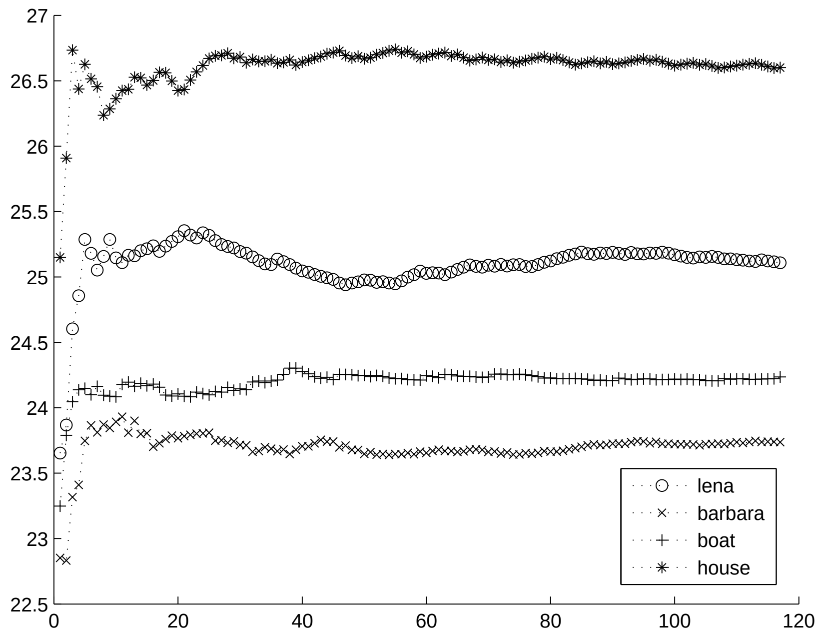


(a) Sigma = 15

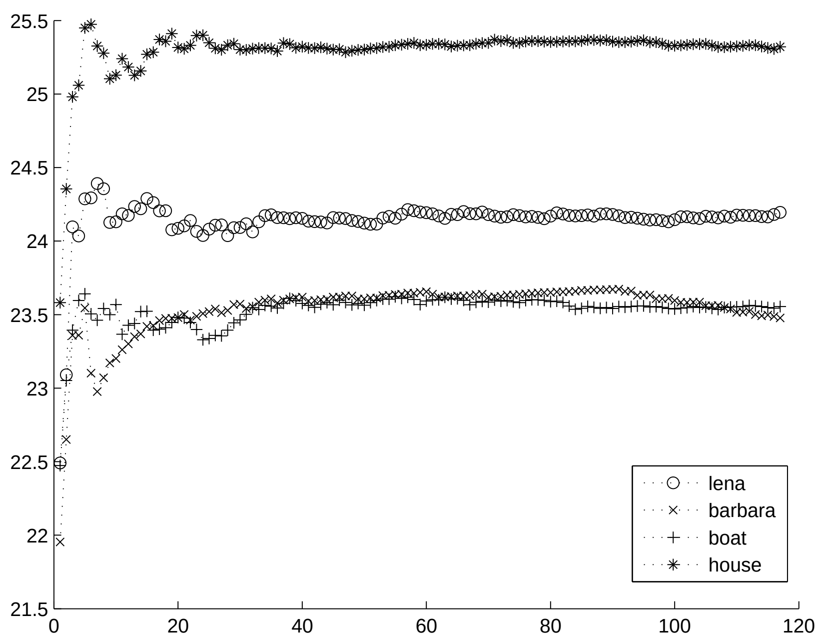


(b) Sigma = 20

Figure 5.2. PSNR versus Number of Tetrominoes Partitions being Averaged (2)



(a) Sigma = 25



(b) Sigma = 30

Figure 5.3. PSNR versus Number of Tetrominoes Partitions being Averaged (3)

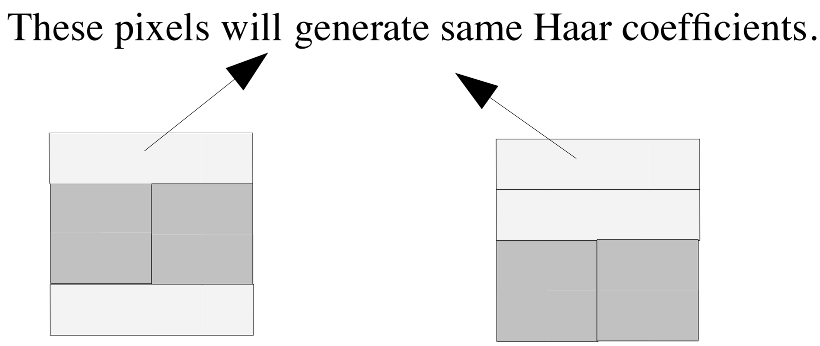


Figure 5.4. Duplicate Haar Coefficients in Two Different Tetrominoe Tilings

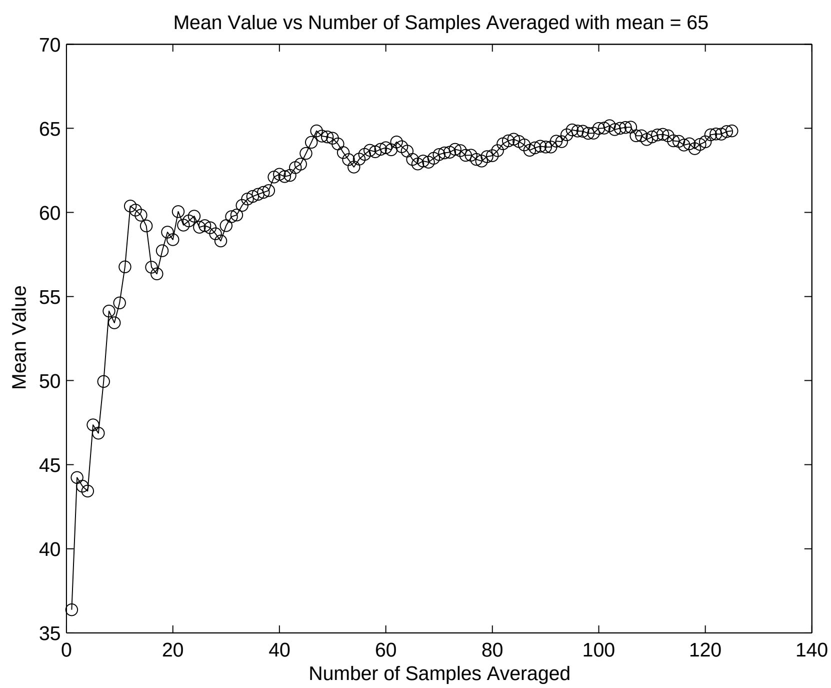


Figure 5.5. Mean Value versus Number of Samples being Averaged

5.3 Visual Comparison

Four well-known test images (Lena, Boat, House, and Barbara) of size 512x512 were corrupted with white noise having a variance of 30. The noisy images as well as the denoised ones (processed using various methods) are presented in this section for visual inspection.

A web based form [23] was created to do a subjective blind test in which the quality of a denoised image was assessed by votes from the audience. People were asked to choose the three least noisy images in their opinion and rank them as their first, second and third choice. The latest results of the poll can be found at the URL in [24]. Figure 5.6 is a snapshot of the results at the time of writing this report.

The method presented in this thesis came up as the second best after the method from Portilla et al. [16]. Due to the simplicity and non-local nature of the presented algorithm, it has advantages over Portilla's method in real-time hardware implementations.

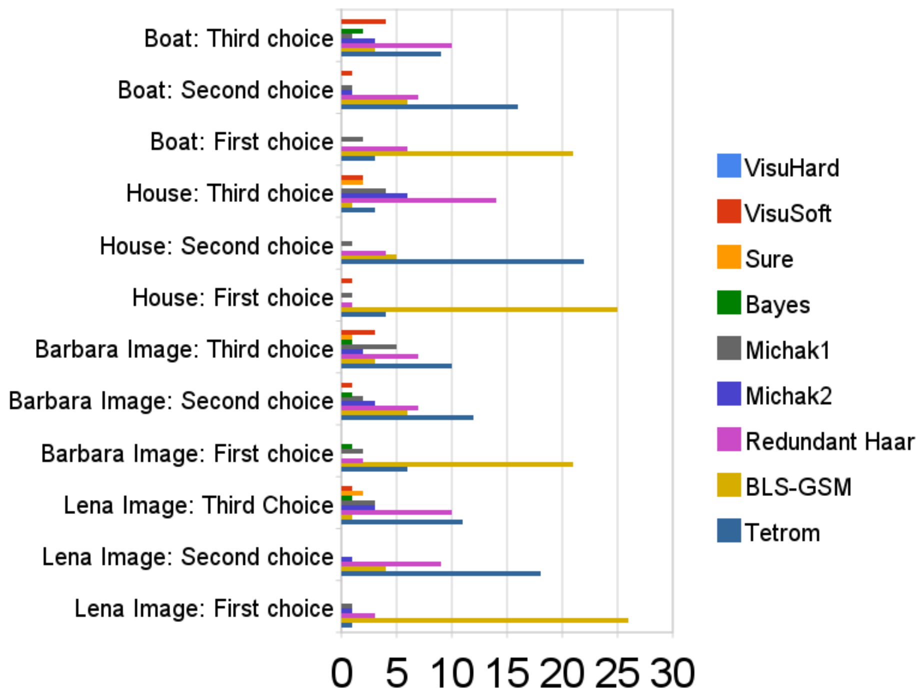


Figure 5.6. Subjective Assessment - People's Votes

5.4 Lena Image Example

Figure 5.7 shows:

(a) Clean Lena image of size 512x512

(b) Noisy Lena image, noise of variance = 30 is added to image (a)

(c) Lena image denoised by universal hard thresholding

(d) Lena image denoised by universal soft thresholding

Denoised Images (c) and (d) in Figure 5.7 are up to 4 dB better compared to the noisy one, but the visual appearance is still noisy. Further optimization is possible if we decompose the image further. Since the new method developed in this thesis uses only one level of decomposition, all compared methods have been kept to one level of decomposition for fairness.

Figure 5.8 shows:

(a) Lena image denoised by SURE thresholding by Donoho and Johnstone [3]

(b) Lena image denoised by Bayes Shrink method by Chang et al. [13]

(c) Lena image denoised by Linear MMSE estimator method 1 by Michak et al. [14]

(d) Lena image denoised by Linear MMSE estimator method 2 by Michak et al. [14]

Figure 5.9 shows:

(a) Lena image denoised by Gaussian scale mixture method of Portilla et al. [16]

(b) Lena image denoised by the method proposed in this thesis

It can be seen that the best image is produced by the Gaussian scale mixture method. The second best picture is produced by the method proposed in this thesis, which exceeds other methods by up to 2 dB. The denoised image also looks less noisy compared to other methods.

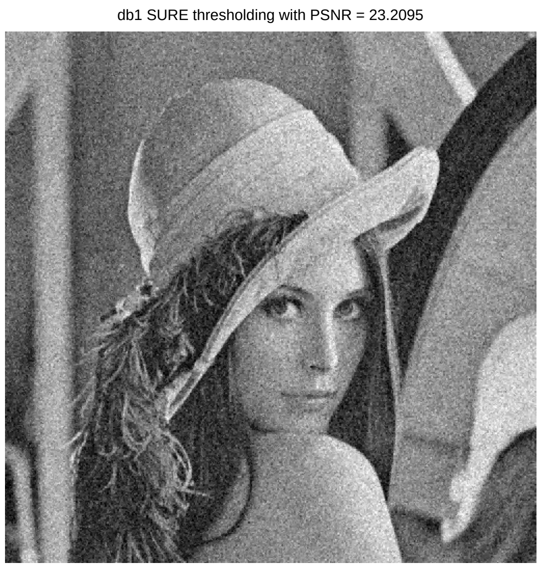
(a) Clean Image

(b) Noisy Image

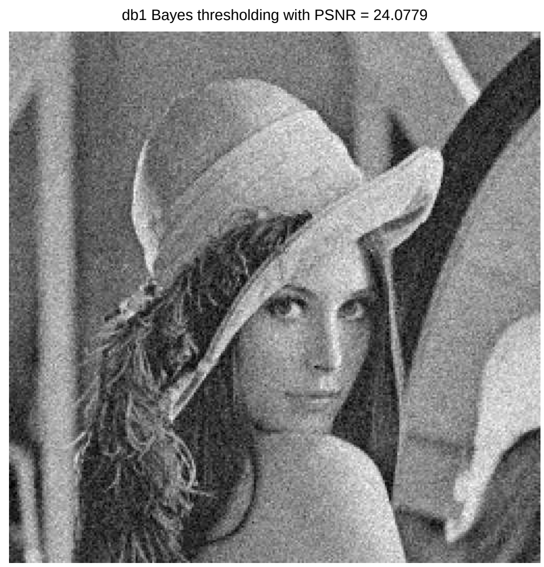
(c) VisuShrink Hard thresholding

(d) VisuShrink Soft thresholding

Figure 5.7. Lena Image Denoised I



(a) SURE thresholding



(b) Bayes thresholding

(c) MMSE shrinkage Michak Method 1

(d) MMSE shrinkage Michak Method 2

Figure 5.8. Lena Image Denoised II

(a) BLS-GSM

(b) Tetrom

Figure 5.9. Lena Image Denoised III

5.5 The Boat Image Example

Figure 5.10 shows:

(a) Clean image of the boat of size 512x512

(b) Noisy image of the boat, noise of variance = 30 is added to image (a)

(c) Image of the boat denoised by universal hard thresholding

(d) Image of the boat denoised by universal soft thresholding

Denoised Images (c) and (d) are up to 3 dB better compared to the noisy one, but the visual appearance is still noisy. Further optimization is possible if we decompose the image further. Since the new method developed in this thesis uses only one level of decomposition, all compared methods have been kept to one level of decomposition for fairness.

Figure 5.11 shows:

(a) The boat image denoised by SURE thresholding method of Donoho and Johnstone [3]

(b) The boat image denoised by Bayes Shrink method of Chang et al. [13]

(c) The boat image denoised by Linear MMSE estimator method 1 of Michak et al. [14]

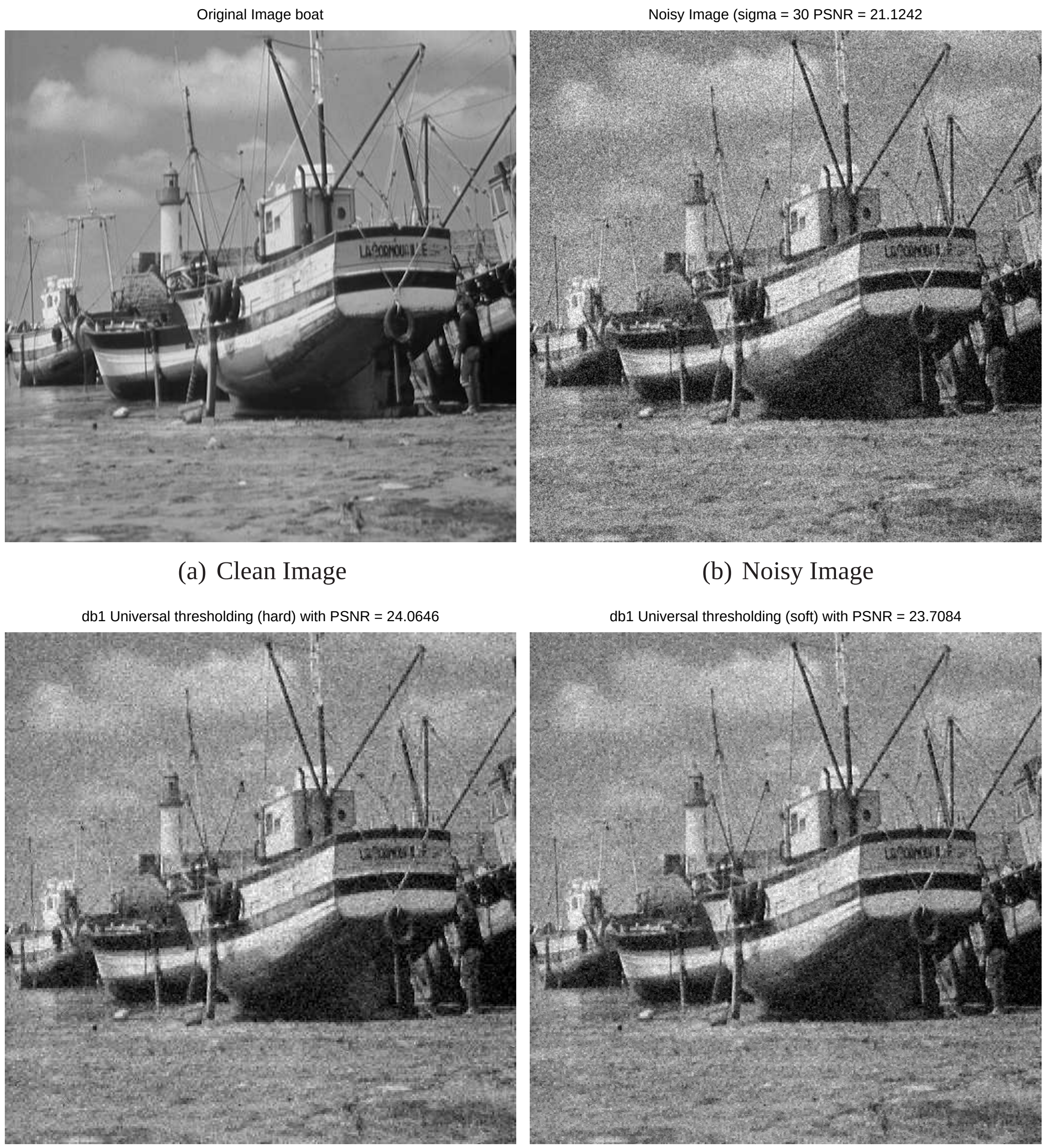
(d) The boat image denoised by Linear MMSE estimator method 2 of Michak et al. [14]

Figure 5.12 shows:

(a) Image of the boat denoised by Gaussian scale mixture method of Portilla et al. [16]

(b) Image of the boat denoised by the new method proposed in this thesis

It can be seen that the best image is produced by the Gaussian scale mixture method. The second best picture is produced by the method proposed in this thesis, which exceeds other methods by up to 2 dB. The denoised image also looks less noisy compared to other methods. Another advantage of the proposed method is the fact that there is no noticeable blurring of the fine details in the original image.



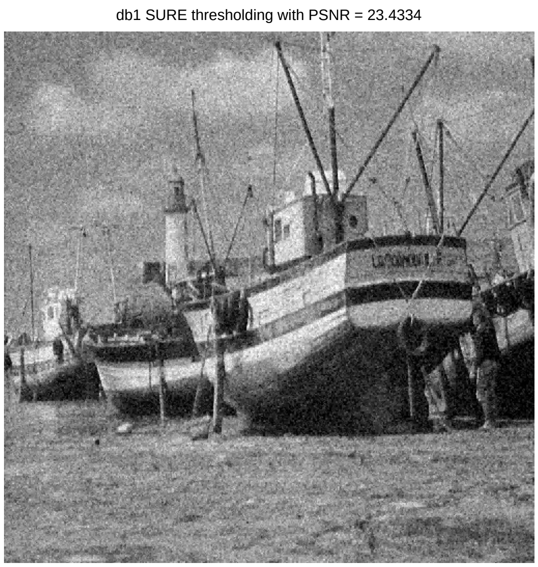
(a) Clean Image

(b) Noisy Image

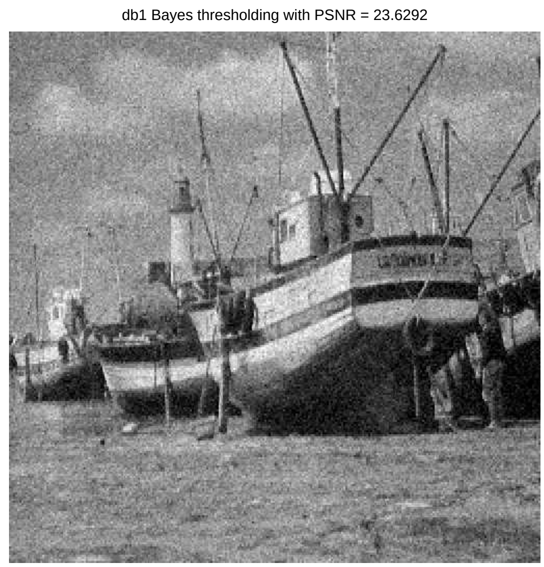
(c) VisuShrink Hard thresholding

(d) VisuShrink Soft thresholding

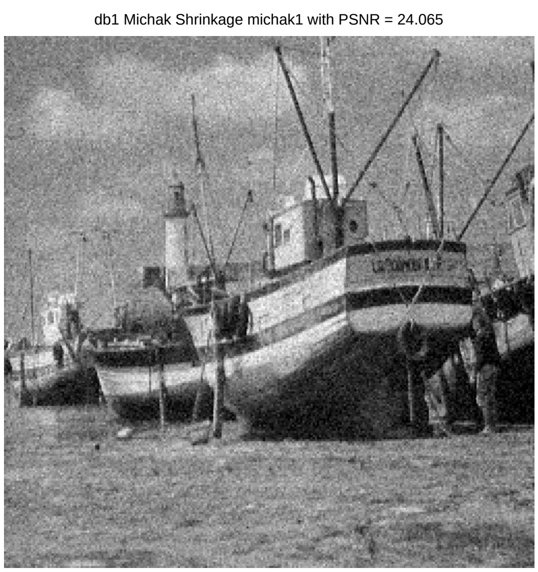
Figure 5.10. Boat Image Denoised I



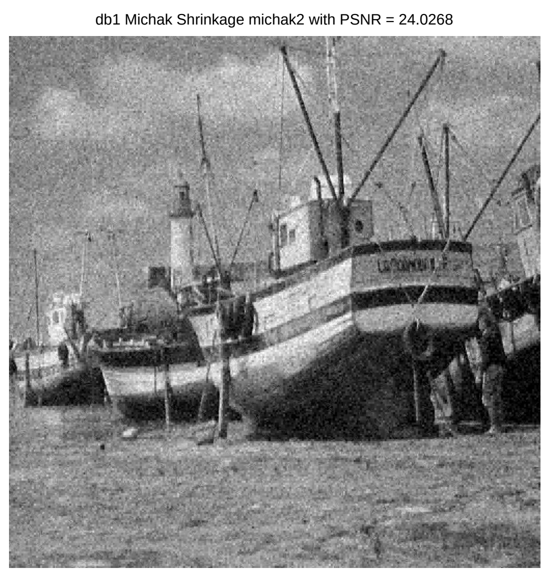
(a) SURE thresholding



(b) Bayes thresholding



(c) MMSE shrinkage Michak Method 1



(d) MMSE shrinkage Michak Method 2

Figure 5.11. Boat Image Denoised II

(a) BLS-GSM

(b) Tetrom

Figure 5.12. Boat Image Denoised III

5.6 The House Image Example

Figure 5.13 shows:

(a) Clean image of the house of size 512x512

(b) Noisy image of the house, noise of variance = 30 is added to image (a)

(c) The house image denoised by universal hard thresholding

(d) The house image denoised by universal soft thresholding

Denoised Images (c) and (d) are up to 4 dB better compared to the noisy one, but the visual appearance is still noisy. Further optimization is possible if we decompose the image further. Since the new method developed in this thesis uses only one level of decomposition, all compared methods have been kept to one level of decomposition for fairness.

Figure 5.14 shows:

(a) The house image denoised by SURE thresholding of Donoho and Johnstone [3]

(b) The house image denoised by Bayes Shrink method of Chang et al. [13]

(c) The house image denoised by Linear MMSE estimator method 1 of Michak et al. [14]

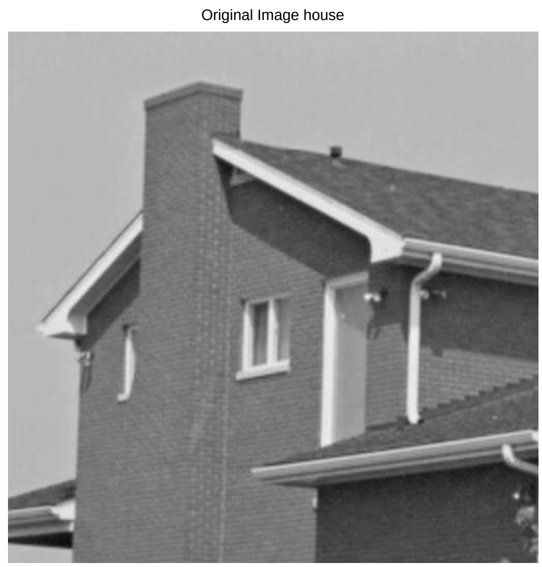
(d) The house image denoised by Linear MMSE estimator method 2 of Michak et al. [14]

Figure 5.15 shows:

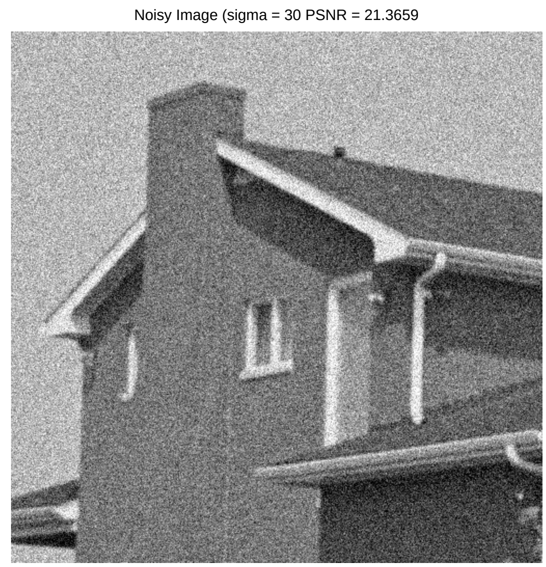
(a) Image of the house denoised by Gaussian scale mixture method of Portilla et al. [16]

(b) Image of the house denoised by new method developed in this thesis

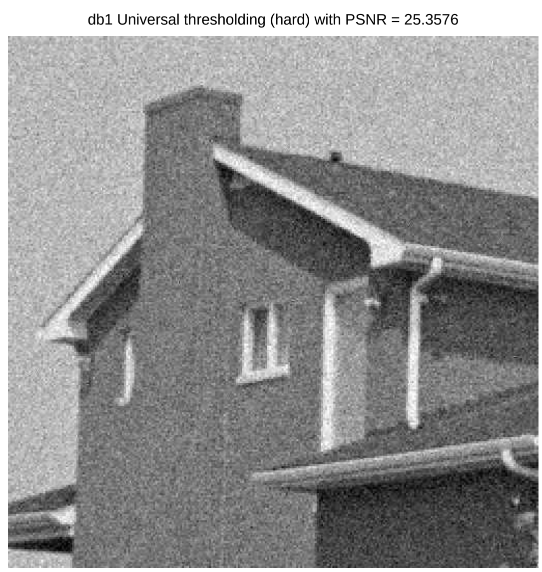
The results for the House image are similar to the ones obtained for the Lena and Boat images. The best image is obtained by the Gaussian scale mixture method, which shows a 9 dB improvement. The second best image is produced by the method proposed in this thesis, with a 6 dB improvement. The proposed method betters other methods by performing upto 3 dB better.



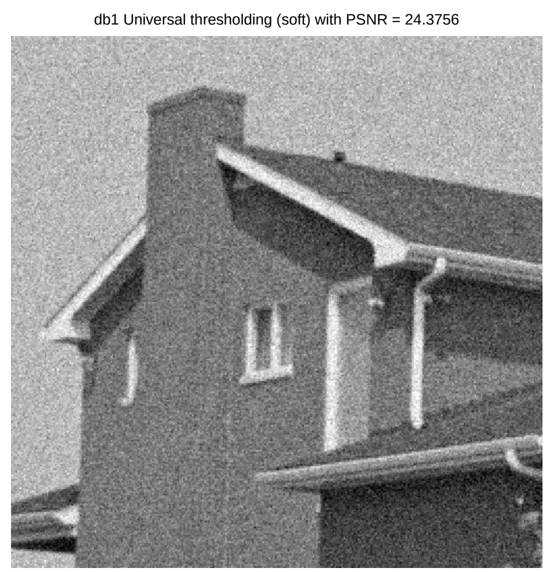
(a) Clean Image



(b) Noisy Image

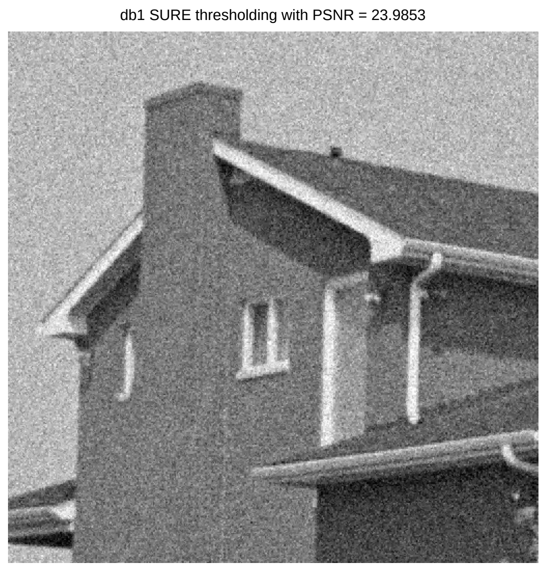


(c) VisuShrink Hard thresholding

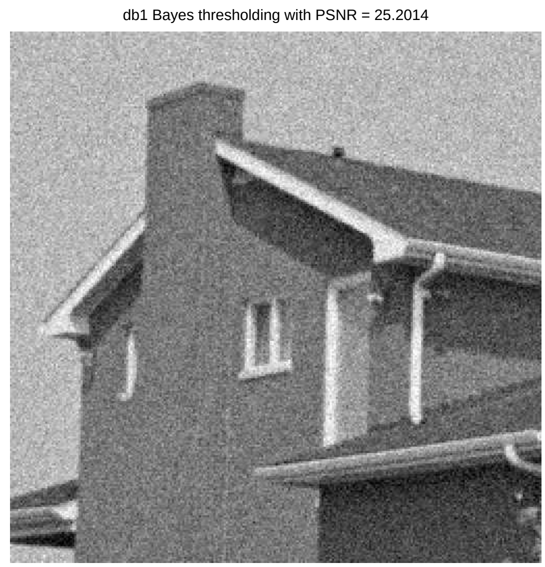


(d) VisuShrink Soft thresholding

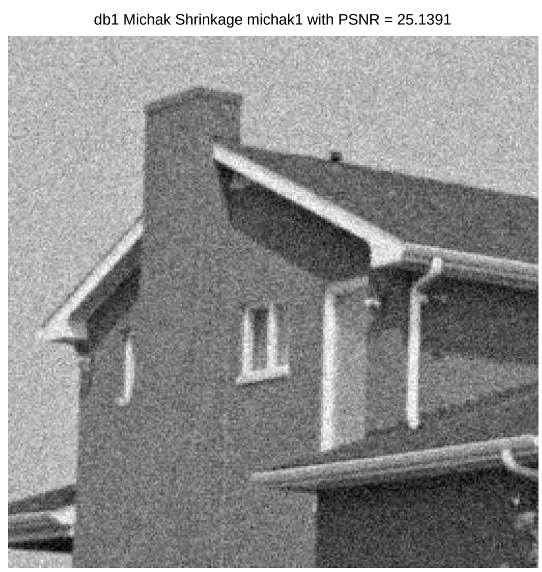
Figure 5.13. House Image Denoised I



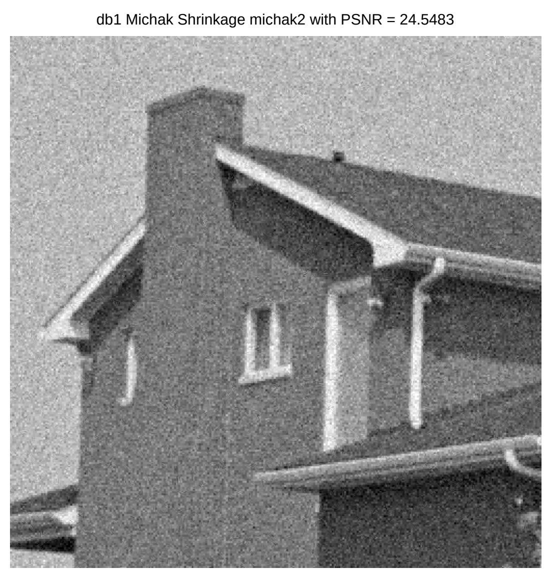
(a) SURE thresholding



(b) Bayes thresholding

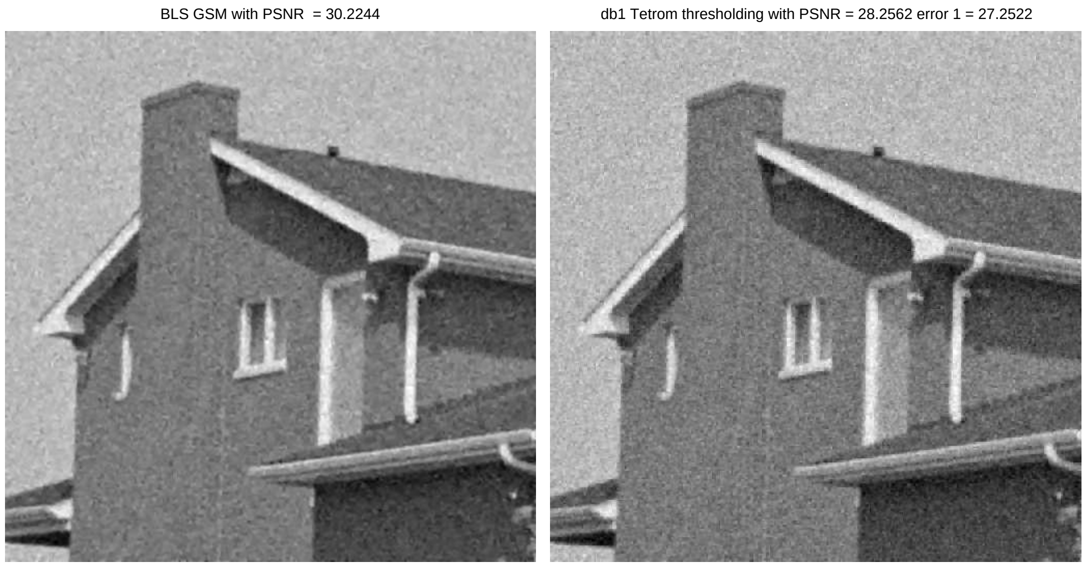


(c) MMSE shrinkage Michak Method 1



(d) MMSE shrinkage Michak Method 2

Figure 5.14. House Image Denoised II



(a) BLS-GSM (b) Tetrom

Figure 5.15. House Image Denoised III

5.7 Barbara Image Example

Figure 5.16 shows:

(a) Clean Barbara image of size 512x512

(b) Noisy Barbara image, noise of variance = 30 is added to image (a)

(c) Barbara image denoised by universal hard thresholding

(d) Barbara image denoised by universal soft thresholding

Denoised Images (c) and (d) are up to 3 dB better compared to the noisy one, but the visual appearance is still noisy. Further optimization is possible if we decompose the image further. Since the new method developed in this thesis uses only one level of decomposition, all compared methods have been kept to one level of decomposition for fairness.

Figure 5.17 shows:

(a) Barbara image denoised by SURE thresholding of Donoho and Johnstone [3]

(b) Barbara image denoised by Bayes Shrink method of Chang et al. [13]

(c) Barbara image denoised by Linear MMSE estimator method 1 of Michak et al. [14]

(d) Barbara image denoised by Linear MMSE estimator method 2 of Michak et al. [14]

Figure 5.18 shows:

(a) Denoised Barbara image by Gaussian scale mixture method by Portilla et al. [16]

(b) Denoised Barbara image by new method developed in this thesis

The results for the Barbara image are similar to the ones obtained for the Lena, Boat and House images. The best image is obtained by the Gaussian scale mixture method, which shows a 6 dB improvement. The second best image is produced by the method proposed in this thesis, with a 4 dB improvement. The proposed method betters other methods by performing upto 2 dB better. The performance of the proposed method is consistent across different natural images, even though they contain different natural objects with different features.

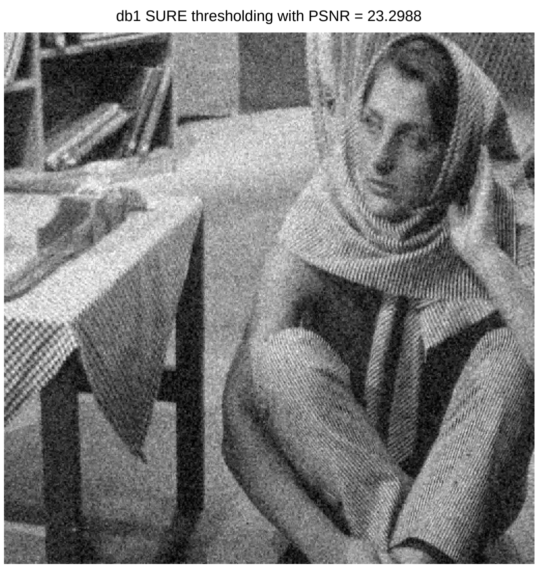
(a) Clean Image

(b) Noisy Image

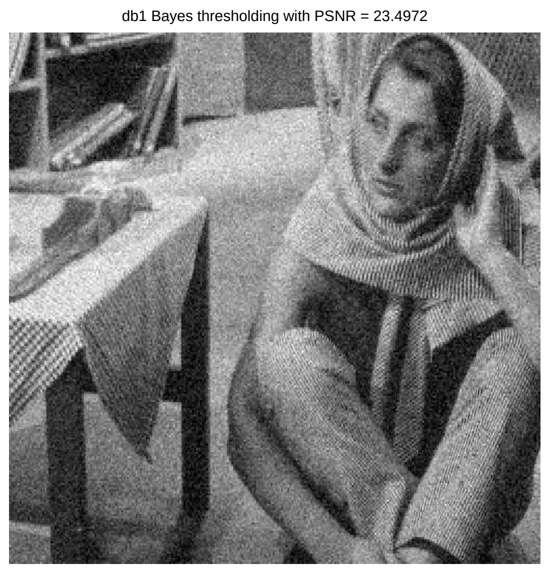
(c) VisuShrink Hard thresholding

(d) VisuShrink Soft thresholding

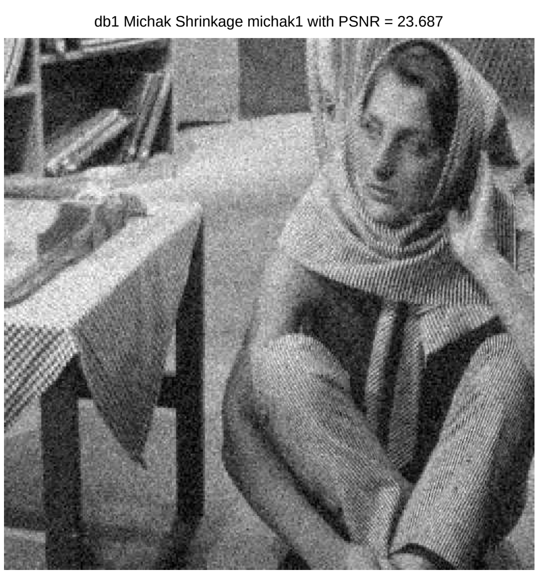
Figure 5.16. Barbara Image Denoised I



(a) SURE thresholding



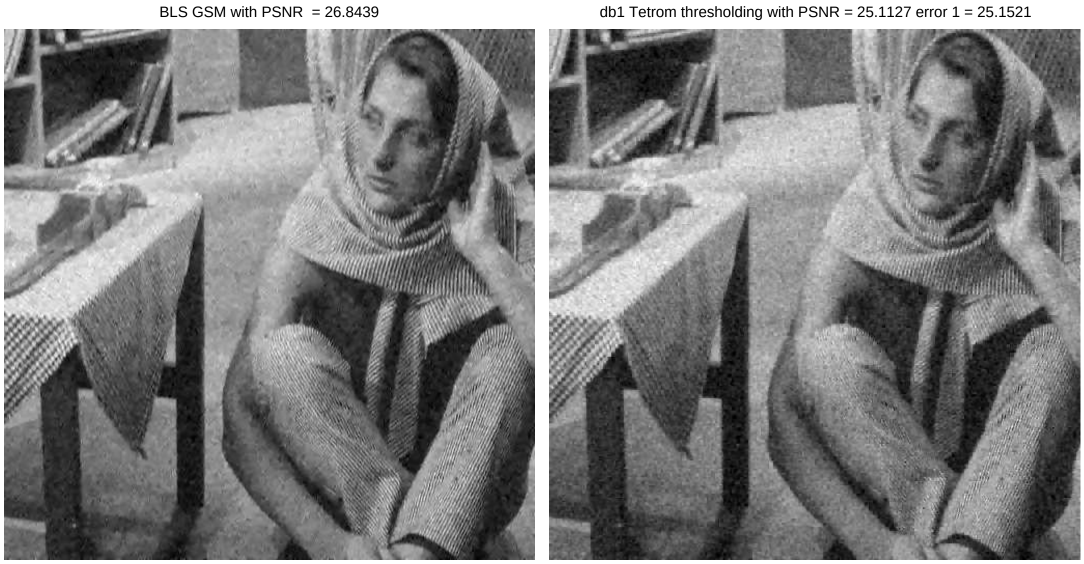
(b) Bayes thresholding



(c) MMSE shrinkage Michak Method 1

(d) MMSE shrinkage Michak Method 2

Figure 5.17. Barbara Image Denoised II

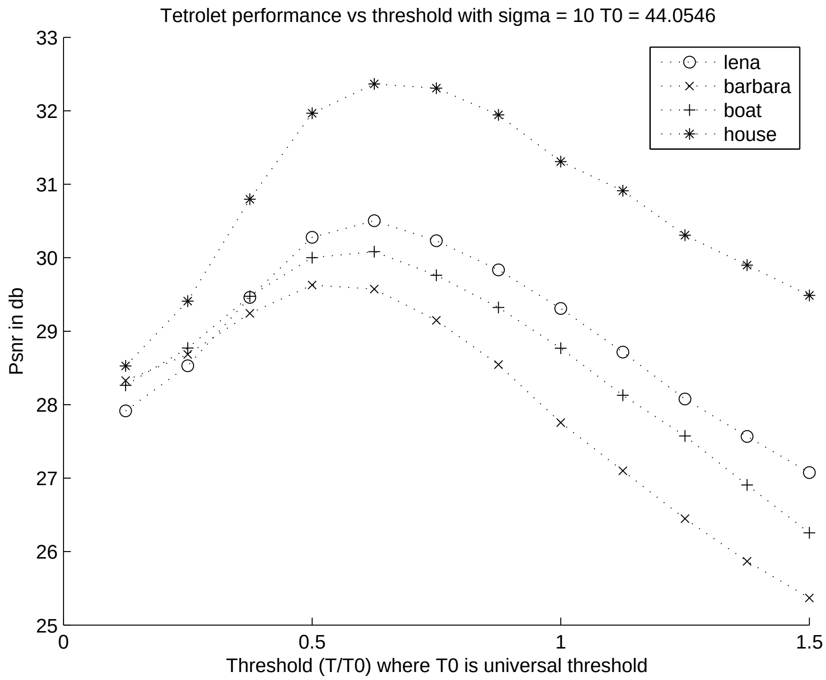


(a) BLS-GSM (b) Tetrom

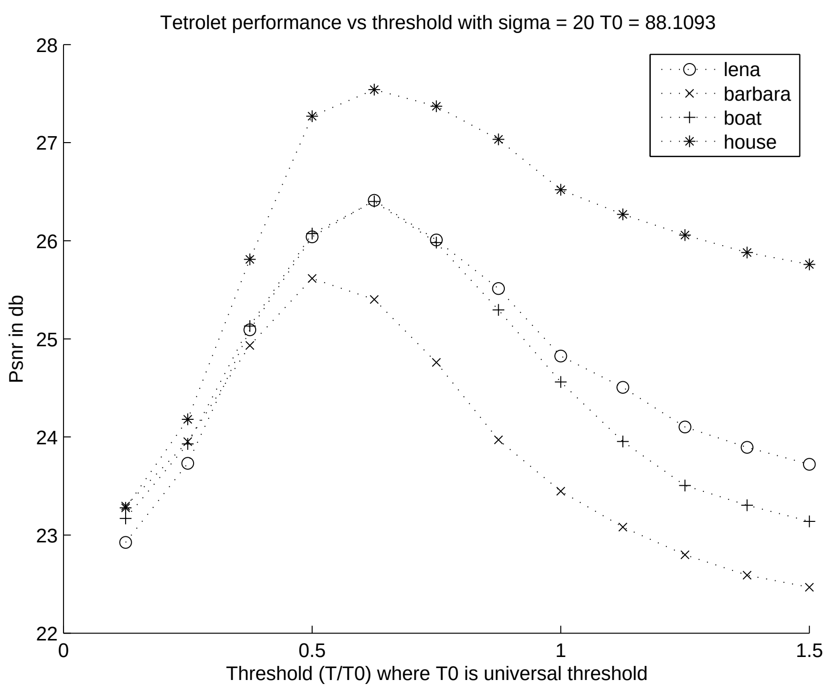
Figure 5.18. Barbara Image Denoised III

5.8 Tetrolet Transform Denoising Performance versus Threshold

A scaled universal threshold, as obtained by formula $T = S * \sigma\sqrt{2log(M)}$, where M is the number of pixels in the image and S is the scaling factor, is used. To obtain the scaling factor, the PSNR of the denoised image is plotted against the threshold value. The results are shown in Figure 5.19. A scaling factor of 0.68 produces optimal results on these images with different noise variance. In real systems, the scaling factor can be obtained by training on known images.



(a) Tetrom performance vs threshold at Sigma 10



(b) Tetrom performance vs threshold at Sigma 20

Figure 5.19. Tetrom Method’s Denoising Performance versus Threshold

5.9 Performance Tables

The four test images (Lena, Barbara, Boat and House) were corrupted with white noise, and denoised using different methods. The variance of the white noise is varied from 10 to 30 in steps of 5. The results are the PSNR values averaged over 10 runs with different random seeds. They are presented in Tables 5.1 and 5.2 and also in the Figures 5.20, 5.21, 5.22 and 5.23. Table 5.1 compares the proposed algorithm with other algorithms such as VisuHard, VisuSoft, Sure, Bayes, Michak1, and Michak2. Table 5.2 compares the proposed algorithm with the redundant Haar method and the Gaussian scale mixture method. The following observations can be drawn from these results:

- The Tetrom method performs, on an average, up to 3.63 dB better when compared with the VisuHard, VisuSoft, Sure, Bayes, Michak1, and Michak2 methods. It performs up to 1.9 dB better compared to the best of the above methods.
- BLS-GSM method performs up to 1.77 dB better than Tetrom, but the local nature and simplicity of the Tetrom algorithm are better suited for hardware implementation.
- The redundant Haar transform method and Tetrom method have similar performance. In some cases, the redundant Haar transform performs up to 0.49 dB better than the Tetrom method; However, in some cases, the Tetrom method performs up to 0.45 dB better than the redundant Haar transform. In visual analysis, the newly proposed method scores above the redundant Haar method. Despite having similar performance, these algorithms are not the same and do not generate the same coefficients. As seen in the visual comparison section, the Tetrom method outperforms the redundant Haar transform method.

PSNR (in dB) Comparison

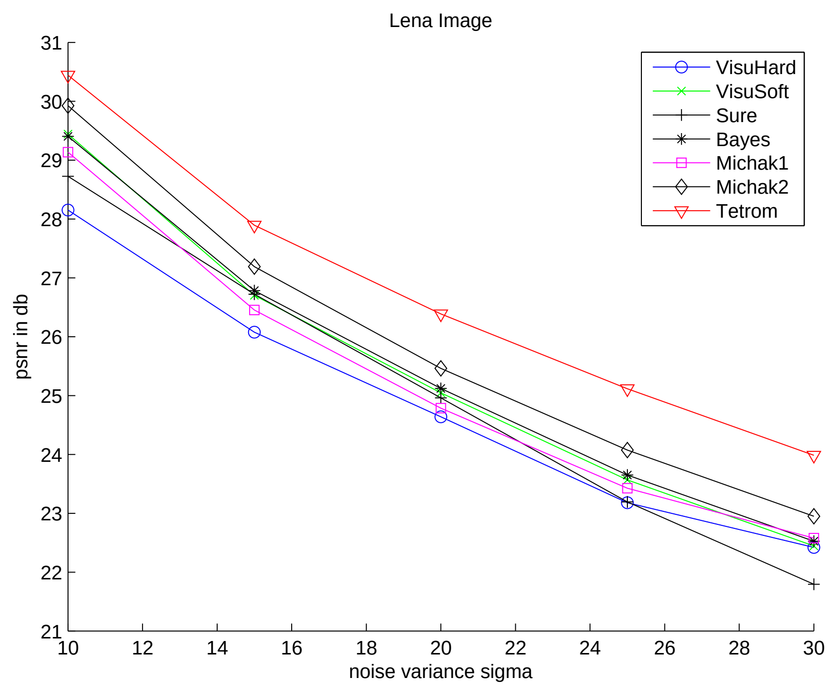
Table 5.1. PSNR Performance Table - 1

Image	VisuHard	VisuSoft	Sure	Bayes	Michak1	Michak2	Tetrom
lena(σ=10)	28.15	29.44	28.72	29.40	29.13	29.92	30.44
lena(σ=15)	26.08	26.70	26.72	26.78	26.45	27.18	27.89
lena(σ=20)	24.64	25.05	24.96	25.12	24.79	25.46	26.39
lena(σ=25)	23.18	23.57	23.19	23.65	23.42	24.07	25.12
lena(σ=30)	22.42	22.45	21.79	22.53	22.58	22.95	23.99
barabara(σ=10)	27.09	28.94	27.81	29.07	28.80	29.44	29.46
barabara(σ=15)	24.90	26.36	26.25	26.39	26.22	26.80	26.80
barabara(σ=20)	23.32	24.64	24.64	24.62	24.32	24.94	25.24
barabara(σ=25)	22.40	23.28	23.08	23.21	23.15	23.59	23.83
barabara(σ=30)	21.65	22.31	21.83	22.20	22.17	22.50	23.01
boat(σ=10)	27.92	29.27	28.40	29.24	29.02	29.55	29.85
boat(σ=15)	25.59	26.56	26.51	26.59	26.34	26.93	27.40
boat(σ=20)	24.11	24.73	24.67	24.80	24.62	25.11	25.85
boat(σ=25)	22.78	23.34	23.07	23.31	23.25	23.71	24.82
boat(σ=30)	22.21	22.42	21.83	22.37	22.42	22.77	23.77
house(σ=10)	30.50	30.52	30.46	30.53	30.68	31.18	32.31
house(σ=15)	28.31	27.78	27.98	28.19	27.97	28.48	29.75
house(σ=20)	26.03	25.59	25.44	26.07	26.12	26.46	28.06
house(σ=25)	24.92	24.40	23.87	24.74	24.88	25.18	27.05
house(σ=30)	23.69	22.92	22.10	23.21	23.60	23.83	25.73

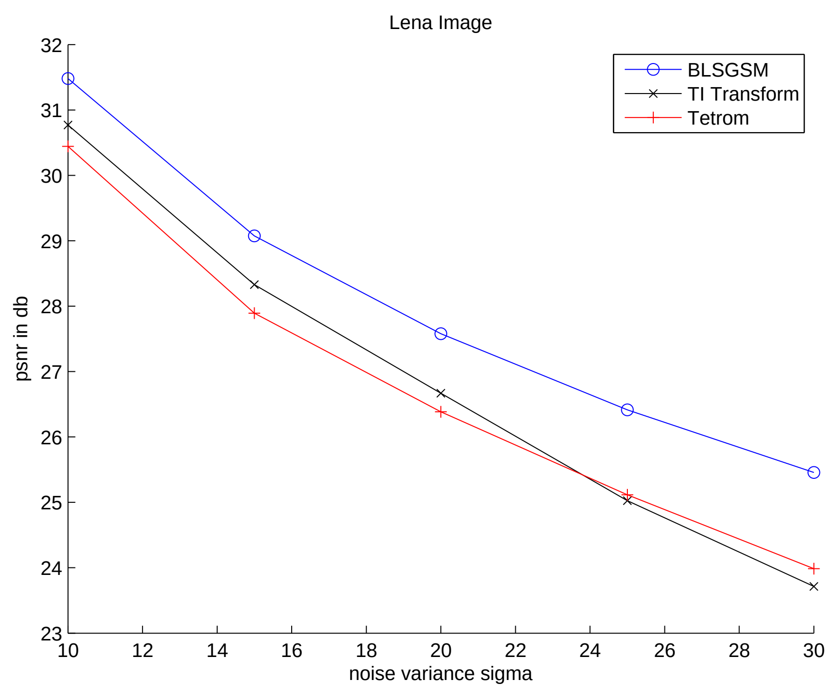
Table 5.2. PSNR Performance Table - 2

Image	BLS-GSM	Redundant Haar	Tetrom
lena(σ=10)	31.48	30.77	30.44
lena(σ=15)	29.07	28.33	27.89
lena(σ=20)	27.58	26.67	26.39
lena(σ=25)	26.42	25.03	25.12
lena(σ=30)	25.46	23.71	23.99
barabara(σ=10)	30.32	29.89	29.46
barabara(σ=15)	27.98	27.23	26.80
barabara(σ=20)	26.41	25.42	25.24
barabara(σ=25)	25.20	23.60	23.83
barabara(σ=30)	24.24	22.56	23.01
boat(σ=10)	30.52	30.13	29.85
boat(σ=15)	28.21	27.66	27.40
boat(σ=20)	26.75	25.91	25.85
boat(σ=25)	25.46	24.64	24.82
boat(σ=30)	24.70	23.35	23.77
house(σ=10)	33.51	32.80	32.31
house(σ=15)	31.43	30.21	29.75
house(σ=20)	29.83	28.21	28.06
house(σ=25)	28.62	26.74	27.05
house(σ=30)	27.48	25.34	25.73

The performance graphs in Figures 5.20, 5.21, 5.22, and 5.23 have two graphs each. Graph (a) compares our method with others where our performance is better. Graph (b) compares our method with the Gaussian scale mixture and the redundant Haar method. The performance of our method is less than the redundant Haar when the amount of noise is small, but surpasses it in higher noise scenarios. This is due to the higher degree of redundancy in our method.

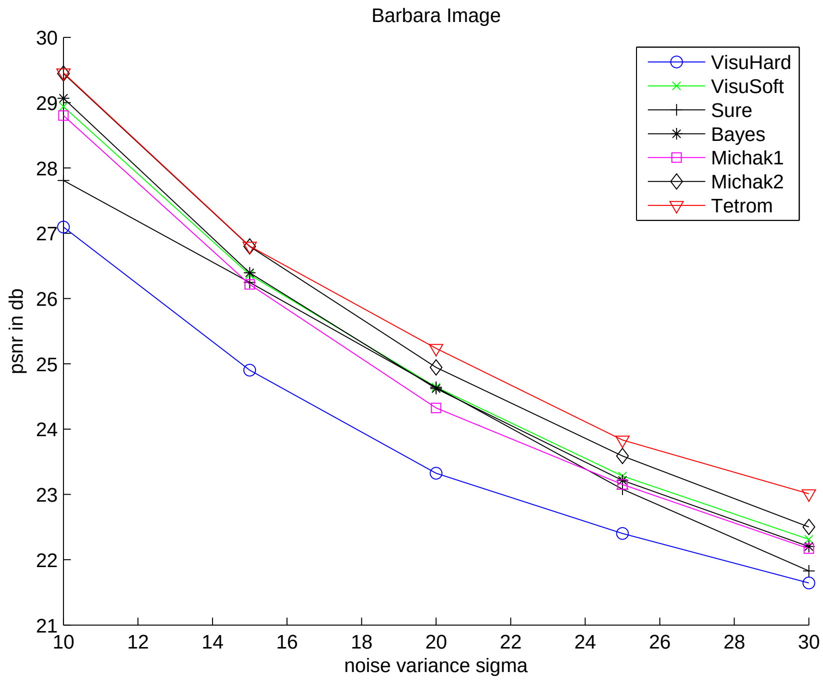


(a) Performance comparison with Lena image - 1

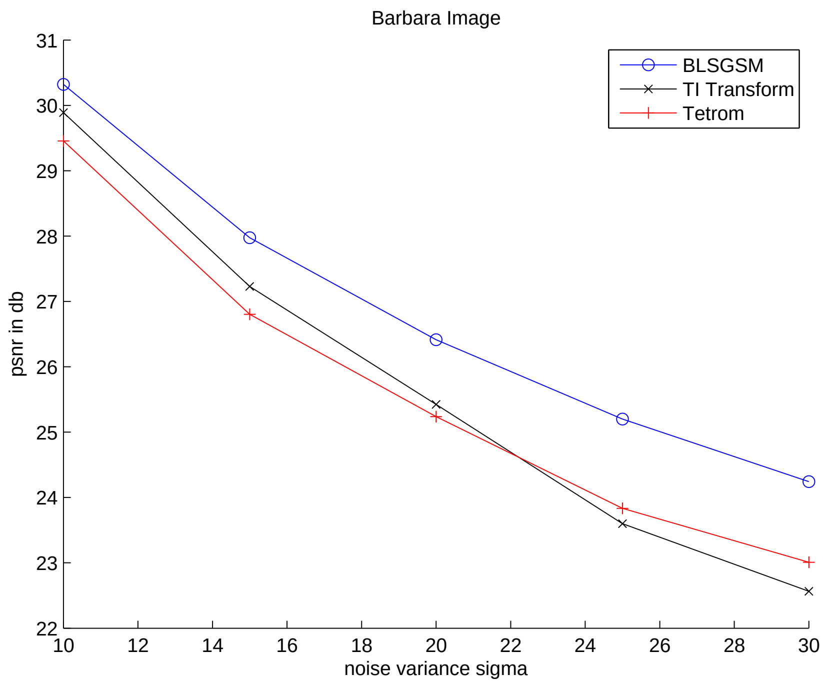


(b) Performance comparison with Lena image - 2

Figure 5.20. Performance Comparison with Different Methods - Lena Image

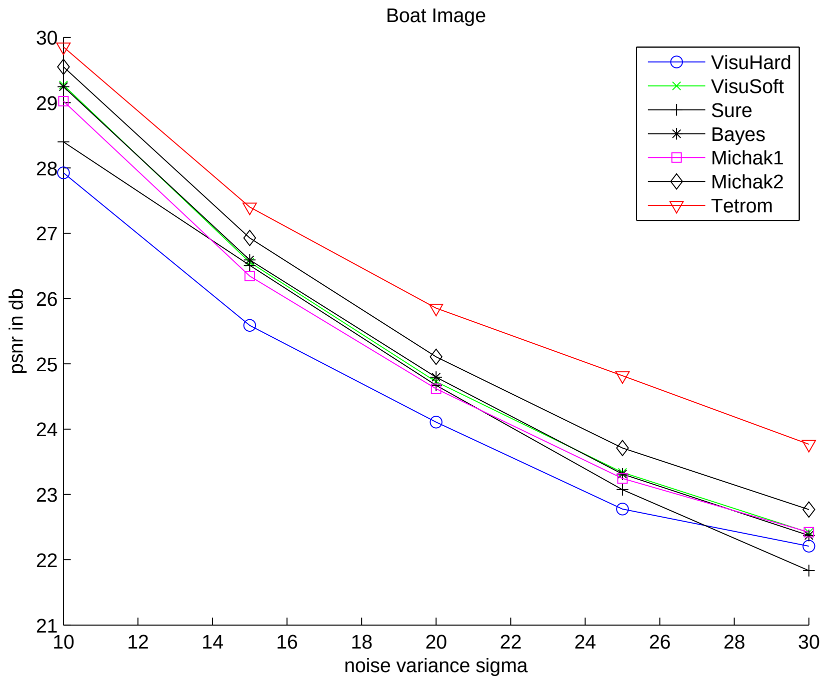


(a) Performance comparison with Barbara image - 1

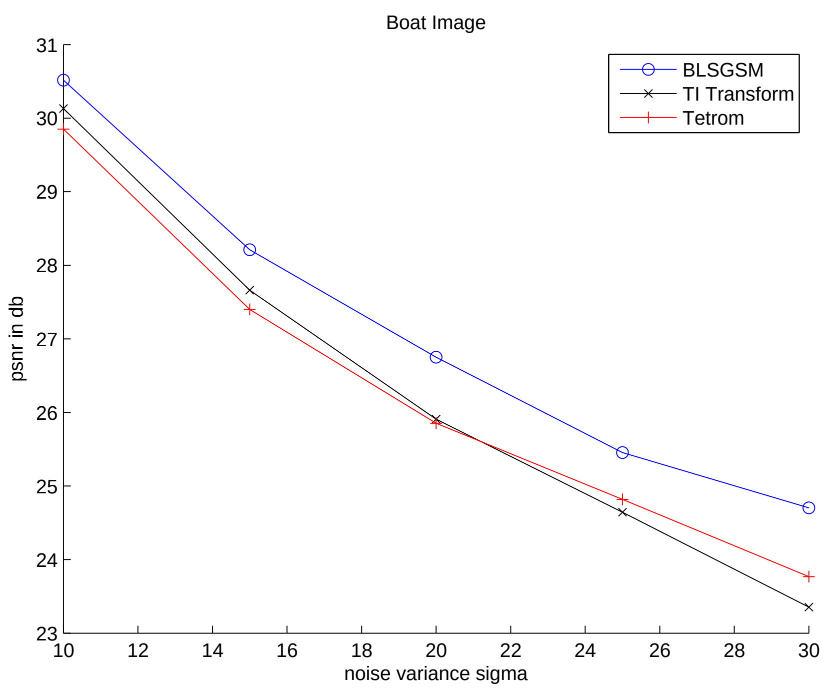


(b) Performance comparison with Barbara image - 2

Figure 5.21. Performance Comparison with Different Methods - Barbara Image

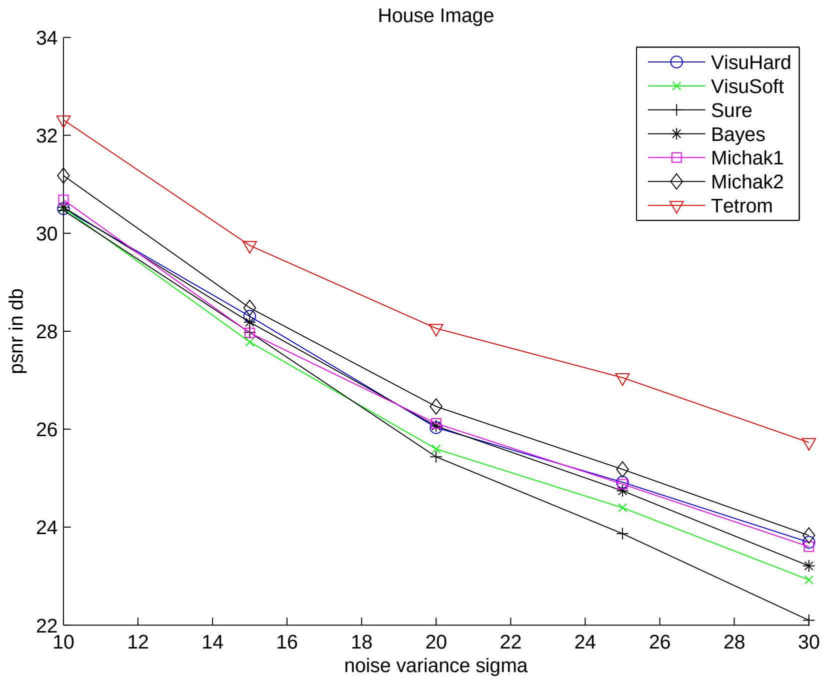


(a) Performance comparison with Boat image - 1

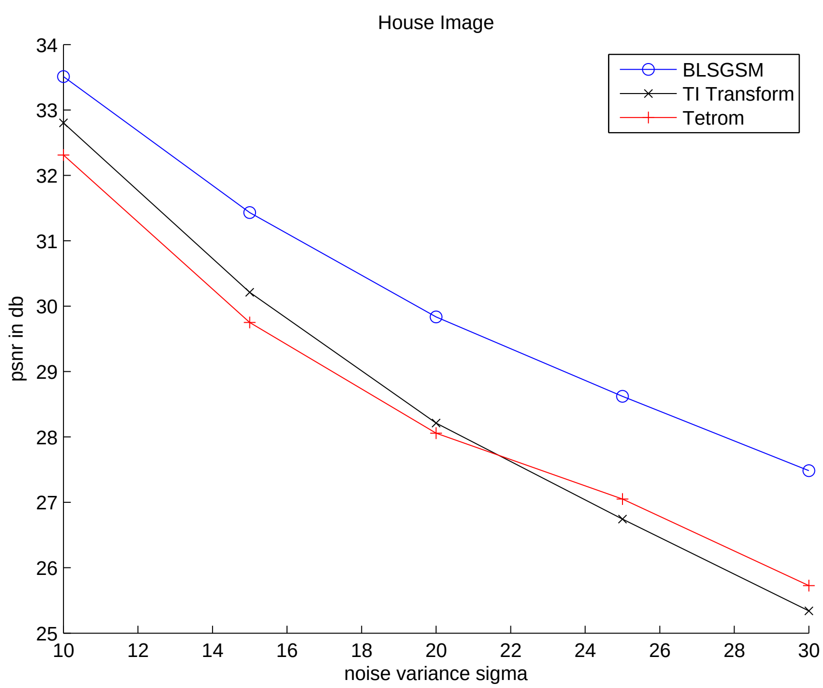


(b) Performance comparison with Boat image - 2

Figure 5.22. Performance Comparison with Different Methods - Boat Image



(a) Performance comparison with House image - 1



(b) Performance comparison with House image - 2

Figure 5.23. Performance Comparison with Different Methods - House Image

5.10 Residuals Analysis

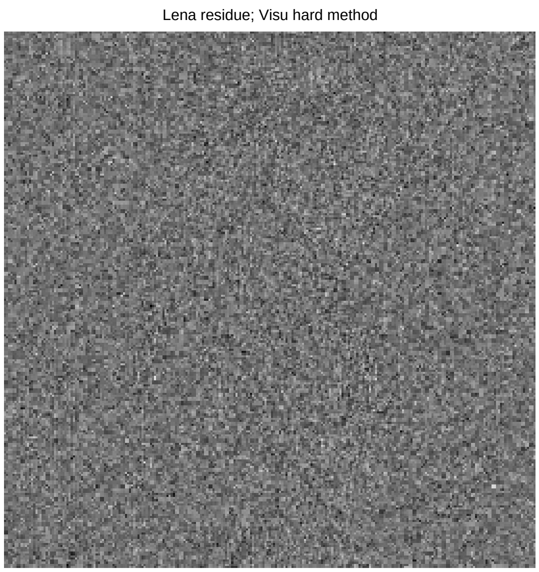
Buades et al. in [17] define a method called “noise” to compare the effectiveness of different denoising algorithms. The method is defined as follows:

$v = Dh(v) + n(Dh, v)$

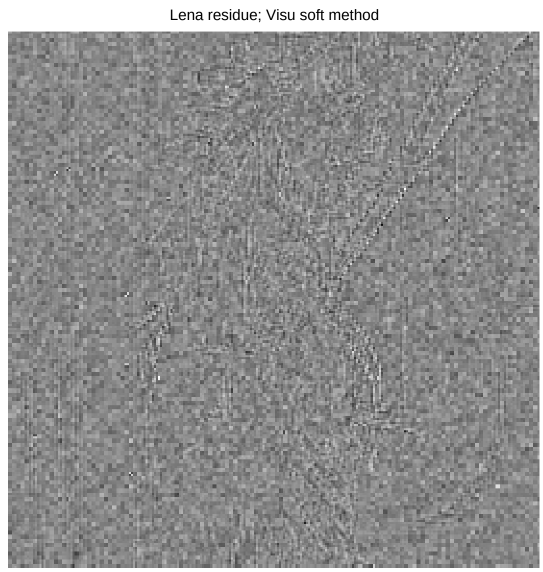
Here v is the noisy image and h is the filtering parameter which usually depends upon the standard deviation of noise. Dh(v) is the filtered image which is ideally smoother than v. n(Dh,v) is the realization of noise. The more this noise looks like white noise, the better is the result of the algorithm. If structures are visible in this noise, it implies that the filtering has removed some real fine structures of the image.

The residuals are calculated by taking the difference between the noisy and the denoised image. They are analyzed for visible image structures. It is noted that one can see image structures in the noise in our method, as well as in others. This means that these algorithms do remove fine structures in the image to some extent. Only the results for the Lena image are plotted here, but the results were similar across all the test images. Pixels are scaled and only the right top section of size 256x256 is plotted for better visibility in Figures 5.24, and 5.25. The original image size was 512x512 pixels.

The residuals in Figure 5.25 (d) shows that our method removes some details in the image. Even with this disadvantage, it outperforms other methods in terms of PSNR as well as subjective blind tests. This indicates that the algorithm has potential to achieve better results with the help of some improvements.

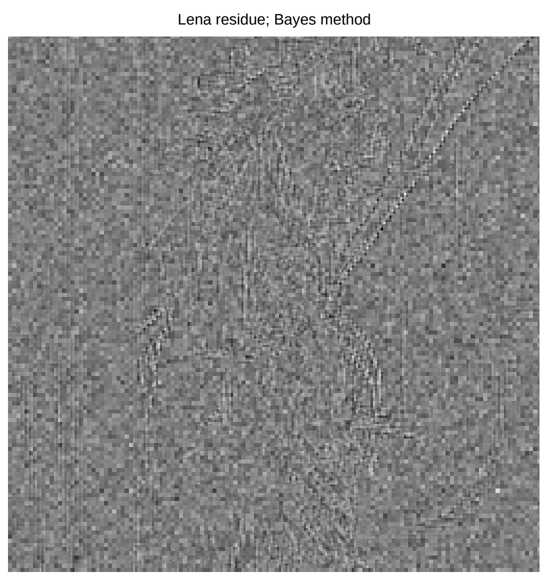


(a) VisuShrink Hard thresholding



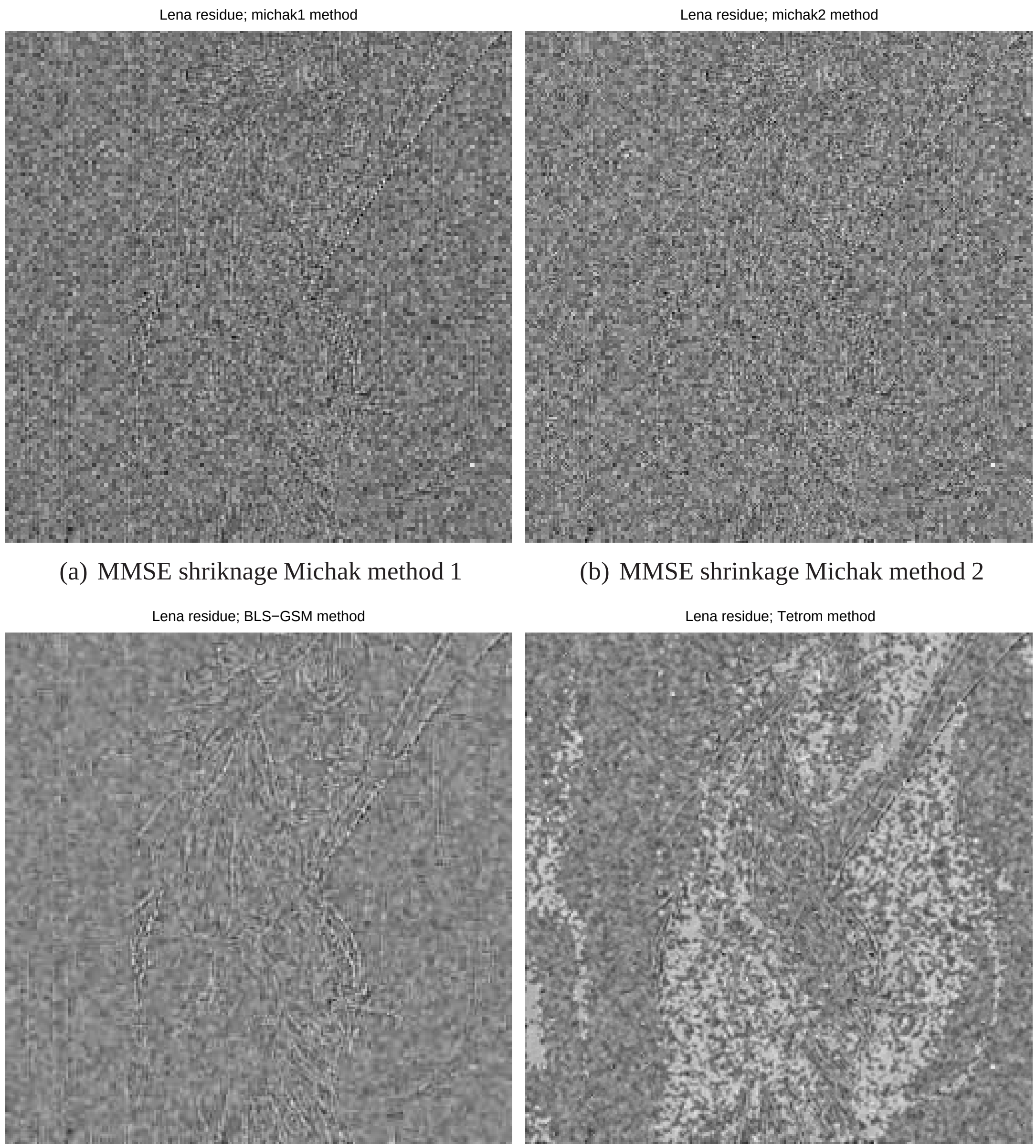
(b) VisuShrink Soft thresholding

(c) Sure thresholding



(d) Bayes thresholding

Figure 5.24. Lena Image Residuals Assessment I



(a) MMSE shriknage Michak method 1

(b) MMSE shrinkage Michak method 2

(c) BLS-GSM method

(d) Tetrom Method

Figure 5.25. Lena Image Residuals Assessment II

Chapter 6

Summary and Conclusions

In this thesis, several well known algorithms for denoising natural images were investigated and their performance was comparatively assessed. A new algorithm based on the so called Tetrolet transform (a descendant of the Haar wavelet transform) was developed. Its performance was shown to be competitive with or exceeding the performance of other algorithms. In addition, it has been shown to enjoy the advantage of implementation simplicity.

There are different types of noises that may corrupt a natural image in real life, such as shot noise, amplification noise, quantization noise etc. However, only zero-mean additive white Gaussian noise was considered because of it's simplicity.

A major part of the thesis was devoted to the review, implementation and performance assessment of published image denoising algorithms based on various techniques including the Wavelet transform. The Wavelet transform and its characteristics were studied. Multi resolution analysis (MRA) and Quadrature mirror filters (QMF) were examined to understand their relation with the the Wavelet transform. Denoising examples with 1D and 2D signals were presented. A one dimensional piece wise regular signal, corrupted with white noise, was denoised by moving average, Wiener and Wavelet methods, and their results were investigated. Similarly, the well known Lena image corrupted with AWGN was denoised using the moving average, Wiener2 filter and Wavelet methods. It was seen that the Wavelet methods yielded good results when denoising both 1D and 2D signals. Effects of different Wavelet bases on the denoising performance were examined. We also computed the histogram of the wavelet coefficients of four natural images as examples. The obtained histograms provided valuable information on the reasons for wavelets being a better choice for denoising natural images.

Different non-wavelet denoising algorithms such as Wiener filtering, moving average, median filtering and the non-local mean algorithm by Buades et al. [17] were studied. Different wavelet based denoising algorithms such as universal hard and soft thresholding methods, Sure Shrink method by Donoho and Johnstone [3], Bayes Shrink method by Chang et al. [13], Linear MMSE estimator methods by Michak et al. [14] and the Gaussian Scale Mixture method by Portilla et al. [16] were studied, implemented and their performance comparatively assessed.

Wavelets have proved to be good for denoising of natural images because of their energy compactness, sparseness and correlation properties. However, simple thresholding methods are limited in their denoising performance. Advanced wavelet methods such as the algorithm proposed by Portilla et al. [16] are too complex to be implemented in hardware for real time applications. Non local averaging methods such as the one proposed by Buades et al. [17] are very computationally intensive, and require large on-chip storage.

We proposed a new approach to the denoising problem based on the Tetrolet transform proposed by Jens Krommweh [21] for image compression. It is based on the Haar wavelet transform, but adapts to image characteristics automatically. Inspired by this idea, we came up with a simple Haar transform based denoising algorithm that works on each 4x4 sub-block of an image independently. The proposed approach requires only adders and shift registers. These properties make it a better choice for hardware implementations. Matlab simulations show up to 2 dB better performance compared to algorithms of similar complexity. Visual analysis also shows promising results. We asked people to vote for the least noisy image among a group of images denoised using several algorithms and collected statistics. Our method came in as second best after the method by Portilla et al. [16]. Given the simplicity and non local nature of our algorithm, it is better suited for real time hardware implementations.

In the proposed algorithm, we consider all the tetromino partitions. Determining the criteria to select the best or few best tetromino partitions among all the possible candidates can be the subject of future work. This way, the algorithm can become adaptive and adapt itself to any given image. The Non-Local-Mean algorithm [17] concept can be applied to select the best partition. While selecting the best partition for any given 4x4 block, information from other denoised blocks can be used. One possibility is that we can add weights to different tetromino partitions. We start with equal weights, but, as we progress through the picture, we change these weights. We increase the weights for tetromino partitions which we think are more probable. The simplest possibility is to increase the weight of those partitions which are being picked up for the current 4x4 block. This way, as we progress through the image, we give priority to the partitions which have already occurred. The underlying concept behind this idea is the presence of repeatability in the natural images. Taking the average of these repeated pixels or patches will result in denoising.

Bibliography

[1] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing.* India: Dorling Kindersley Pvt. Ltd., licensee of Pearson Education in South Asia, 2008, pp. 156, 311, 316-317.

[2] Wikipedia, "Image noise," Internet: http://en.wikipedia.org/wiki/Image_noise [Jan. 18, 2009].

[3] David L. Donoho and Iain M. Johnstone. "Adapting to Unknown Smoothness via Wavelet Shrinkage." *Journal of the American Statistical Association*, Vol. 90, No. 432, pp. 1200-1224, Dec. 1995.

[4] C. Sidney Burrus, Ramesh A. Gopinath, and Haitao Guo. *Introduction to Wavelets and Wavelet Transforms A Primer.* New Jersey: Prentice-Hall Pub., 1998, p. 6.

[5] David L. Donoho. "De-noising by soft-thresholding." *IEEE Trans. on Information Theory*, Vol 41, No. 3, May 1995.

[6] David L. Donoho. "Unconditional bases are optimal bases for data compression and for statistical estimation." *Applied and Computational Harmonic Analysis*, Vol. 1, No. 1, pp. 100-115, Dec. 1993.

[7] Dane Mackenzie. "WAVELETS: Seeing The Forest and the Trees," Internet: http://www.beyonddiscovery.org/content/view.page.asp?I=1959, 2001 [May 6, 2010].

[8] Anonymous. "The Portal to Science, Engineering and Technology. Resonance Publications, Inc.," Internet: http://www.resonancepub.com/wavelets.htm [May 06, 2010].

[9] Stephane G. Mallat. “A Theory for Multiresolution Signal Decomposition: The Wavelet Representation.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 11, No. 7, Jul. 1989.

[10] N. G. Kingsbury. “Complex wavelets for shift invariant analysis and filtering of signals.” *Journal of Applied and Computational Harmonic Analysis*, Vol. 10, No. 3, pp. 234-253, May 2001.

[11] Dongwook Cho. *Image Denoising Using Wavelet Transforms.* Germany: VDM Verlag Dr. Muller Aktiengesellschaft Co. 2008. p. 13.

[12] V. Balakrishnan, Nash Borges, and Luke Parchment. “Wavelet Denoising and Speech Enhancement.” Research paper, Johns Hopkins University, Baltimore, MD, 2001

[13] S. Grace Chang, Bin YU, and Martin Vetterli. “Adaptive Wavelet Thresholding for Image Denoising and Compression.” *IEEE Transactions on Image Processing*, Vol. 9, No. 9, pp. 1532-1546, Sep. 2000.

[14] M. K. Michak, Igor Kozintsev, Kannan Ramchandran, and Pierre Moulin. “Low-Complexity Image Denoising Based on Statistical Modeling of Wavelet Coefficients.” *IEEE Signal Processing Letters*, Vol. 6, No. 12, pp. 300-302, Dec. 1999.

[15] Levent Sundur, and Ivan W. Selesnick. “Bivariate Shrinkage Functions for Wavelet-Based Denoising Exploiting Interscale Dependency.” *IEEE Transactions on Signal Processing*, Vol. 50, No. 11, pp. 2744-2756, Nov. 2002.

[16] Javier Portilla, Vasily Strela, Martin J. Wainwright, and Eero P. Simoncelli. “Image Denoising Using Scale Mixtures of Gaussian in the Wavelet Domain.” *IEEE Transactions on Image Processing*, Vol. 12, No. 11, pp. 1338-1351, Nov. 2003.

[17] A. Buades, B. Coll, and J. M. Morel. “A review of image denoising algorithms, with a new one.” *Multiscale Model. Simul.*, Vol. 4, No. 2, pp. 490-530, Jul. 2005.

[18] Jin Wang, Yanwen Guo, Yiting Ying, Yanli Liu, and Qunsheng Peng. “Fast non-local Algorithm for Image Denoising.” *IEEE International conference on Image Processing*, pp. 1429-1432, Oct. 2006.

[19] Mark Miller, and Nick Kingsbury. “Image Denoising Using Derotated Complex Wavelet Coefficients.” *IEEE Transactions on Image Processing*, Vol. 17, No. 9, pp. 1500-1511, Sep. 2008.

[20] Anil K. Jain. *Fundamentals of Digital Image Processing*. India: Dorling Kindersley Pvt. Ltd., licensees of Pearson Education in Sour Asia, 2008, pp. 298-314.

[21] Jens Krommweh. “Tetrolet Transform: A New Adaptive Haar Wavelet Algorithm for Sparse Image Representation.” Research paper, Department of Mathematics, University of Duisburg-Essen, Germany, 2009.

[22] “Matlab Pyre Tool Box,” Internet: *http://www.cns.nyu.edu/∼eero/software.html* [Nov. 15, 2008].

[23] “Web form for denoising poll,” Internet: *http://spreadsheets.google.com/embeddedform?formkey=dFdWQzNqQXJqVzF3UDd5QlJFRWlmdmc6MA* [Feb. 23, 2010].

[24] “Web Poll Results,” Internet: *http://spreadsheets.google.com/oimg?key=0Ag6dOcwtG-xRdFdWQzNqQXJqVzF3UDd5QlJFRWlmdmc&oid=6&v=1267473608612* [Mar. 30, 2010].

[25] "Portilla BLS-GSM matlab software," Internet: *http://decsai.ugr.es/∼javier/denoise/software/index.htm* [Mar. 15, 2009].

[26] "Rice Wavelet Tool Box," Internet: *http://www.dsp.rice.edu/software/rwt.shtml* [Nov. 15, 2008].

Appendix A

Tetrominoe Shapes

Tetrominoes are shapes joined by 4 equal sized squares such that they connect with each other on at least one side. As shown in Figure A.1, there are five different shapes called free tetrominoes. These are the shapes in the popular computer game “Tetris”.

There are 22 basic tiling methods to cover a 4x4 region with the free tetrominoes, as shown in Figure A.2. Considering rotations and reflections there are totally 117 ways in which a 4x4 region can be covered with tetrominoes. These are shown in Figures A.3, A.4, and A.5.

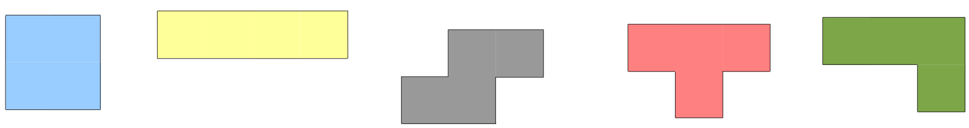


Figure A.1. Shapes of Free Tetrominoes

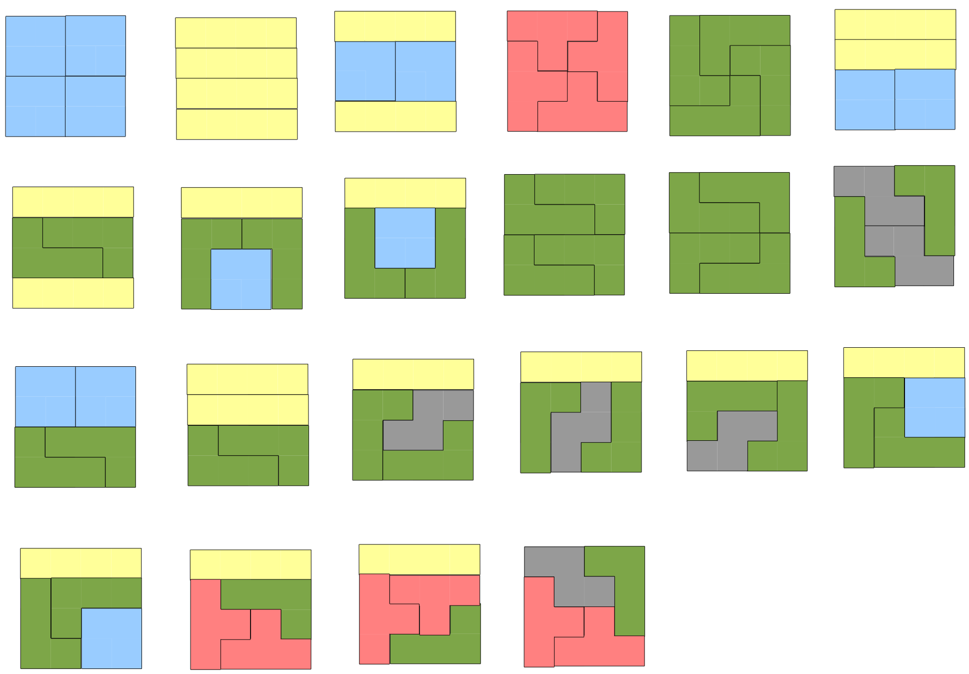


Figure A.2. 22 Different Basic Ways of Tetrolet Paritions for a 4x4 Block

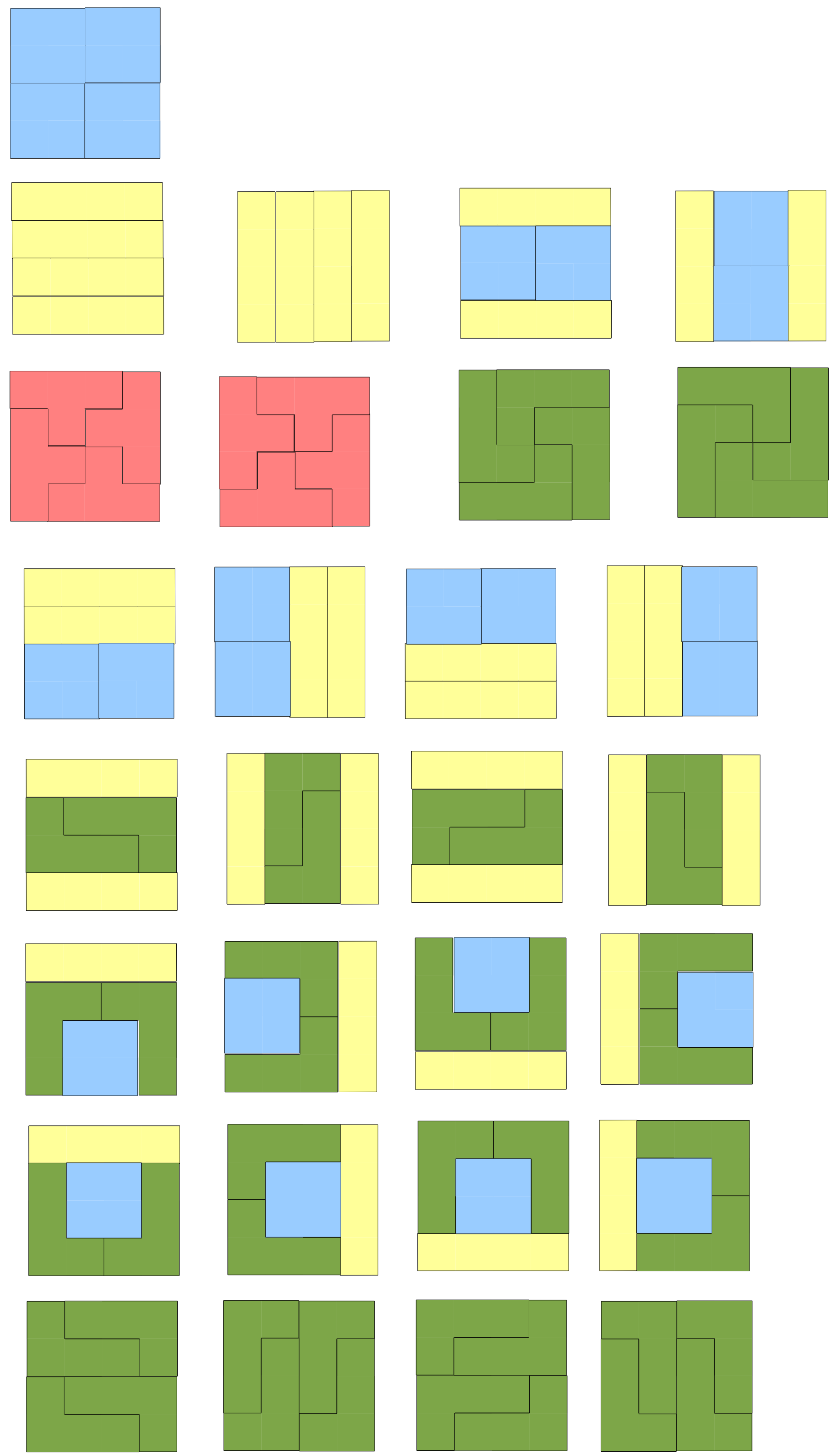


Figure A.3.   117 Different Ways of Tetrolet Partitions for a 4x4 Block (1 to 29)

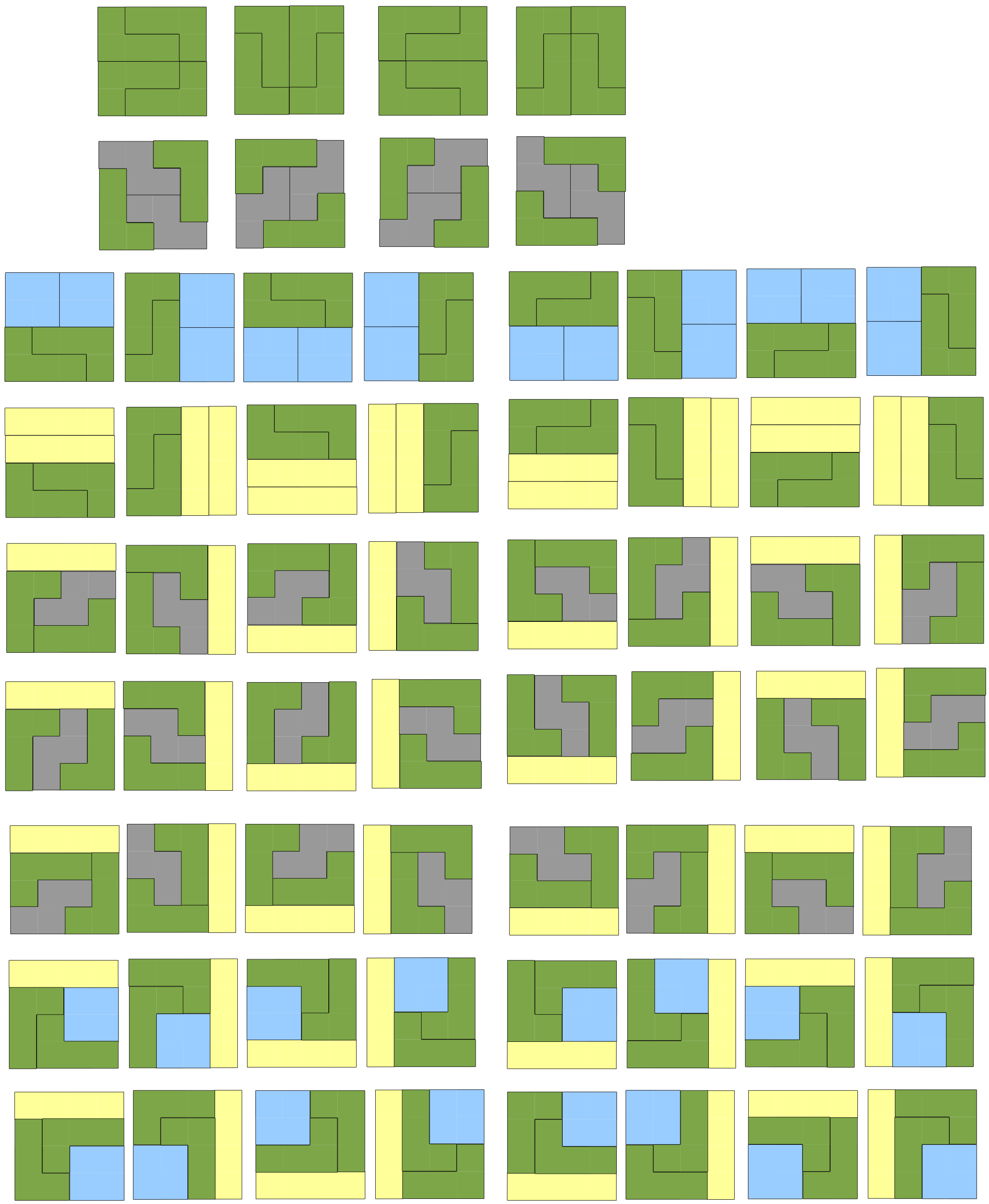


Figure A.4.   117 Different Ways of Tetrolet Partitions for a 4x4 Block (30 to 94)

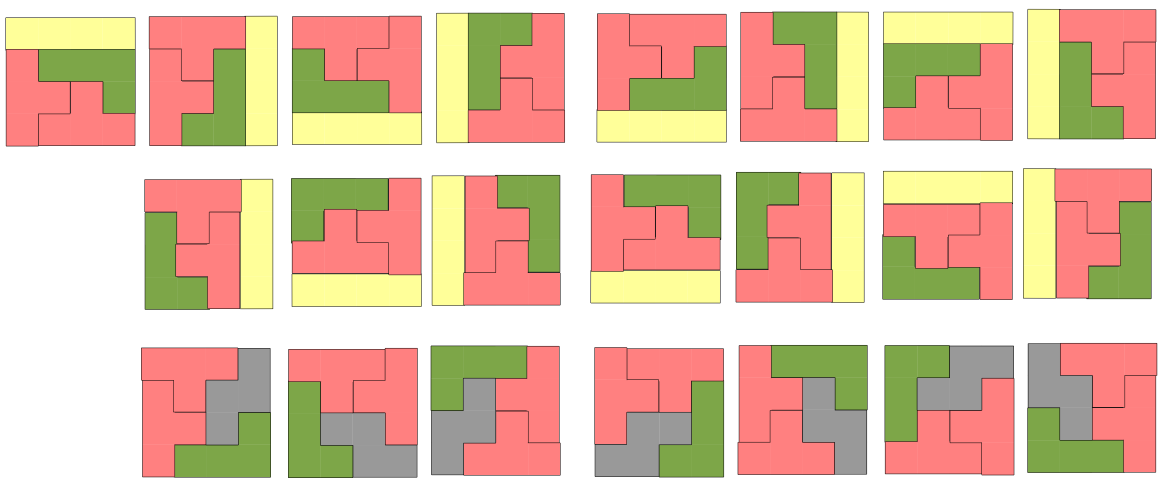


Figure A.5. 117 Different Ways of Tetrolet Partitions for a 4x4 Block (95 to 117)

Appendix B

Matlab Code

B.1 Functions

```
function denoise_image = denoise_image(imn, options, ...
              sigma, errtype, plot, im, printfname, dna)
%
% This program uses following third party programms.
% (1) Portilla BLS-GSM matlab software,
%     http://decsai.ugr.es/¬javier/denoise/software/index.htm
%
% imn = noisy image
% options - structure array with fields 'name' & 'params'
%         - 'name'   field is the name of the method.
%         - 'params' field is another structure with parameters
%                    related to method.
% Supported methods -
%  visu  : Do thresholding of wavelet coefficients based on universal
%          threshold.
%         (Reference: Unconditional bases are optimal bases for data
%              compression and for statistical estimation ...
%              by David L. Donoho,
%              Applied and Computational Harmonic Analysis,
%              1(1):100-115, December 1993
%              De-noising by soft-thresholding by David L. Donoho,
%              IEEE Transactions on Information Theory,
%              Vol. 41, No. 3, May 1995)
```

```
%        : params:
%                 incd = [0|1] (1 means threshold LL band, default 0)
%                 type = [Hard|Soft] (default Soft)
%                 wnam = name of the wavelet (default db8)
%                 decl = number of decomposition levels (default 4)
%
%  sure  : Do thresholding of wavelet coefficients based on SURE
%          method.
%         (Reference: Adapting to Unknown Smoothness via Wavelet
%                     Shrinkage by
%                     David L. Donoho and Iain M. Johnstone,
%                     Journal of the American Statistical Association,
%                     Vol. 90,  No. 432 (Dec., 1995), pp. 1200-1224)
%        : params:
%                 incd = [0|1] (1 means threshold LL band, default 0)
%                 wnam = name of the wavelet (default db8)
%                 decl = number of decomposition levels (default 4)
%
%  bayes : Do thresholding of wavelet coefficients based on Bayes
%          method.
%         (Reference:
%                Adaptive Wavelet Thresholding for Image Denoising
%                and Compression, by S. Grace Chang, Student Member,
%                IEEE, Bin YU, Senior Member, IEEE,
%        and Martin Vetterli, Fellow, IEEE,
%                IEEE Transactions on Image Processing,
%                Vol. 9, No. 9, September 2000)
%        : params:
%                 incd = [0|1] (1 means threshold LL band, default 0)
%                 wnam = name of the wavelet (default db8)
%                 decl = number of decomposition levels (default 4)
```

```
%
%  michak1 : Miachak Method 1
%      (Reference: Low-Complexity Image Denoising Based on Statistical
%                    Modeling of Wavelet Coefficients M. K, Michak,
%                    Igor Kozintsev, Kannan Ramchandran, Member, IEEE,
%                    and Pierre Moulin, Senior Member, IEEE
%                    [IEEE SIGNAL PROCESSING LETTERS, VOL. 6,
%               NO. 12, DECEMBER 1999])
%          : params:
%                   incd = [0|1] (1 means threshold LL band, default 0)
%                   wnam = name of the wavelet (default db8)
%                   decl = number of decomposition levels (default 4)
%                   wind = window size (2*l+1) for neigboring pixels to
%                          consider (default 3)
%
%  michak2 : Miachak Method 2
%      (Reference: Low-Complexity Image Denoising Based on Statistical
%                    Modeling of Wavelet Coefficients M. K, Michak,
%                    Igor Kozintsev, Kannan Ramchandran, Member, IEEE,
%                    and Pierre Moulin, Senior Member, IEEE
%                    [IEEE SIGNAL PROCESSING LETTERS, VOL. 6,
%              NO. 12, DECEMBER 1999])
%          : params:
%                   incd = [0|1] (1 means threshold LL band, default 0)
%                   wnam = name of the wavelet (default db8)
%                   decl = number of decomposition levels (default 4)
%                   wind = window size (2*l+1) for neigboring pixels to
%                         consider (default 1)
%
% BlsGsm : Bayesian Least square method using Gaussian Scale Mixture
%   (Reference: Image Denosing Using Scale Mixtures of Gaussians in
```

```
%           the Wavelet Domain, Javier Portilla, Vasily Strela,
%               Martin J. Wainwright, and Eero P. Simoncelli,
%               IEEE Transactions on Image Processing, Vol. 12,
%                 No. 11, November 2003)
% : params:
%           Nor = number of orientations (default 3, for X-Y separable
%           wavelets it can be only be 3)
%           repres1 = Type of pyramid (default 'uw' See help on
%                 "denoi_BLS_GSM" for possible choices)
%           repres2 = Type of wavelet (default 'daub1', see help on
%            "denoi_BLS_GSM" for posible choices)
%           blkSize = nxn coefficient neighborhood of spatial neigbors
%                     within the same subband, n must be odd)
%                        (default 3x3)
%           parent   = [1|0] 1 means include parent (default 0)
%           boundary = [1|0] 1 means boundary mirror extension
%                      (default 1)
%           covariance = [1|0] Full covariance matrix (1) or only
%                        diagonal elements (0) (default 1)
%           optim = [1|0] Bayes Least Squares solution (1), or
%                        MAP-Wiener solution in two steps (0)
%  Tetrom: Proposed tetrom based method
%       Works good among algoritm that are not non-local mean type, in
%       other words which are local to a particular region instead of
%       looking at whole picture.
%       Other advantages are - eaiser to implement, adaptive and
%       scalable in nature, Does not look beyond 4x4 region at a
%       time so easily fits in other encoding/decoding algorithms.
%     : params:
%        T0   = Threshold value (default is universal threshold *
%               3/4)
```

```
%        MaxC = Maximum Number of Tetrom Paritions that are considered
%        decl = number of decomposition levels (default 1)
%
%
%
% Nlm    : Non-Local mean algorithm (TBD)
%
%    (Reference: A non-local algorithm for image denoising Buades, A;
%    Coll,B.; Morel,J-M.; [Computer Vision and Pattern Recognition,
%    2005. CVPR 2005, IEEE Computer Society Conference on,
%    Volume 2, 20-25, June 2005, Pages: 60-65]
%
%
%
% optional parameters:
%
% errtype = 'a' -> determine absolute error
%         = 'm' -> determine mean square error (default)
%         = 's' -> determine SNR
%         = 'p' -> determine PSNR
%
% plot    = [1|0] : 1 plot, 0 no plot (default 0)
%
% sigma   = noise variance, if null then derive from HH
%           band using median
%
% im      = original image required for error calculation
%           if not given, them we will calculate the enerey
%           in the difference (noisy - recovered)
%           with referene to noise energy (sigma).
%
```

```
%   Copyright (c) 2009 Manish K. Singh
%

if nargin < 2
  display('imn and options argument are necessary, Please see help');
end

if nargin < 3
   sigma = find_sigma(imn);
end

if nargin < 4
  errtype = 'm';
  plot    = 0;
  im      = 'null';
end

if nargin < 7
   printfname = 'null'
end

if nargin < 8
  dna  = 0;
end

type = 'null';

switch errtype
  case 'm', errname = 'MSE';
  case 'p', errname = 'PSNR';
  case 'a', errname = 'ABS';
```

```
  case 's', errname = 'SNR';
  otherwise, display('Unknown method, see help');
end


% function returns list of errors for each method
denoise_image = [];

%% Find out how many methods
[t,NumMethods] = size(options);

% Plot coordinates
switch NumMethods
 case 1, px = 1; py = 1;
 case 2, px = 1; py = 1;
 otherwise, px = 1; py = 1;
end

% Iterate through all the methods
wnam_old = 'null';
decl_old = 0;
figcnt   = 0;

for method = 1:NumMethods
    if (plot)
      figcnt = figcnt + 1;
      if (figcnt > 1)
        figure;
        figcnt = 1;
      end
    end
```

```
    params = options(method).params;
    switch lower(options(method).name)

       %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
       %% Universal Threshold method
       %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
       case 'visu'
         % Parse the parameters
         if (¬isfield(params,'incd')), incd = 0;
         else incd = params.incd; end
         if (¬isfield(params,'type')), type = 'soft';
         else type = params.type; end
         if (¬isfield(params,'wnam')), wnam = 'db8';
         else wnam = params.wnam; end
         if (¬isfield(params,'decl')), decl = 4;
         else decl = params.decl; end

         % decompose the image if necessary
         if (¬strcmp(wnam_old,wnam) || decl_old ≠ decl)
            if (strcmp(wnam,'tetr'))
              [C,L,B] = tetrom2(imn,decl);
            else
             [C,L] = wavedec2(imn,decl,wnam);
            end
            wnam_old = wnam;
            decl_old = decl;
         end

         % Wavelet thresholding
         if (type == 'hard')
```

```
            opt.type = 'visu_hard';
        else
            opt.type = 'visu_soft';
        end
        opt.incd = incd;
        opt.sigma = sigma;
        CT = perform_wavelet_thresholding(C,L,opt);
        clear opt;

        % Reconstruct the image
    if (strcmp(wnam,'tetr'))
      im_hat = invtetrom2(CT,L,B);
        else
            im_hat = waverec2(CT,L,wnam);
        end

        % calculate error if original image is given
        if (¬strcmp(im,'null'))
            err = calculate_error(im,im_hat,errtype);
        end
      denoise_image = [denoise_image; ...
                     collect_image_statistics(im,im_hat)];

        % Plot the image
    fname = strcat(printfname,'_', ...
       lower(options(method).name),'_',type);
        if (plot)
            subplot(px,py,figcnt); image(im_hat); ...
             axis image; axis off; colormap gray(256);
            title([wnam, ' Universal thresholding (', type,...
                ') with ',errname,' = ' num2str(err)]);
```

```
            print('-deps',fname)
            end

            if (dna)
           t = strcat(fname,' method_noise');
           t
           method_noise(im, im_hat, t);
           end



          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
          %% sure Threshold method
          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
          case 'sure'
            % Parse the parameters
            if (¬isfield(params,'incd')), incd = 0;
        else incd = params.incd; end
            if (¬isfield(params,'wnam')), wnam = 'db8';
        else wnam = params.wnam; end
            if (¬isfield(params,'decl')), decl = 4;
        else decl = params.decl; end

            % decompose the image if necessary
            if (¬strcmp(wnam_old,wnam) || decl_old ≠ decl)
               if (strcmp(wnam,'tetr'))
                 [C,L,B] = tetrom2(imn,decl);
               else
                [C,L] = wavedec2(imn,decl,wnam);
               end
               wnam_old = wnam;
```

```
            decl_old = decl;
        end

        % Wavelet thresholding
        opt.type = 'sure';
        opt.incd = incd;
        opt.sigma = sigma;
        CT = perform_wavelet_thresholding(C,L,opt);
        clear opt;

        % Reconstruct the image
    if (strcmp(wnam,'tetr'))
      im_hat = invtetrom2(CT,L,B);
        else
      im_hat = waverec2(CT,L,wnam);
        end

        % calculate error if original image is given
        if (¬strcmp(im,'null'))
            err = calculate_error(im,im_hat,errtype);
        end
        denoise_image = [denoise_image; ...
                        collect_image_statistics(im,im_hat)];

        % Plot the image
    fname = strcat(printfname,'_',...
           lower(options(method).name),...
           '_',type);
        if (plot)
            subplot(px,py,figcnt); image(im_hat);
        axis image; axis off; colormap gray(256);
```

```
        title([wnam, ' SURE thresholding with ',...
           errname,' = ' num2str(err)]);
        print('-deps',fname)
        end

        if (dna)
       t = strcat(fname,'method_noise');
    method_noise(im, im_hat, t);
        end


      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      %% Bayes Threshold method
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      case 'bayes'
        % Parse the parameters
        if (¬isfield(params,'incd')), incd = 0;
    else incd = params.incd; end
        if (¬isfield(params,'wnam')), wnam = 'db8';
    else wnam = params.wnam; end
        if (¬isfield(params,'decl')), decl = 4;
    else decl = params.decl; end

        % decompose the image if necessary
        if (¬strcmp(wnam_old,wnam) || decl_old ≠ decl)
           if (strcmp(wnam,'tetr'))
             [C,L,B] = tetrom2(imn,decl);
           else
            [C,L] = wavedec2(imn,decl,wnam);
           end
           wnam_old = wnam;
```

```
            decl_old = decl;
         end

         % Wavelet thresholding
         opt.type = 'bayes';
         opt.incd = incd;
         opt.sigma = sigma;
         CT = perform_wavelet_thresholding(C,L,opt);
         clear opt;

         % Reconstruct the image
         if (strcmp(wnam,'tetr'))
            im_hat = invtetrom2(CT,L,B);
         else
             im_hat = waverec2(CT,L,wnam);
         end

         % calculate error if original image is given
         if (¬strcmp(im,'null'))
             err = calculate_error(im,im_hat,errtype);
         end
          denoise_image = [denoise_image; ...
                    collect_image_statistics(im,im_hat)];

         % Plot the image
         fname = strcat(printfname,'_',...
            lower(options(method).name),...
            '_',type);

         if (plot)
       subplot(px,py,figcnt); image(im_hat);
```

```
        axis image; axis off; colormap gray(256);
        title([wnam,' Bayes thresholding with ',...
               errname,' = ' num2str(err)]);
        print('-deps',fname)
          end

          if (dna)
           t = strcat(fname,'method_noise');
      method_noise(im, im_hat, t);
          end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %% michak1 method
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        case 'michak1'
          % Parse the parameters
          if (¬isfield(params,'incd')), incd = 0;
      else incd = params.incd; end
          if (¬isfield(params,'wnam')), wnam = 'db8';
      else wnam = params.wnam; end
          if (¬isfield(params,'decl')), decl = 4;
      else decl = params.decl; end
          if (¬isfield(params,'wind')), wind = 3;
      else wind = params.wind; end

          % decompose the image if necessary
          if (¬strcmp(wnam_old,wnam) || decl_old ≠ decl)
             if (strcmp(wnam,'tetr'))
               [C,L,B] = tetrom2(imn,decl);
             else
              [C,L] = wavedec2(imn,decl,wnam);
```

```
            end
            wnam_old = wnam;
            decl_old = decl;
         end

         % miachak1 shrinkage
         opt.type = 'michak_mmse_1';
         opt.l     = wind;
         opt.sigma = sigma;
         CT        = perform_wavelet_shrinkage(C,L,opt);
         clear opt;

         % Reconstruct the image
     if (strcmp(wnam,'tetr'))
       im_hat = invtetrom2(CT,L,B);
         else
             im_hat = waverec2(CT,L,wnam);
         end

         % calculate error if original image is given
         if (¬strcmp(im,'null'))
             err = calculate_error(im,im_hat,errtype);
         end
     denoise_image = [denoise_image; ...
                   collect_image_statistics(im,im_hat)];

         % Plot the image
         fname = strcat(printfname,'_',...
            lower(options(method).name),...
            '_',type);

```

```
    if (plot)
  subplot(px,py,figcnt); image(im_hat);
  axis image; axis off; colormap gray(256);
  title([wnam,' Michak Shrinkage ',...
         lower(options(method).name),...
             ' with ',errname,' = ' num2str(err)]);
  print('-deps',fname)
    end

    if (dna)
   t = strcat(fname,'method_noise');
method_noise(im, im_hat, t);
    end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  case 'michak2'
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Parse the parameters
    if (¬isfield(params,'incd')), incd = 0;
else incd = params.incd; end
    if (¬isfield(params,'wnam')), wnam = 'db8';
else wnam = params.wnam; end
    if (¬isfield(params,'decl')), decl = 4;
else decl = params.decl; end
    if (¬isfield(params,'wind')), wind = 1;
else wind = params.wind; end

    % decompose the image if necessary
    if (¬strcmp(wnam_old,wnam) || decl_old ≠ decl)
       if (strcmp(wnam,'tetr'))
         [C,L,B] = tetrom2(imn,decl);
```

```
            else
             [C,L] = wavedec2(imn,decl,wnam);
            end
            wnam_old = wnam;
            decl_old = decl;
        end

        % miachak1 shrinkage
        opt.type = 'michak_mmse_1';
        opt.l    = wind;
        opt.sigma = sigma;
        CT       = perform_wavelet_shrinkage(C,L,opt);
        clear opt;

        % Reconstruct the image
    if (strcmp(wnam,'tetr'))
      im_hat = invtetrom2(CT,L,B);
        else
            im_hat = waverec2(CT,L,wnam);
        end

        % calculate error if original image is given
        if (¬strcmp(im,'null'))
            err = calculate_error(im,im_hat,errtype);
        end
    denoise_image = [denoise_image; ...
                  collect_image_statistics(im,im_hat)];

        % Plot the image
        fname = strcat(printfname,'_',...
           lower(options(method).name),'_'...
```

```
            ,type);

         if (plot)
       subplot(px,py,figcnt); image(im_hat);
       axis image; axis off; colormap gray(256);
       title([wnam,' Michak Shrinkage ',lower(options(method).name),...
              ' with ',errname,' = ' num2str(err)]);
       print('-deps',fname)
         end

         if (dna)
        t = strcat(fname,'method_noise');
     method_noise(im, im_hat, t);
         end

       %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
       %% Tetrom
       %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
       case 'tetrom'
     [m,n] = size(imn);
         % Parse the parameters
         if (¬isfield(params,'T0')),
       T0 = sqrt(2*log(m*n))*sigma*0.68;
     else
       T0     = params.T0;
     end
         if (¬isfield(params,'MaxC')),
       MaxC = 117;
         else
       MaxC   = params.MaxC;
     end
```

```
        if (¬isfield(params,'decl')),
      dec  = 1;
    else
      decl   = params.decl;
    end
        if (¬isfield(params,'wnam')),
      wnam = 'haar';
    else
      wnam   = params.wnam;
    end

    % Form option for perform_tetrom_denoising function
        opt.L                = decl;
        opt.PrintStatistics = 0;
        opt.PrintStatFname  = 'none';
        opt.sigma           = sigma;
        opt.T               = T0;

        %% Now do tetrom based denoising
        i_hat_sum = zeros(n);
        for j=1:117
         %  opt.TilingGroup = j;
           opt.Tiling = j;
           % call the denoise function (tetrom)
           [f c_tetrom] = perform_tetrom_denoising(imn,opt,im);
           i_hat_sum = i_hat_sum+f;
        end
        im_hat = i_hat_sum./j;
        clear i_hat_sum;
        err_0 = calculate_error(im,im_hat,errtype);

```

```
        clear opt;
        [C,L] = wavedec2(im_hat,1,'db3');
        opt.sigma = sigma;
        thr = sqrt(2*log(length(C)))*sigma*1/8;
        CT = C.*(abs(C) > thr);
        clear opt;
        im_hat = waverec2(CT,L,'db3');

        % calculate error if original image is given
        if (¬strcmp(im,'null'))
            err = calculate_error(im,im_hat,errtype);
        end
        denoise_image = [denoise_image; ...
                        collect_image_statistics(im,im_hat)];

        % Plot the image
        fname = strcat(printfname,'_',...
           lower(options(method).name),...
           '_',type);

        if (plot)
      subplot(px,py,figcnt); image(im_hat);
      axis image; axis off; colormap gray(256);
      title([wnam,' Tetrom thresholding with ',errname, ...
             ' = ' num2str(err), ' error 1 = ', num2str(err_0)]);
      print('-deps',fname)
        end

        if (dna)
       t = strcat(fname,'method_noise');
    method_noise(im, im_hat, t);
```

```
        end



      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      %% Redundant using Pyre software
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      case 'redun'
        [m,n] = size(imn);
        if (¬isfield(params,'wnam')), wnam = 'haar';
    else wnam = params.wnam; end
        if (¬isfield(params,'decl')), decl = 4;
    else decl = params.decl; end
        if (¬isfield(params,'vm')),   vm   = 1;
    else vm   = params.vm  ; end
        if (¬isfield(params,'T0')),T0 = sqrt(2*log(m*n))*sigma*0.68;
    else T0     = params.T0;     end

        opt.wavelet_type = wnam;
        opt.wavelet_vm   = vm;
        Jmin = log2(m)-decl;
        opt.ti = 1;

        y  = perform_wavelet_transform(imn,Jmin,+1,opt);
        y  = y.*(abs(y) > T0);
        im_hat = perform_wavelet_transform(y,Jmin,-1,opt);
        clear y;

        % calculate error if original image is given
        if (¬strcmp(im,'null'))
            err = calculate_error(im,im_hat,errtype);
```

```
        end
        denoise_image = [denoise_image; ...
                      collect_image_statistics(im,im_hat)];

        % Plot the image
        fname = strcat(printfname,'_',...
           lower(options(method).name),...
           '_',type);

        if (plot)
       subplot(px,py,figcnt); image(im_hat);
       axis image; axis off; colormap gray(256);
       title([wnam,' Redundant thresholding with ',errname,' = '...
              num2str(err)]);
       print('-deps',fname)
        end

        if (dna)
       t = strcat(fname,'method_noise');
    method_noise(im, im_hat, t);
        end


      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      %% BlsGsm
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      case 'blsgsm'
        % Parse the parameters
        if (¬isfield(params,'Nor')),  Nor = 3;
    else Nor     = params.Nor;     end
        if (¬isfield(params,'repres1')), repres1 = 'uw';
```

```
        else repres1 = params.repres1; end
            if (¬isfield(params,'repres2')), repres2 = 'daub1';
        else repres2 = params.repres2; end
            if (¬isfield(params,'blkSize')), blkSize = [3 3];
        else blkSize = params.blkSize; end
            if (¬isfield(params,'parent')),   parent = 0;
        else parent  = params.parent; end
            if (¬isfield(params,'boundary')), boundary = 1;
        else boundary  = params.boundary; end
            if (¬isfield(params,'covariance')),  covariance = 1;
        else covariance  = params.covariance; end
            if (¬isfield(params,'optim')),  optim = 0;
        else optim  = params.optim; end

            % Use of software from portilla
        [Ny,Nx] = size(imn);
            PS      = ones(size(imn));
            if (¬isfield(params,'Nsc')), Nsc = 1;
            else Nsc = params.Nsc; end
            seed = 0;

        tic; im_hat = denoi_BLS_GSM(imn, sigma, PS, blkSize, parent,...
                        boundary, Nsc, Nor, covariance,...
                        optim, repres1, repres2, seed); toc
            % calculate error if original image is given
            if (¬strcmp(im,'null'))
                err = calculate_error(im,im_hat,errtype);
            end
        denoise_image = [denoise_image;...
                          collect_image_statistics(im,im_hat)];

```

```
        % Plot the image
        fname = strcat(printfname,'_',...
           lower(options(method).name));
        if (plot)
      subplot(px,py,figcnt); image(im_hat);
      axis image; axis off; colormap gray(256);
      title(['BLS GSM with ',errname,'  = ' num2str(err)]);
      print('-deps',fname)
        end

        if (dna)
       t = strcat(fname,'_method_noise');
    method_noise(im, im_hat, t);
        end

      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
       otherwise, display('Unknown method, see help');
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    end
end
```

```
function CT = perform_wavelet_thresholding(C,L,options)
%
% perform_wavelet_thresholding ->
% Do thresholding of wavelet coefficients.
%
% CT  = perform_wavelet_thresholding(C,L,options);
%
% C,L is the result of wavedec2 function in matlab.
```

```
% CT (result) can be directly used in waverec2 function in matlab.
%
% options.type :
%    visu_hard : universal hard thresholding (default)
%    visu_soft : universal soft thresholding
%    sure      : Sure thresholding method
%    bayes     : Bayes thresolding method
%
%  options.incd :
%    0  : Don't threshold average coefficients (default)
%    1  : Threhold average ceofficients
%  options.sigma :
%   v   : noise variance (default is 1)
%
%   Copyright (c) 2009 Manish K. Singh


%%% Parse options structure
options.null = 0;

if isfield(options, 'type')
   type = options.type;
else
   type = 'visu_hard';
end

if isfield(options, 'incd')
   incd = options.incd;
else
   incd = 0;
end
```

```matlab

if isfield(options, 'sigma')
    sigma = options.sigma;
else
    sigma = 1;
end


switch lower(type)

  case 'visu_hard'
       CT = visu_threshold(C,L,incd,'Hard',sigma);
  case 'visu_soft'
       CT = visu_threshold(C,L,incd,'Soft',sigma);
  case 'sure'
       CT = sure_threshold(C,L,incd,sigma);
  case 'bayes'
       CT = bayes_threshold(C,L,incd,sigma);
  otherwise
      error(['Unknown option type = ',type]);
end
```

```matlab
function CT = perform_wavelet_shrinkage(C,L,options)
%
% perform_wavelet_shrinkage ->
% X = y.C where y is the shrinkage factor.
%
% Usage:
% CT  = perform_wavelet_shrinkage(C,L,options);
```

```
%
% C,L is the result of wavedec2 function in matlab.
% CT (result) can be directly used in waverec2 function in matlab.
%
% options.type :
%    michak_mmse_1 : Michak method 1 (relevant arguments: options.l)
%                  : (default method)
%    michak_mmse_2 : Michak method 2 (relevant arguments: options.l)
%
%  (Reference: Low-Complexity Image Denoising Based on
%   Statistical Modeling of Wavelet Coefficients M. K,
%   Michak, Igor Kozintsev, Kannan Ramchandran, Member,
%   IEEE, and Pierre Moulin, Senior Member,
% IEEE [IEEE SIGNAL PROCESSING LETTERS, VOL. 6, NO. 12, DECEMBER 1999]
%
%  options.l: window size to estimate local parameters
%            (default l = 2*l+1)
%  options.sigma :
%   v   : noise variance (default is 1)
%  options.incd  : 0 (don't include average coefficients, default)
%                  1 (include average coefficients)
%
%   Copyright (c) 2009 Manish K. Singh


%%% Parse options structure
options.null = 0;

if isfield(options, 'type')
   type = options.type;
else
```

```
   type = 'michak_mmse_1';
end

if isfield(options, 'sigma')
   sigma = options.sigma;
else
   sigma = 1;
end

if isfield(options, 'l')
   l = options.l;
else
   l = 3;
end

if isfield(options,'incd')
  incd = options.incd;
else
  incd = 0;
end


switch lower(type)

  case 'michak_mmse_1'
       CT = michak_mmse_shrinkage(C,L,incd,sigma,l);

  case 'michak_mmse_2'
       CT   = michak_mmse_shrinkage(C,L,incd,sigma,l,'method2');

  otherwise
```

```
      error(['Unknown option type = ',type]);
end
```

```
function [f coeff] = perform_tetrom_denoising(I,options, Iclean)
%
% I -> noisy image
% f -> clean image (used in method p1; see below)
% options:
%  method -> 'L1', 'L2', 'T1','T2','s1','c1' (default 'l1')
%  These methods are criterians to select best tetrom partitions.
%   'l1' -> Minimize Sum of absolute values of detailed coefficients
%   'l2' -> Minimize Energy in detailed coefficients
%   't1' -> Maximize Number of detailed coefficients greater than
%           given threshold (T)
%   't2' -> Zero out detailed coefficients less than T, and then
%           maximise sum energy in the coefficients
%   's1' -> Minimize Standard Deviation of I
%   'c1' -> Maximize score = var*coeff_var + abs(I)*coeff_abs +
%           max(abs(I))*coeff_max,
%           where var_c + var_i + var_m = 1
%   'p1' -> Minimize mean squre error given clean image
%  T       -> threshold
%  L       -> Number of decompositions

sigma = 10;

if nargin < 2
  options.method = 'T1'
  options.T      = 50;
```

```
end

if ¬isfield(options,'method')
  options.method = 'L1';
end

if nargin < 3
 Iclean = I;
end

if isfield(options,'T')
 T = options.T;
end

if isfield(options,'L')
 L = options.L;
end

PrintStatistics = 0;
if isfield(options,'PrintStatistics')
 PrintStatistics = options.PrintStatistics;
 PrintStatFname  = options.PrintStatFname;
end

if (PrintStatistics)
 FidStat = fopen(PrintStatFname, 'a');
 fprintf(FidStat,'%s %s %s %s %s %s %s %s %s', ...
     ['blk :', 'TetromNo :', 'mean :', 'var :', 'mode :', ...
         'max :', 'min :', 'absI :', 'absI2 :']);
else
 FidStat = 'null';
```

```
end

% Get the dimensions
[m n] = size(I);

% Make sure m, and n are multiple of 4
if (mod(m,4))
  error('Picture size has to be multiple of 4');
end

if (mod(n,4))
  error('Picture size has to be multiple of 4');
end

%% TBD (Check for valid L)

ws = 4;  %% window size

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Do adaptive Haar on 4x4 window.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

TetromCoeff   = zeros(m,n);
TetromTiling  = [];

% We start with full Image, treated as coefficients
I_t = I;
I_tclean = Iclean;
BlkNo = 1;
MinEnergy = 2^32-1;
MaxEnergy = 0;
```

```
for dec=1:L
  TilingInfo    = [];
  [a b] = size(I_t);
  TetromCoeffA  = zeros(a/2,b/2);
  TetromCoeffH  = zeros(a/2,b/2);
  TetromCoeffV  = zeros(a/2,b/2);
  TetromCoeffD  = zeros(a/2,b/2);
  ridx = 1;
  cidx = 1;
  for r=1:ws:a
    for c=1:ws:b
        I4x4 = I_t(r:r+ws-1,c:c+ws-1);
        Iclean4x4 = I_tclean(r:r+ws-1,c:c+ws-1);
        if isfield(options,'Tiling')
          BestTile = options.Tiling;
          C4x4     = TetroletXform4x4(I4x4,options.Tiling);
%          c4x4_temp = C4x4;
%          c4x4_temp(1:2,1:2) = zeros(2);
%          EnergyInDetails = sum(c4x4_temp.^2);
%          if EnergyInDetails > MaxEnergy
%             MaxEnergy = EnergyInDetails;
%          end
%          if EnergyInDetails < MinEnergy
%             MinEnergy = EnergyInDetails;
%          end
%         AverageEnergy = (MaxEnergy + MinEnergy)/2;
%         EnergyThreshold_0 = (MinEnergy + AverageEnergy)/2;
%         EnergyThreshold_1 = (MaxEnergy + AverageEnergy)/2;
%         if EnergyInDetails > EnergyThreshold_1
%            T = options.T*5/4;
%          elseif EnergyInDetails < EnergyThreshold_0
```

```
%             T = options.T/2;
%          else
%             T = options.T*3/4;
%          end
%          T = find_sure_thres(c4x4_temp(:),sigma);
%           T = options.T;
%         c4x4_temp = SoftThresh(C4x4,T);
%          c4x4_temp = c4x4_temp.*(abs(c4x4_temp) > T);

%         c4x4_temp(1:2,1:2) = C4x4(1:2,1:2);
%         C4x4 = c4x4_temp;
        elseif isfield(options,'TilingGroup')
            switch options.TilingGroup
                    case 1, Start=1; End=1;
                    case 2, Start=2; End=3;
                    case 3, Start=4; End=5;
                    case 4, Start=6; End=7;
                    case 5, Start=8; End=9;
                    case 6, Start=10; End=13;
                    case 7, Start=14; End=17;
                    case 8, Start=18; End=21;
                    case 9, Start=22; End=25;
                    case 10, Start=26; End=29;
                    case 11, Start=30; End=33;
                    case 12, Start=38; End=45;
                    case 13, Start=46; End=53;
                    case 14, Start=54; End=61;
                    case 15, Start=62; End=69;
                    case 16, Start=70; End=77;
                    case 17, Start=78; End=85;
                    case 18, Start=86; End=93;
```

```
                    case 19, Start=94; End=101;
                    case 20, Start=102; End=109;
                    case 21, Start=110; End=117;
             end
                options.Start=Start;
                options.End  = End;
                [C4x4 BestTile] = GetBestTetromCoeff(I4x4,options,...
                 Iclean4x4, PrintStatistics, FidStat);
        else
          if (PrintStatistics)
            [meanV varV modeV maxV minV absIV ...
           absI2V] = Get4x4BlockStat(I4x4);
            fprintf(FidStat,'%d %d %f %f %f %f %f %f %f\n', ...
            [BlkNo 0 meanV varV modeV maxV minV absIV absI2V]);
          end
          [C4x4 BestTile] = GetBestTetromCoeff(I4x4,options,...
                              Iclean4x4,...
                              PrintStatistics, FidStat);
          if (PrintStatistics)
            [meanV varV modeV maxV minV absIV ...
           absI2V] = Get4x4BlockStat(C4x4);
            fprintf(FidStat,'%d %d %f %f %f %f %f %f %f\n', ...
        [BlkNo, BestTile, meanV varV modeV maxV minV absIV absI2V]);
            BlkNo = BlkNo + 1;
         end
        end
        TetromCoeffA(ridx:ridx+1,cidx:cidx+1) = C4x4(1:2,1:2);
        TetromCoeffH(ridx:ridx+1,cidx:cidx+1) = C4x4(1:2,3:4);
        TetromCoeffV(ridx:ridx+1,cidx:cidx+1) = C4x4(3:4,1:2);
        TetromCoeffD(ridx:ridx+1,cidx:cidx+1) = C4x4(3:4,3:4);
        TilingInfo             = [TilingInfo,BestTile];
```

```
        cidx = cidx+2;
    end
    ridx = ridx + 2;
    cidx = 1;
 end
 TetromCoeff(1:a/2,1:b/2)      = TetromCoeffA;
 TetromCoeff(1:a/2,b/2+1:b)    = TetromCoeffH;
 TetromCoeff(a/2+1:a,1:b/2)    = TetromCoeffV;
 TetromCoeff(a/2+1:a,b/2+1:b)  = TetromCoeffD;
 TetromTiling = [TilingInfo,TetromTiling];
 I_t          = TetromCoeffA;
 I_tclean     = zeros(a/2,b/2); %% TBD
end

clear I_t;
clear I_tclean;

coeff = TetromCoeff;

%%% plot best tiling for now
%figure
%x = 1:length(TetromTiling);
%plot(x,TetromTiling,'r+'); title('Teterom Tiling');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Thresholding
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% start from highest level
a = m/2^(L-1);
b = n/2^(L-1);
```

```
TetromCoeffA = TetromCoeff(1:a/2,1:b/2);

%TetromCoeffA = TetromCoeffA.*(abs(TetromCoeffA) > T/16);

for dec=1:L
%   figure
   TetromCoeffH    = TetromCoeff(1:a/2,b/2+1:b);
   TetromCoeffV    = TetromCoeff(a/2+1:a,1:b/2);
   TetromCoeffD    = TetromCoeff(a/2+1:a,b/2+1:b);

%   NumCoeffsGtT = sum((abs(TetromCoeffH(:)) > 0));
%   subplot(712); plot(TetromCoeffH(:)); ...
%   title(['Tetrominos coefficients H (Level= ', num2str(dec), ') ...
%           Coeff. Count = ', num2str(NumCoeffsGtT)]);

%   NumCoeffsGtT = sum((abs(TetromCoeffV(:)) > 0));
%   subplot(714); plot(TetromCoeffV(:)); ...
%   title(['Tetrominos coefficients V (Level= ', num2str(dec), ')...
%           Coeff. Count = ', num2str(NumCoeffsGtT)]);

%   NumCoeffsGtT = sum((abs(TetromCoeffD(:)) > 0));
%   subplot(716); plot(TetromCoeffD(:));
%   title(['Tetrominos coefficients D (Level= ', num2str(dec), ')...
%           Coeff. Count = ', num2str(NumCoeffsGtT)]);

   TetromCoeffH = TetromCoeffH.*(abs(TetromCoeffH) > (T/2^(dec-dec)));
   TetromCoeffV = TetromCoeffV.*(abs(TetromCoeffV) > (T/2^(dec-dec)));
   TetromCoeffD = TetromCoeffD.*(abs(TetromCoeffD) > (T/2^(dec-dec)));

   TetromCoeff(1:a/2,b/2+1:b)   = TetromCoeffH;
   TetromCoeff(a/2+1:a,1:b/2)   = TetromCoeffV;
```

```
   TetromCoeff(a/2+1:a,b/2+1:b) = TetromCoeffD;

%   NumCoeffsGtT = sum((abs(TetromCoeffA(:)) > 0));
%   subplot(711); plot(TetromCoeffA(:));
%   title(['Tetrominos coefficients A ', num2str(NumCoeffsGtT)]);

%   NumCoeffsGtT = sum((abs(TetromCoeffH(:)) > T));
%   subplot(713); plot(TetromCoeffH(:));
%   title(['Tetrominos coefficients thresholded H (Level= ', ...
%    num2str(dec), ')  Coeff. Count = ', num2str(NumCoeffsGtT)]);

%   NumCoeffsGtT = sum((abs(TetromCoeffV(:)) > T));
%   subplot(715); plot(TetromCoeffV(:));
%   title(['Tetrominos coefficients thresholded V (Level= ', ...
%   num2str(dec), ')  Coeff. Count = ', num2str(NumCoeffsGtT)]);

%   NumCoeffsGtT = sum((abs(TetromCoeffD(:)) > T));
%   subplot(717); plot(TetromCoeffD(:));
%   title(['Tetrominos coefficients thresholded D (Level= ', ...
%   num2str(dec), ')  Coeff. Count = ', num2str(NumCoeffsGtT)]);

   a = a*2;
   b = b*2;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Inverse transform
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

f = zeros(m,n);
i = 1;
```

```
% start from highest level
a = m/2ˆ(L-1);
b = n/2ˆ(L-1);
f = TetromCoeff;

for dec=1:L
  ridx = 1;
  cidx = 1;
  t    = zeros(a,b);

  TetromCoeffA = f(1:a/2,1:b/2);
  TetromCoeffH = f(1:a/2,b/2+1:b);
  TetromCoeffV = f(a/2+1:a,1:b/2);
  TetromCoeffD = f(a/2+1:a,b/2+1:b);

  for r=1:ws:a
    for c=1:ws:b
       I4x4          = zeros(4);
       I4x4(1:2,1:2) = TetromCoeffA(ridx:ridx+1,cidx:cidx+1);
       I4x4(1:2,3:4) = TetromCoeffH(ridx:ridx+1,cidx:cidx+1);
       I4x4(3:4,1:2) = TetromCoeffV(ridx:ridx+1,cidx:cidx+1);
       I4x4(3:4,3:4) = TetromCoeffD(ridx:ridx+1,cidx:cidx+1);
       t(r:r+ws-1,c:c+ws-1)=InvTetroletXform4x4(I4x4,TetromTiling(i));
       i = i+1;
       cidx=cidx+2;
    end
    ridx=ridx+2;
    cidx = 1;
  end
  % update Tetrom Coefficients
```

```
  f(1:a,1:b) = t;
  a = a*2;
  b = b*2;
end

if (PrintStatistics)
  fclose(FidStat)
end
```

```
function CT = visu_threshold(C,L,incd,type,sigma)

% visu_threshold -> Do thresholding of wavelet coefficients based
%                   on universal threshold
%                -> Reference: Donoho papers,
%                   It also uses functions HardThresh and SoftThresh
%                   from Wavelab.
%
% CT  = visu_threshold(C,L,incd,type);
%
% C,L is the result of wavedec2 function in matlab.
% CT (result) can be directly used in waverec2 function in matlab.
%
% type :
%    'hard' : hard threshold method
%    'soft' : soft threshold method

%  incd :
%    0  : Don't threshold average coefficients (default)
%    1  : Threhold average ceofficients
```

```
%
%   Copyright (c) 2009 Manish K. Singh


CT = [];
thr = sqrt(2*log(length(C)))*sigma;

%% Reduce the soft threshold,
%% because generally threshold is too large.
%%
if strcmp(type,'Soft'),
 thr = thr*2/8;
end

thr=thr*3/4

if incd == 0
  mn = L(1,:); m=mn(1); n=mn(2);
  cD = C(m*n+1:end);
  if strcmp(type,'Hard'),
     CT = [C(1:m*n),HardThresh(cD,thr)];
  else
     CT = [C(1:m*n),SoftThresh(cD,thr)];
  end
else
  if strcmp(type,'Hard'),
     CT = HardThresh(C,thr);
  else
     CT = SoftThresh(C,thr);
  end
end
```

```matlab
function thre = BayesThres(y,sigma);
%
% Estimate bayes threshold as
%
%  T = sigmaN^2/sigmaS
%
%  Reference:
% Adaptive Wavelet Thresholding for image denoising and compression
%            By S. Grace Chang etc.
%  sigmaS = sqrt(max((sigmaY^2 - sigmaN^2),0))
%  sigmaY = 1/N(sum(Y^2))
%
% In case of SigmaS is 0, set the threshold to be minimum value.
%
%   Copyright (c) 2009 Manish K. Singh

n = length(y);
y = y - mean(y); % Shift it so mean becomes 0.
sigmaYSquare = (1/n)*sum(y.^2);
sigmaS = sqrt(max((sigmaYSquare-sigma^2),0));

if sigmaS == 0
  sigmaS = max(y(:)); % this will set the threshold to low
end

thre   = sigma^2/sigmaS;
```

```
function CT = sure_threshold(C,L,incd,sigma)

% sure_threshold
%  -> Do thresholding of wavelet coefficients based on SURE
%  -> level based thresholding
%
% CT  = sure_threshold(C,L,incd,sigma);
%
% C,L is the result of wavedec2 function in matlab.
% CT (result) can be directly used in waverec2 function in matlab.
%
%  incd :
%    0  : Don't threshold average coefficients (default)
%    1  : Threhold average ceofficients
%
%   Copyright (c) 2009 Manish K. Singh


% FindOut number of decompositions
DecLevels = length(L)-2;

CT      = [];
index   = 1;

%% Average coefficients
mn = L(1,:); m=mn(1); n=mn(2);
y  = C(1:m*n);
if (incd == 0)
  CT = [CT,y];
else
```

```
  t  = find_sure_thres(y,sigma);
  CT = [CT,SoftThresh(y,t)];
end
index = m*n+1;

%% Detail coefficients
for i = 2:(DecLevels+1)
  mn = L(i,:); m=mn(1); n=mn(2);
  for j = 1:3 %% 3 loops for horizontal, vertical and diagonal details
     y  = C(index:index+m*n-1);
     index = index+m*n;
     t  = find_sure_thres(y,sigma);
     CT = [CT,SoftThresh(y,t)];
  end
end
```

```
function thres = find_sure_thres(x,sigma)
% find_sure_thres -- Adaptive Threshold Selection Using
%                    principle of SURE
%
%  Description
%    SURE referes to Stein's Unbiased Risk Estimate.
%    Reference:
%      Wavelet Denoising and Speech Enhancement
%      By V. Balakrishnan, Nash Borges, Luke Parchment
%
%  lamda = arg min SURE(x,thres)
%
% SURE(x,thres) =
```

```
% sigma^2+1/n(sum(min(abs(x),thres)^2))- ...
%           2*sigma^2/n*sum(abs(x) < thres)
%
%   Copyright (c) 2009 Manish K. Singh
%

n           = length(x);
thre_range = linspace(0,sqrt(2*log(n)),20); %
r_list     = [];

for t = thre_range
  thres   = t;
  r       = (n*sigma^2-2*sigma^2*(sum(abs(x) < thres))...
            + sum(min(abs(x),thres).^2))/n;
  r_list = [r_list,r];
end
[tmp,i] = min(r_list); thres = thre_range(i);

%% Multiply it with log10(n) to achieve the better performance.

thres = log10(n)*thres;
```

```
function CT = bayes_threshold(C,L,incd,sigma)
%
% bayes_threshold -> Do thresholding of wavelet coefficients
%                     based on bayes method
%
% CT  = bayes_threshold(C,L,incd,sigma);
%
```

```
% C,L is the result of wavedec2 function in matlab.
% CT (result) can be directly used in waverec2 function in matlab.
%
%  incd :
%    0  : Don't threshold average coefficients (default)
%    1  : Threhold average ceofficients
%
%
% sigma is estimated if not provided.
%
%   Copyright (c) 2009 Manish K. Singh


%% TBD: add sigma calculation logic.

% FindOut number of decompositions
DecLevels = length(L)-2;

%% Average coefficients
CT = [];
mn = L(1,:); m=mn(1); n=mn(2);
y  = C(1:m*n);
if (incd == 0)
  CT = [CT,y];
else
  t  = BayesThres(y,sigma);
  CT = [CT,SoftThresh(y,t)];
end
index = m*n+1;

%% Detail coefficients
```

```
for i = 2:(DecLevels+1)
  mn = L(i,:); m=mn(1); n=mn(2);
  for j = 1:3 %% 3 loops for hor., vert. and diag. details
     y  = C(index:index+m*n-1);
     index = index+m*n;
     t  = BayesThres(y,sigma);
     CT = [CT,SoftThresh(y,t)];
  end
end
```

```
function CT = michak_mmse_shrinkage(C,L,incd,sigma,l,method)

% michak_mmse_shrinkage ->
%  Do thresholding of wavelet coefficients based on
%  wavelet shrinkage method suggested by Michak
%
% C,L is the result of wavedec2 function in matlab.
% CT (result) can be directly used in waverec2 function in matlab.
%
%  incd :
%    0  : Don't threshold average coefficients
%    1  : Threhold average ceofficients
%
%  sigma is the noise variance.
%  l spcifies the window size - 2*l+1
%
% Optional arguments:
%  method = method1 or method2 (Reference:
% Low-Complexity Image Denoising Based on Statistical Modeling of
```

```
% Wavelet Coefficients M. K, Michak, Igor Kozintsev, Kannan
% Ramchandran, Member, IEEE, and Pierre Moulin, Senior Member, IEEE
% [IEEE SIGNAL PROCESSING LETTERS, VOL. 6, NO. 12, DECEMBER 1999]
%
%
%   Copyright (c) 2009 Manish K. Singh

if nargin < 6
  method = 'method1';
end

% FindOut number of decompositions
DecLevels = length(L)-2;

CT      = [];
index   = 1;

%% Average coefficients
mn = L(1,:); m=mn(1); n=mn(2);
y  = C(1:m*n);
if (incd == 0)
  CT = [CT,y];
else
  CTM = michak_mmse(y,m,n,sigma,l,'method1');
  if (method == 'method2')
     lambda = 1/std(CTM);
     CT = [CT,michak_mmse(y,m,n,sigma,l,'method2',lambda)];
  else
     CT = [CT,CTM];
  end
end
```

```
index = m*n+1;

%% Detail coefficients
for i = 2:(DecLevels+1)
  mn = L(i,:); m=mn(1); n=mn(2);
  %% 3 loops for horizontal, vertical and diagonal details
  for j = 1:3
     y  = C(index:index+m*n-1);
     index = index+m*n;
     CTM = michak_mmse(y,m,n,sigma,l,'method1');
     if (method == 'method2')
       lambda = 1/std(CTM);
       CT = [CT,michak_mmse(y,m,n,sigma,l,'method2',lambda)];
     else
       CT = [CT,CTM];
    end
  end
end
```

```
function CT = michak_mmse(C,m,n,sigma,l,method,lambda,bext_type)
%
% Usage:
  % CT  = michak_mmse(C,m,n,sigma,window,lambda,bext_type);
%
% C, is the result of wavedec2 function in matlab.
% CT (result) can be directly used in waverec2 function in matlab.
% m is number of rows, n is number of columns. mxn is image size.
% sigma is noise variance
  % bext_type = extension method  (default : 'sym');
```

```
% (all methods supported in "wextend" wavelet matlab toolbox)
%
% l = specified the neighbour hood (window size = 2*l+1)
%  (Reference: Low-Complexity Image Denoising Based on
%   Statistical Modeling of Wavelet Coefficients M. K,
%   Michak, Igor Kozintsev, Kannan Ramchandran, Member,
%   IEEE, and Pierre Moulin, Senior Member,
%   IEEE [IEEE SIGNAL PROCESSING LETTERS, ...
%          VOL. 6, NO. 12, DECEMBER 1999]
%
%    X(k) = Y(k)*(sigmaXK^2)/(sigmaXK^2+sigma^2)
%    sigmaXK = (1/M)*(sum(Y(j)^2-sigma^2)) where sum is taken
%              over a window around the coefficient
%
%   Copyright (c) 2009 Manish K. Singh
%

if nargin < 7
  lambda = 1;
end

if nargin < 8
   bext_type = 'sym';
end

% Boundary extension of the image
CM = bextend_wavelet_coeffs(C,m,n,l,bext_type);

CT = [];
for i = 1:m
  for j = 1:n
```

```
    N = get_window_pixels(CM,m,n,i,j,l);
    M = (2*l+1)^2;
    if method == 'method2'
      varxk = ((M/(4*lambda))*(-1+sqrt(1+(8*lambda/M^2)...
                        *sum(N.^2))))-sigma^2;
      if varxk < 0
        varxk = 0;
      end
    else
      varxk = (1/M)*sum((N.^2)-sigma^2);
      if varxk < 0
        varxk = 0;
      end

    end
    y  = C((i-1)*n+j);
    ym = y*(varxk)/(varxk+sigma^2);
    CT = [CT, ym];
  end
end
```

```
function f = TetroletXform4x4(I,C)
%
% Perform Tetrolet Transform on 4x4 block given tetrominos
% tiling C. It will return a list matix with [A W0; W1 W2 ]
% where
% A,W0, W1 and W2 are 2x2 matrices.

% Collect 4 pixels as per tetrominoes tiling.
```

```
% Each column will contain one group of pixels.

t = GetTetromPermMatrix4x4(C);
t = t(:);
Imod = zeros(4);
for col=1:4
  for row=1:4
  Imod(col,row)= I(t((col-1)*4+row));
  end
end
I = Imod;

clear Imod, t;

% Do the haar transform
W = [1 1 1 1; 1 1 -1 -1; 1 -1 1 -1; 1 -1 -1 1];
W = 0.5.*W;
f = [W(1,1:4)*I;W(2,1:4)*I;W(3,1:4)*I;W(4,1:4)*I];

% Now put them into correct order
% TBD (We can threshold detailed coefficients here)
r = zeros(4);
f = f';
r(1,1) = f(1);
r(2,1) = f(2);
r(1,2) = f(3);
r(2,2) = f(4);

r(3,1) = f(5);
r(4,1) = f(6);
r(3,2) = f(7);
```

```
r(4,2) = f(8);

r(1,3) = f(9);
r(2,3) = f(10);
r(1,4) = f(11);
r(2,4) = f(12);

r(3,3) = f(13);
r(4,3) = f(14);
r(3,4) = f(15);
r(4,4) = f(16);

f = r;
```

```
function f = InvTetroletXform4x4(I,C)
%
% Perform Tetrolet inverse Transform on 4x4 block given
% tetrominos tiling C. It will return 4x4 matix.

% Reorder coefficients so that we perform Haar
% filtering.
I_r = zeros(4);
I_r(1,:) = [I(1,1) I(3,1) I(1,3) I(3,3)];
I_r(2,:) = [I(2,1) I(4,1) I(2,3) I(4,3)];
I_r(3,:) = [I(1,2) I(3,2) I(1,4) I(3,4)];
I_r(4,:) = [I(2,2) I(4,2) I(2,4) I(4,4)];
I = I_r';

clear I_r;
```

```
% Do the haar transform
W = [1 1 1 1; 1 1 -1 -1; 1 -1 1 -1; 1 -1 -1 1];
W = 0.5.*W;
f = [W(1,1:4)*I;W(2,1:4)*I;W(3,1:4)*I;W(4,1:4)*I];

% Now put them into correct order
t = GetTetromPermMatrix4x4(C);
t = t';
t = t(:);
r = zeros(4);

for i=1:16
   r(t(i)) = f(i);
end

f = r;
```

```
function [C S B] = tetrom2(I,L)
%
% Tetrom decomposition
%

% Get the dimensions
[m n] = size(I);

% Make sure m, and n are multiple of 4
if (mod(m,4))
  error('Picture size has to be multiple of 4');
```

```
end

if (mod(n,4))
  error('Picture size has to be multiple of 4');
end

%% TBD (Check for valid L)

ws = 4;  %% window size

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Do adaptive Haar on 4x4 window.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

TetromCoeff   = zeros(m,n);
C             = [];
TetromTiling  = [];

% We start with full Image, treated as coefficients
I_t = I;
for dec=1:L
  TilingInfo    = [];
  [a b] = size(I_t);
  TetromCoeffA  = zeros(a/2,b/2);
  TetromCoeffH  = zeros(a/2,b/2);
  TetromCoeffV  = zeros(a/2,b/2);
  TetromCoeffD  = zeros(a/2,b/2);
  ridx = 1;
  cidx = 1;
  for r=1:ws:a
    for c=1:ws:b
```

```
        I4x4 = I_t(r:r+ws-1,c:c+ws-1);
        [C4x4 BestTile] = GetBestTetromCoeff(I4x4);
        TetromCoeffA(ridx:ridx+1,cidx:cidx+1) = C4x4(1:2,1:2);
        TetromCoeffH(ridx:ridx+1,cidx:cidx+1) = C4x4(1:2,3:4);
        TetromCoeffV(ridx:ridx+1,cidx:cidx+1) = C4x4(3:4,1:2);
        TetromCoeffD(ridx:ridx+1,cidx:cidx+1) = C4x4(3:4,3:4);
        TilingInfo             = [TilingInfo,BestTile];
        cidx = cidx+2;
    end
    ridx = ridx + 2;
    cidx = 1;
  end
 TetromCoeff(1:a/2,1:b/2)      = TetromCoeffA;
 TetromCoeff(1:a/2,b/2+1:b)    = TetromCoeffH;
 TetromCoeff(a/2+1:a,1:b/2)    = TetromCoeffV;
 TetromCoeff(a/2+1:a,b/2+1:b)  = TetromCoeffD;
 TetromTiling = [TilingInfo,TetromTiling];
 I_t          = TetromCoeffA;
 C      = [TetromCoeffH(:)' TetromCoeffV(:)' TetromCoeffD(:)' C];
end

 C  = [I_t(:)' C];
average_size = size(I)/(2^L);
S(1,:) = average_size;
for i=2:L+1
 S(i,:) = average_size;
 average_size = average_size.*2;
end
S(i+1,:) = size(I);

B = TetromTiling;
```

```
function [meanV varV modeV maxV ...
    minV absIV absI2V] = Get4x4BlockStat(I);
%
% Collect statistics of block I
% statistics: mean, variance, mode, median, max, min,
%           : sum(abs(I)), sum(abs(Iˆ2))

I = I(:);
meanV   = mean(I);
varV    = std(I);
modeV   = mode(I);
maxV    = max(I);
minV    = min(I);
absIV   = sum(abs(I));
absI2V  = sum(abs(I.ˆ2));
```

```
function f = MatlabCoeffInImageFormat(C,L,DoScale);

% Convert the one dimensional array of wavelet coefficient
% from wavedec2 command  to image format (2D).
% C,L are outputs of wavedec2 matlab command.
% DoScale can be set to 1 to scale coefficients to cover
% entire range (0 to 255).

if nargin < 3
 DoScale = 0;
end
```

```
f = [];

% Average coefficients
mn = L(1,:); m=mn(1); n=mn(2);

start = 0;
for i = 1:m
  f = [f;C(start+1:start+n)];
  start = start+ n;
end

if (DoScale)
 f = scale(f);
end

f = f';

% Detail coefficients
MaxDecLevels = length(L)-2;

for level = 2:MaxDecLevels+1
    mn = L(level,:); m=mn(1); n=mn(2);

    % Horizontal
    H = [];
    for i = 1:m
        H = [H;C(start+1:start+n)];
        start = start+n;
    end

```

```
    if (DoScale)
      H = scale(H);
    end

    H = H';

    % Vertical
    V = [];
    for i = 1:m
        V = [V;C(start+1:start+n)];
        start = start+n;
    end

    if (DoScale)
      V = scale(V);
    end

    V = V';

    % Diagonal
    D = [];
    for i = 1:m
        D = [D;C(start+1:start+n)];
        start = start+m;
    end

    if (DoScale)
      D = scale(D);
    end

    D = D';
```

```

    % TBD: We are dropping the pixels at the end.
    newf = f(1:n,1:m);
    f = [newf,H;V,D];
    clear newf;
end
```

```
function abserr = abserr(x,y)
%
% Absolute error - compute the absolute error in db.
%       abserr(x,y) = 10*log10((sum(x(:)-y(:))^2));
%
%   e = abserr(x,y);
%
%   Copyright (c) 2009 Manish K. Singh

abserr = 10*log10((sum(x(:)-y(:))^2));
```

```
function calculate_error = calculate_error(x,y,s)

% Calculate error - compute the error based.
%       Error can be either of the followings:
%       s = 'a', absolute error = 10*log10((sum(x(:)-y(:))^2));
%       s = 'm', MSE error      = mean( (x(:)-y(:)).^2 );
%       s = 'p', PSNR error     = max/mse (PSNR)
%       s = 's', SNR error      = 10*log10(s^2/n^2)
%
%   e = calculate_error(x,y,s); where s is either a, m or p.
```

```
%
%   Copyright (c) 2009 Manish K. Singh

if (strcmp(s,'a'))
   calculate_error = abserr(x,y);
elseif (strcmp(s,'m'))
   calculate_error = mse(x,y);
elseif (strcmp(s,'p'))
   calculate_error = psnr(x,y); %% Function from PyreToolbox
elseif (strcmp(s,'s'))
   calculate_error = SNR(x,y);   %% Function from Wavelab
else
   error(['option s = ',s, 'is not supported. Possible', ...
           'options are p, m, s, or a']);
end
```

```
function collect_image_statistics=collect_image_statistics(im_hat,im)
%
% Collect image statistics
% At present, It only collects errors.
% Returned value is a list with following enteries
% [<abs.error> <mse> <psnr> <snr>]

collect_image_statistics = [];

collect_image_statistics = [collect_image_statistics,...
                 calculate_error(im,im_hat,'a')];
collect_image_statistics = [collect_image_statistics,...
                             calculate_error(im,im_hat,'m')];
```

```
collect_image_statistics = [collect_image_statistics,...
                             calculate_error(im,im_hat,'p')];
collect_image_statistics = [collect_image_statistics,...
                 calculate_error(im,im_hat,'s')];
```

```
function YW = get_window_pixels(Y,m,n,i,j,l)
%
% Get all the pixels around a pixel(i,j) in a window.
% Where window size = 2*l+1
% m is number of rows, n is number of columns.
% Y is all the image, boundary extended by l pixels on
% each side.
%
%   Copyright (c) 2009 Manish K. Singh
%

YW = [];
m_ = m+2*l;
n_ = n+2*l;
i_ = i+l;
j_ = j+l;

for y = [-l:1:l]
% r = [];
 for x = [-l:l:l]
    i__ = i_+y;
    j__ = j_+x;
    index = (i__-1)*n_+(j__);
    YW = [YW,Y(index)];
```

```
 end
% YW = [YW;r];
end
```

```
function [f] = invtetrom2(C,S,B)
%
% Inverse tetrom transform
% C = tetrom coefficients, B = tiling info
% S = house keeping matrix for C (same format as wavedec2)

% Arrange C in 2 D image format
% L is number of decompositions
L = length(S)-2;
t = S(L+2,:); m=t(1); n=t(2);
C_2D = zeros(m,n);

% average coefficients
a = m/2^(L-1)
b = n/2^(L-1)
coeff_ptr = 1;
t = S(1,:); coeff_m = t(1); coeff_n=t(2);
t = zeros(coeff_m,coeff_n);
t(:) = C(coeff_ptr:coeff_ptr+coeff_m*coeff_n-1);
coeff_ptr = coeff_ptr + coeff_m*coeff_n;
C_2D(1:a/2,1:b/2) = t;

for i=1:L
  t = S(i+1,:); coeff_m=t(1); coeff_n=t(2);
  % horizontal
```

```
  t = zeros(coeff_m,coeff_n);
  t(:) = C(coeff_ptr:coeff_ptr+coeff_m*coeff_n-1);
  coeff_ptr = coeff_ptr + coeff_m*coeff_n;
  C_2D(1:a/2,b/2+1:b) = t;
  % Vertical
  t = zeros(coeff_m,coeff_n);
  t(:) = C(coeff_ptr:coeff_ptr+coeff_m*coeff_n-1);
  coeff_ptr = coeff_ptr + coeff_m*coeff_n;
  C_2D(a/2+1:a,1:b/2) = t;
  % Diagonal
  t = zeros(coeff_m,coeff_n);
  t(:) = C(coeff_ptr:coeff_ptr+coeff_m*coeff_n-1);
  coeff_ptr = coeff_ptr + coeff_m*coeff_n;
  C_2D(a/2+1:a,b/2+1:b) = t;
  a = a*2;
  b = b*2;
end

clear C;
C = C_2D;

[m n] = size(C);
f = zeros(m,n);
i = 1;

% start from highest level
a = m/2^(L-1);
b = n/2^(L-1);
f = C;

ws=4;
```

```
for dec=1:L
  ridx = 1;
  cidx = 1;
  t    = zeros(a,b);

  TetromCoeffA = f(1:a/2,1:b/2);
  TetromCoeffH = f(a/2+1:a,1:b/2);
  TetromCoeffV = f(1:a/2,b/2+1:b);
  TetromCoeffD = f(a/2+1:a,b/2+1:b);

  for r=1:ws:a
    for c=1:ws:b
        I4x4          = zeros(4);
        I4x4(1:2,1:2) = TetromCoeffA(ridx:ridx+1,cidx:cidx+1);
        I4x4(3:4,1:2) = TetromCoeffH(ridx:ridx+1,cidx:cidx+1);
        I4x4(1:2,3:4) = TetromCoeffV(ridx:ridx+1,cidx:cidx+1);
        I4x4(3:4,3:4) = TetromCoeffD(ridx:ridx+1,cidx:cidx+1);
        t(r:r+ws-1,c:c+ws-1) = InvTetroletXform4x4(I4x4,B(i));
        i = i+1;
        cidx=cidx+2;
    end
    ridx=ridx+2;
    cidx = 1;
  end
  % update Tetrom Coefficients
  f(1:a,1:b) = t;
  a = a*2;
  b = b*2;
end
```

```
function method_noise = method_noise(I, I_hat, plottitle,  noplot)
% Do the noise analysis given original and noisy image.
% Usage:  f = method_noise(I,In,options)
%
% I   = clean image
% In  = noisy image
% plottitle = 'title for the plot'
% noplot    = default 0, if set will not produce noise plot.
%
%   Copyright (c) 2009 Manish K. Singh
%
%

plottitle

if nargin < 4
  noplot = 0;
end

diff = abs(I_hat - I - 255);

%% Scale the range so that it fills 0 to 255.
%% min: max -> x*255/max
%%

[n1 n2] = size(diff);

diff  = scale(diff);
% 1,n2: structure is visible to lesser extent.

```

```
%if (¬noplot)
  figure
  subplot(111); image(diff(256:512,1:255));
  axis image; axis off; colormap gray(256);
  title([plottitle]);
%  switch noplot
%
%  case 1,
%    title('Lena residue; Visu soft method');
%    print('-deps','lena_residue_visusoft.eps')
%
%  case 2,
%    title('Lena residue; Visu hard method');
%    print('-deps','lena_residue_visuhart.eps')
%
%  case 3,
%    title('Lena residue; sure method');
%    print('-deps','lena_residue_sure.eps')
%
%  case 4,
%    title('Lena residue; Bayes method');
%    print('-deps','lena_residue_bayes.eps')
%
%  case 5,
%    title('Lena residue; michak1 method');
%    print('-deps','lena_residue_michak1.eps')
%
%  case 6,
%    title('Lena residue; michak2 method');
%    print('-deps','lena_residue_michak2.eps')
```

```
%
%  case 7,
%    title('Lena residue; BLS-GSM method');
%    print('-deps','lena_residue_blsgsm.eps')
%
%  case 8,
%    title('Lena residue; Tetrom method');
%    print('-deps','lena_residue_tetrom.eps')
%
%  case 9,
%    title('Lena residue; Redundant Haar method');
%    print('-deps','lena_residue_redun.eps')
%
%    end
%
end
```

```
function mse = mse(x,y)

% mse - compute the mean square error defined as
%       MSE(x,y) = mean((x(:)-y(:)).^2);
%
%   m = mse(x,y);
%
%   Copyright (c) 2009 Manish K. Singh

[a1 b1] = size(x);
[a2 b2] = size(y);

```

```
a = max(a1,a2);
b = max(b1,b2);

mse = (1/(a*b))*sum( (x(:)-y(:)).^2 );
```

```
function [f,p] = plot_fft(s);

% Calculate the power vs frequency of signal s.
% signal is assumed to be result of fft function.

n = length(s);
p = abs(s(1:floor(n/2))).^2
nyquist = 1/2;
f = (1:n/2)/(n/2)*nyquist
```

```
function scale = scale(I,a,b, MaximumValue)
% Scale the image locally so that we can view the hidden details
% Scale the block to full range

[n1 n2] = size(I);
if (nargin < 2)
  a = n1;
  b = n2;
end

if (nargin < 4)
  MaximumValue = 255;
end
```

```

%% Scale it to 0 to max.
for i = 1:a:n1
  for j = 1:b:n2
    p = I(i:i+a-1,j:j+b-1);
    minValue = min(p(:));
    maxValue = max(p(:));
    I(i:i+a-1,j:j+b-1) = ...
      ceil(MaximumValue*(p-minValue)/(maxValue-minValue));
  end
end

scale = I;
```

```
function [BestCoeff BestTile] = ...
  GetBestTetromCoeff(I, ...
             options, ...
             Iclean,  ...
             PrintStatistics,...
             FidStat)
%
% Get best tetrom coefficients.
% Returns Best coefficients [A W0;W1 W2] and BestTile.
% Where
%  A = 4 average coefficients.
%  W0, W1, W2 are detailed coefficients
% options are
%  method = criteria to select based ...
%           (possible values are L1, L2, T1)
```

```
%           (default is L1)
%  T       = threshold for T1 method
%  MaxC    = limit the number of tiling.


BestCoeff = zeros(4);
BestTile  = 1;
options.null = 0;
MaxC         = 117;
End          = 117;
method       = 'L1';

if nargin < 4
 PrintStatistics = 0;
 FidStat   = 0;
end

%% Keep MaxC option for backward compatiblility

if isfield(options,'MaxC')
 MaxC = options.MaxC;
 End  = options.MaxC;
end

Start = 1;
if isfield(options,'Start')
    Start = options.Start;
end

if isfield(options,'End')
    End = options.End;
```

```
end

T = 10;
if isfield(options,'T')
 T = options.T;
end

if isfield(options,'method')
  method = options.method;
end

% Initialize the BestScore variable
BestScore = -1;
if method == 'p1'
 BestScore = 2^31-1;
end
if method == 's1'
 BestScore = 2^31-1;
end
if method == 'l1'
 BestScore = 2^31-1;
end
if method == 'l2'
 BestScore = 2^31-1;
end


for C = Start:End
  % Take a transform
  XformCoeffs = TetroletXform4x4(I,C);
  if (PrintStatistics)
```

```
    [meanV varV modeV maxV minV absIV ...
       absI2V] = Get4x4BlockStat(XformCoeffs);
    fprintf(FidStat,'%d %d %f %f %f %f %f %f %f\n', ...
        [0 C meanV varV modeV maxV minV absIV absI2V]);
end

% calculate score
if method == 'p1'
  % Threshold detailed coefficients,
  a = XformCoeffs;
  a = a.*(abs(a) > T);
  a(1:2,1:2) = XformCoeffs(1:2,1:2);
  I_hat               = InvTetroletXform4x4(a,C);
  Score               = calculate_error(Iclean,I_hat,'m');
  if (Score < BestScore)
     BestScore = Score;
     BestTile  = C;
     BestCoeff = XformCoeffs;
  end
elseif method == 's1'
  Score       = GetTetromScore(XformCoeffs,options);
  if (Score < BestScore)
     BestScore = Score;
     BestTile  = C;
     BestCoeff = XformCoeffs;
  end
elseif method == 'l1'
  Score       = GetTetromScore(XformCoeffs,options);
  if (Score < BestScore)
     BestScore = Score;
     BestTile  = C;
```

```
        BestCoeff = XformCoeffs;
     end
  elseif method == 'l2'
     Score       = GetTetromScore(XformCoeffs,options);
     if (Score < BestScore)
        BestScore = Score;
        BestTile  = C;
        BestCoeff = XformCoeffs;
     end
  else
     Score       = GetTetromScore(XformCoeffs,options);
     if (C == 1)
      BestScore = Score;
      BestTile  = C;
      BestCoeff = XformCoeffs;
     end
     if (Score > BestScore)
        BestScore = Score;
        BestTile  = C;
        BestCoeff = XformCoeffs;
     end
  end
  % remember the best one
end
```

```
function f = GetBestTetromLabelling(I);
% Get best order that minimizes distance
% from respective Haar Partition. See reference:
% Jens Krommweh, Department of Mathematics,
```

```
% University of Duisburg-Essen, Germany ``Tetrolet Transform:
% A New Adaptive Haar Wavelet Algorithm for Sparse Image
% Representation''

Bestscore = 16;
HaarLabel = [0 0 2 2; 0 0 2 2; 1 1 3 3; 1 1 3 3];

C(1,:) = [1 2 3 4];
C(2,:) = [1 2 4 3];
C(3,:) = [1 3 2 4];
C(4,:) = [1 3 4 2];
C(5,:) = [1 4 2 3];
C(6,:) = [1 4 3 2];

C(7,:) = [2 1 3 4];
C(8,:) = [2 1 4 3];
C(9,:) = [2 3 1 4];
C(10,:) = [2 3 4 1];
C(11,:) = [2 4 1 3];
C(12,:) = [2 4 3 1];

C(13,:) = [3 1 2 4];
C(14,:) = [3 1 4 2];
C(15,:) = [3 2 1 4];
C(16,:) = [3 2 4 1];
C(17,:) = [3 4 1 2];
C(18,:) = [3 4 2 1];

C(19,:) = [4 1 2 3];
C(20,:) = [4 1 3 2];
C(21,:) = [4 2 1 3];
```

```
C(22,:) = [4 2 3 1];
C(23,:) = [4 3 1 2];
C(24,:) = [4 3 2 1];


for count=1:24
   temp1 = C(count,1);
   temp2 = C(count,2);
   temp3 = C(count,3);
   temp4 = C(count,4);
   T = [I(temp1,:); I(temp2,:); I(temp3,:); I(temp4,:)];
   A = zeros(4);
   A(T(1,:)) = 0;
   A(T(2,:)) = 1;
   A(T(3,:)) = 2;
   A(T(4,:)) = 3;
   P = A - HaarLabel;
   score = sum(P(:) ≠ 0);
   if (score < Bestscore)
     Bestscore = score;
     f         = T;
   end
end
```

```
function f = GetTetromPermMatrix4x4(Index)
%
% There are 417 ways to fill a 4x4 square with tetrominoes shapes.
% These configurations are indexed using 1 to 417. Given any index
% this function will return 4x4 matrix. Each row of which specifies
```

```
% the respective pixel positions in 4x4 block.

% Positions are numbered as follows:
% 1 5 9 13
% 2 6 10 14
% 3 7 11 15
% 4 8 12 16

M =        [1 2 5 6; 9 10 13 14; 3 4 7 8; 11 12 15 16];

M(:,:,2) = [1 5 9 13; 3 7 11 15; 2 6 10 14; 4 8 12 16];
M(:,:,3) = [1 2 3 4; 5 6 7 8; 9 10 11 12; 13 14 15 16];

M(:,:,4) = [1 5 9 13; 2 3 6 7; 10 11 14 15; 4 8 12 16];
M(:,:,5) = [1 2 3 4; 7 8 11 12; 5 6 9 10; 13 14 15 16];

M(:,:,6) = [1 5 6 9; 2 3 4 7; 10 13 14 15; 8 11 12 16];
M(:,:,7) = [1 2 3 6; 4 7 8 12; 5 9 10 13; 11 14 15 16];

M(:,:,8) = [1 2 3 7; 4 8 11 12; 5 6 9 13; 10 14 15 16];
M(:,:,9) = [1 5 9 10; 2 3 4 6; 11 13 14 15; 7 8 12 16];

M(:,:,10) = [1 5 9 13; 3 4 7 8; 2 6 10 14; 11 12 15 16];
M(:,:,11) = [1 2 5 6; 3 4 7 8; 9 10 11 12; 13 14 15 16];
M(:,:,12) = [1 2 5 6; 3 7 11 15; 9 10 13 14; 4 8 12 16];
M(:,:,13) = [1 2 3 4; 5 6 7 8; 9 10 13 14; 11 12 15 16];

M(:,:,14) = [1 5 9 13; 2 3 7 11; 6 10 14 15; 4 8 12 16];
M(:,:,15) = [5 6 7 9; 1 2 3 4; 13 14 15 16; 8 10 11 12];
M(:,:,16) = [2 3 6 10; 4 8 12 16; 1 5 9 13; 7 11 14 15];
M(:,:,17) = [1 2 3 4; 6 7 8 12; 5 9 10 11; 13 14 15 16];
```

```
M(:,:,18) = [1 5 9 13; 2 3 4 6; 10 14 15 16; 7 8 11 12];
M(:,:,19) = [1 5 9 10; 2 3 6 7; 13 14 15 16; 4 8 11 12];
M(:,:,20) = [1 2 3 7; 4 8 12 16; 5 6 9 10; 11 13 14 15];
M(:,:,21) = [1 2 3 4; 7 8 12 16; 5 6 9 13; 10 11 14 15];

M(:,:,22) = [1 5 9 13; 2 3 4 8; 6 7 10 11; 12 14 15 16];
M(:,:,23) = [1 2 5 9; 3 4 8 12; 6 7 10 11; 13 14 15 16];
M(:,:,24) = [1 2 3 5; 6 7 10 11; 9 13 14 15; 4 8 12 16];
M(:,:,25) = [1 2 3 4; 6 7 10 11; 5 9 13 14; 8 12 15 16];

M(:,:,26) = [1 2 6 10; 3 4 8 12; 5 9 13 14; 7 11 15 16];
M(:,:,27) = [1 2 3 5; 4 6 7 8; 9 10 11 13; 12 14 15 16];
M(:,:,28) = [1 2 5 9; 3 4 7 11; 6 10 13 14; 8 12 15 16];
M(:,:,29) = [1 5 6 7; 2 3 4 8; 9 13 14 15; 10 11 12 16];

M(:,:,30) = [1 2 6 10; 3 4 7 11; 5 9 13 14; 8 12 15 16];
M(:,:,31) = [1 5 6 7; 2 3 4 8; 9 10 11 13; 12 14 15 16];
M(:,:,32) = [1 2 5 9; 3 4 8 12; 6 10 13 14; 7 11 15 16];
M(:,:,33) = [ 1 2 3 5; 4 6 7 8; 9 13 14 15; 10 11 12 16];

M(:,:,34) = [ 1 5 6 10; 2 3 4 8; 9 13 14 15; 7 11 12 16];
M(:,:,35) = [ 1 2 5 9; 3 4 6 7; 10 11 13 14; 8 12 15 16];
M(:,:,36) = [ 1 2 3 5; 4 7 8 11; 6 9 10 13; 12 14 15 16];
M(:,:,37) = [ 1 2 6 7; 3 4 8 12; 5 9 13 14; 10 11 15 16];

M(:,:,38) = [ 1 2 5 6; 3 4 8 12; 9 10 13 14; 7 11 15 16];
M(:,:,39) = [ 1 2 3 5; 4 6 7 8; 9 10 13 14; 11 12 15 16];
M(:,:,40) = [ 1 2 6 10; 3 4 7 8; 5 9 13 14; 11 12 15 16];
M(:,:,41) = [ 1 2 5 6; 3 4 7 8; 9 10 11 13; 12 14 15 16];
M(:,:,42) = [ 1 2 5 9; 3 4 7 8; 6 10 13 14; 11 12 15 16];
```

```
M(:,:,43) = [ 1 5 6 7; 2 3 4 8; 9 10 13 14; 11 12 15 16];
M(:,:,44) = [ 1 2 5 6; 3 4 7 11; 9 10 13 14; 8 12 15 16];
M(:,:,45) = [ 1 2 5 6; 3 4 7 8; 9 13 14 15; 10 11 12 16];

M(:,:,46) = [ 1 5 9 13; 3 4 8 12; 2 6 10 14; 7 11 15 16];
M(:,:,47) = [ 1 2 3 5; 4 6 7 8; 9 10 11 12; 13 14 15 16];
M(:,:,48) = [ 1 2 6 10; 3 7 11 15; 5 9 13 14; 4 8 12 16];
M(:,:,49) = [ 1 2 3 4; 5 6 7 8; 9 10 11 13; 12 14 15 16];
M(:,:,50) = [ 1 2 5 9; 3 7 11 15; 6 10 13 14; 4 8 12 16];
M(:,:,51) = [ 1 5 6 7; 2 3 4 8; 9 10 11 12; 13 14 15 16];
M(:,:,52) = [ 1 5 9 13; 3 4 7 11; 2 6 10 14; 8 12 15 16];
M(:,:,53) = [ 1 2 3 4; 5 6 7 8; 9 13 14 15; 10 11 12 16];

M(:,:,54) = [ 1 5 9 13; 2 3 4 6; 7 10 11 14; 8 12 15 16];
M(:,:,55) = [ 1 5 9 10; 2 3 4 8; 13 14 15 16; 6 7 11 12];
M(:,:,56) = [ 1 2 5 9; 3 6 7 10; 11 13 14 15; 4 8 12 16];
M(:,:,57) = [ 5 6 10 11; 1 2 3 4; 9 13 14 15; 7 8 12 16];
M(:,:,58) = [ 1 2 3 7; 4 8 12 16; 5 9 13 14; 6 10 11 15];
M(:,:,59) = [ 1 2 3 5; 4 8 11 12; 6 7 9 10; 13 14 15 16];
M(:,:,60) = [ 2 6 7 11; 3 4 8 12; 1 5 9 13; 10 14 15 16];
M(:,:,61) = [ 1 2 3 4; 7 8 10 11; 5 6 9 13; 12 14 15 16];

M(:,:,62) = [ 2 3 4 6; 7 8 10 11; 1 5 9 13; 12 14 15 16];
M(:,:,63) = [ 2 6 7 11; 3 4 8 12; 1 5 9 10; 13 14 15 16];
M(:,:,64) = [ 1 2 3 5; 4 8 12 16; 6 7 9 10; 11 13 14 15];
M(:,:,65) = [ 1 2 3 4; 7 8 12 16; 5 9 13 14; 6 10 11 15];
M(:,:,66) = [ 5 6 10 11; 1 2 3 7; 9 13 14 15; 4 8 12 16];
M(:,:,67) = [ 1 2 5 9; 3 6 7 10; 13 14 15 16; 4 8 11 12];
M(:,:,68) = [ 1 5 9 13; 2 3 4 8; 10 14 15 16; 6 7 11 12];
M(:,:,69) = [ 5 6 9 13; 1 2 3 4; 7 10 11 14; 8 12 15 16];

```

```
M(:,:,70) = [ 2 3 6 10; 4 7 8 11; 1 5 9 13; 12 14 15 16];
M(:,:,71) = [ 1 2 6 7; 3 4 8 12; 5 9 10 11; 13 14 15 16];
M(:,:,72) = [ 1 2 3 5; 4 8 12 16; 6 9 10 13; 7 11 14 15];
M(:,:,73) = [ 1 2 3 4; 6 7 8 12; 5 9 13 14; 10 11 15 16];
M(:,:,74) = [ 1 5 6 10; 2 3 7 11; 9 13 14 15; 4 8 12 16];
M(:,:,75) = [ 1 2 5 9; 3 4 6 7; 13 14 15 16; 8 10 11 12];
M(:,:,76) = [ 1 5 9 13; 2 3 4 8; 6 10 14 15; 7 11 12 16];
M(:,:,77) = [ 5 6 7 9; 1 2 3 4; 10 11 13 14; 8 12 15 16];

M(:,:,78) = [ 1 5 9 13; 2 3 4 6; 10 11 14 15; 7 8 12 16];
M(:,:,79) = [ 1 5 9 10; 2 3 4 6; 13 14 15 16; 7 8 11 12];
M(:,:,80) = [ 1 5 9 10; 2 3 6 7; 11 13 14 15; 4 8 12 16];
M(:,:,81) = [ 1 2 3 4; 7 8 12 16; 5 6 9 10; 11 13 14 15];
M(:,:,82) = [ 1 2 3 7; 4 8 12 16; 5 6 9 13; 10 11 14 15];
M(:,:,83) = [ 1 2 3 7; 4 8 11 12; 5 6 9 10; 13 14 15 16];
M(:,:,84) = [ 1 5 9 13; 2 3 6 7; 10 14 15 16; 4 8 11 12];
M(:,:,85) = [ 1 2 3 4; 7 8 11 12; 5 6 9 13; 10 14 15 16];

M(:,:,86) = [ 1 5 9 13; 2 3 4 8; 6 7 10 14; 11 12 15 16];
M(:,:,87) = [ 1 2 5 9; 3 4 7 8; 13 14 15 16; 6 10 11 12];
M(:,:,88) = [ 1 2 5 6; 3 7 10 11; 9 13 14 15; 4 8 12 16];
M(:,:,89) = [ 5 6 7 11; 1 2 3 4; 9 10 13 14; 8 12 15 16];
M(:,:,90) = [ 1 2 3 5; 4 8 12 16; 9 10 13 14; 6 7 11 15];
M(:,:,91) = [ 1 2 5 6; 3 4 8 12; 7 9 10 11; 13 14 15 16];
M(:,:,92) = [ 2 6 10 11; 3 4 7 8; 1 5 9 13; 12 14 15 16];
M(:,:,93) = [ 1 2 3 4; 6 7 8 10; 5 9 13 14; 11 12 15 16];

M(:,:,94) = [ 1 5 9 13; 2 3 4 7; 6 10 14 15; 8 11 12 16];
M(:,:,95) = [ 1 5 6 9; 2 3 4 7; 13 14 15 16; 8 10 11 12];
M(:,:,96) = [ 1 5 6 9; 2 3 7 11; 10 13 14 15; 4 8 12 16];
M(:,:,97) = [ 5 6 7 9; 1 2 3 4; 10 13 14 15; 8 11 12 16];
```

```
M(:,:,98) = [ 1 2 3 6; 4 8 12 16; 5 9 10 13; 7 11 14 15];
M(:,:,99) = [ 1 2 3 6; 4 7 8 12; 5 9 10 11; 13 14 15 16];
M(:,:,100) = [ 2 3 6 10; 4 7 8 12; 1 5 9 13; 11 14 15 16];
M(:,:,101) = [ 1 2 3 4; 6 7 8 12; 5 9 10 13; 11 14 15 16];

M(:,:,102) = [ 1 5 9 13; 2 3 4 7; 6 10 11 14; 8 12 15 16];
M(:,:,103) = [ 1 5 6 9; 2 3 4 8; 13 14 15 16; 7 10 11 12];
M(:,:,104) = [ 1 2 5 9; 3 6 7 11; 10 13 14 15; 4 8 12 16];
M(:,:,105) = [ 5 6 7 10; 1 2 3 4; 9 13 14 15; 8 11 12 16];
M(:,:,106) = [ 1 2 3 6; 4 8 12 16; 5 9 13 14; 7 10 11 15];
M(:,:,107) = [ 1 2 3 5; 4 7 8 12; 6 9 10 11; 13 14 15 16];
M(:,:,108) = [ 2 6 7 10; 3 4 8 12; 1 5 9 13; 11 14 15 16];
M(:,:,109) = [ 1 2 3 4; 6 7 8 11; 5 9 10 13; 12 14 15 16];

M(:,:,110) = [ 1 5 6 10; 2 3 4 7; 9 13 14 15; 8 11 12 16];
M(:,:,111) = [ 1 5 6 9; 2 3 4 7; 10 11 13 14; 8 12 15 16];
M(:,:,112) = [ 1 5 6 9; 2 3 4 8; 10 13 14 15; 7 11 12 16];
M(:,:,113) = [ 1 2 5 9; 3 4 6 7; 10 13 14 15; 8 11 12 16];
M(:,:,114) = [ 1 2 3 6; 4 7 8 11; 5 9 10 13; 12 14 15 16];
M(:,:,115) = [ 1 2 3 6; 4 7 8 12; 5 9 13 14; 10 11 15 16];
M(:,:,116) = [ 1 2 3 5; 4 7 8 12; 6 9 10 13; 11 14 15 16];
M(:,:,117) = [ 1 2 6 7; 3 4 8 12; 5 9 10 13; 11 14 15 16];

f = M(:,:,Index);
```

```
function f = GetTetromScore(I,options)
%
% Calculate a score for given coefficients in I.
% options.sigma -> variance of noise
```

```
% options.method -> method used in score calculation.
% options.coeff_var -> used in method c1 in core calculation.
% options.coeff_abs -> used in method c1 in core calculation.
% options.coeff_max -> used in method c1 in core calculation.
% methods are: (score reperesents ..)
%   'l1' -> Sum of absolute values of detailed coefficients
%   'l2' -> Energy in detailed coefficients
%   't1' -> Number of detailed coefficients greater than given
%           threshold (T)
%   't2' -> Zero out detailed coefficients less than T, and then
%           sum energy in the coefficients
%   's1' -> Standard Deviation of I
%   'c1' -> score = var*coeff_var + abs(I)*coeff_abs +
%           max(abs(I))*coeff_max,
%           where var_c + var_i + var_m = 1
% I = coefficients

options.null = 0;

if isfield(options, 'method')
  method = options.method;
else
  method = 'T1';
end

if isfield(options, 'T')
  T = options.T;
else
  T = 10;
end

```

```
if isfield(options,'sigma')
 sigma = options.sigma;
else
  sigma = 15;
end

if isfield(options,'coeff_var')
  coeff_var = options.coeff_var;
else
  coeff_var = 1;
end

if isfield(options,'coeff_abs')
  coeff_abs = options.coeff_abs;
else
  coeff_abs = 0;
end

if isfield(options,'coeff_max')
  coeff_max = options.coeff_max;
else
  coeff_max = 0;
end

%% Ignore average coefficients from
a = I;
a(1:2,1:2) = zeros(2);


% L1 score
switch lower(method)
```

```
 case 'l1'
 %% minimum sum of detailed coefficients
 %% Same as in proposed paper
   f = sum(abs(a(:)));

 case 'l2'
     %% Minimum Energy in detail coefficients
   a = a(:);
   f = sum(a.^2);

 case 't1'
 %% More number of large coefficients
 a = a(:);
 a = abs(a) > T;
 f = sum(a);

 case 't2'
 %% 0 weight to detailed coefficient smaller than threshold.
 %% I^2 -> larger weight to large coefficients
 a = I;
 I = I.*(abs(I) ≥ T);
 I(1:2,1:2) = a(1:2,1:2);
 I = I(:);
 f = sum(I.^2);

case 's1'
% Minimum standard deviations
f = std(I(:));

case 'c1'
% score = var*coeff_var + abs(I)*coeff_abs + max(abs(I))*coeff_max
```

```
 % where = var_c + var_i + var_m = 1
 I = I(:);
 f = var(I)*coeff_var + sum(abs(I))*coeff_abs + max(abs(I))*coeff_max;

  otherwise
      error(['Unknown option method = ',method]);

end
```

B.2 Code used to Generate the Thesis Figures

```
% Generate figure 1
% Load an image
I   = load_image('boat');
n   = length(I);

% Add noise
sigma = 40;
Noise =  sigma*randn(n);
In    = I + Noise;

% Plot the image and noisy image
figure
subplot(111); image(I); axis square; axis off;
title('Clean boat image'); colormap gray(256);
print('-deps','CleanBoat.eps');
figure
subplot(111); image(In); axis square; axis off;
title('noisy boat image'); colormap gray(256);
print('-deps','NoisyBoat.eps');
```

```
%% Generate and plot histogram for boat, lena,
%% barb and mandrill images.

for index = 1:4
  if (index == 1)
     name = 'boat';
```

```
elseif (index == 2)
   name = 'lena';
elseif (index == 3)
   name = 'mandrill';
else
   name = 'barbara';
end

% Load an image
L    = 2     ; % Number of decomposition levels
M    = load_image(name);
MW   = perform_wavelet_transform(M,L,1);

% Extract detail coefficients and plot their histogram
LH1  = MW(end/2+1:end, 1:end/2);
HL1  = MW(1:end/2    , end/2+1:end);
HH1  = MW(end/2+1:end, end/2+1:end);

% Quantize all coefficients to finite precision
T = 3; % bins for histogram
[tmp,LH1q] = perform_quantization(LH1,T);
[tmp,HL1q] = perform_quantization(HL1,T);
[tmp,HH1q] = perform_quantization(HH1,T);

% Generate histogram
[LH1h,LH1x] = compute_histogram(LH1q);
[HL1h,HL1x] = compute_histogram(HL1q);
[HH1h,HH1x] = compute_histogram(HH1q);

% Plot the results
figure
```

```
    subplot(221); image(M);
    axis image; axis off; title(['Image ',name]);
    subplot(222); plot(LH1x,LH1h);
    title(['LH1 coefficients histogram']);
    xlabel('Coefficient Value'); ylabel('Normalized frequency');
    subplot(223); plot(HL1x,HL1h);
    title(['HL1 coefficients histogram']);
    xlabel('Coefficient Value'); ylabel('Normalized frequency');
    subplot(224); plot(HH1x,HH1h);
    title(['HH1 coefficients histogram']);
    xlabel('Coefficient Value'); ylabel('Normalized frequency');
    colormap gray(256);
    fname = strcat('histogram',name);
    print('-deps',fname);
end
```

```
%% Plot sinwave and Db10 wavelet

s              = sin(20.*linspace(0,pi,1000));
[phi, psi, x] = wavefun('db10', 5);

subplot(121); plot(s)  ; title('Sine wave');
subplot(122); plot(psi); title('Db10 Wavelet');
print('-deps','SineVsDb10Wave');
```

```
%% Generate 1D Denoising example for Thesis report.
%% Uses load_signal function from PyreToolBox.

```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Some global variables that control how this program is run.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
n           = 1024;    %% length of signal
DecLevels   = 6;
waveletname = 'db4';
err_type    = 'm';    %% a -> abs, m -> mse, p -> psnr
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Load piece wise regular signal
randn('state',1001);
y     = load_signal('piece-regular',n);   %% Clean signal
sigma = 0.06 * (max(y)-min(y));           %% Noise level
yn    = y + sigma*randn(n,1);             %% Noisy signal
errA  = calculate_error(y,yn,'a');        %% Quantify the error
errM  = calculate_error(y,yn,'m');        %% Quantify the error
errP  = calculate_error(y,yn,'p');        %% Quantify the error

% Plot the clean and noisy signal
figure('Name','1-D Denoising Examples');
plot(y); title('Original clean signal');
print('-deps','Fig3_1_CleanSignal');
axis tight;
figure
plot(yn);
title(['Noisy signal with abs. err. =  ',num2str(errA),...
       ' mse = ',num2str(errM),' psnr = ',num2str(errP)]);
axis tight;
print('-deps','Fig3_1_NoisySignal');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Running average and plot it's result.
% Sharp edges will be smoothed out.
% Iterate and find out best window to lower MSE.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
windowrange = [1:2:15];
e_list      = [];

for w = windowrange
 y_hat  = filter(ones(1,w)/w,1,yn);
 err    = calculate_error(y,y_hat,err_type);
 e_list = [e_list,err];
end

% Plot the error vs window
figure
plot(windowrange,e_list);
title('Error vs averaging window');
xlabel('Window size');
ylabel('MSE in db');
print('-deps','Fig3_1_MSEvsWindowSize');

%% Calculate & plot the optimally filtered result
[tmp,i] = min(e_list); w = windowrange(i);
y_hat = filter(ones(1,w)/w,1,yn);
errA  = calculate_error(y,y_hat,'a');     %% Quantify the error
errM  = calculate_error(y,y_hat,'m');     %% Quantify the error
errP  = calculate_error(y,y_hat,'p');     %% Quantify the error
figure
plot(y_hat);
title(['Denoised signal with running average of window ',...
        num2str(w), 'abs. err. =  ',num2str(errA),...
```

```
        'mse = ',num2str(errM),' psnr = ',num2str(errP)]);
axis tight;
print('-deps','Fig3_1_DenoisedAverage');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Wiener filtering
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ff    = fft(y); ffn = fft(yn);
pf    = abs(ff).^2;  % spectral power
hwf   = pf./(pf+n*sigma^2);
y_hat = real(ifft(ffn.* hwf));
errA  = calculate_error(y,y_hat,'a');
errM  = calculate_error(y,y_hat,'m');
errP  = calculate_error(y,y_hat,'p');
figure
plot(y_hat);
title(['Denoised signal with wiener filtering, and' ...
        'abs. err. =  ',num2str(errA),' mse = ',...
        num2str(errM),' psnr = ',num2str(errP)]); axis tight;
print('-deps','Fig3_1_DenoisedWeiner');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Wavelet transform and hard threshold denoising
% Try different thresholds and pick the best.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[C,L]    = wavedec(yn,DecLevels,waveletname);

% Iterate over different thresholds
t_list = linspace(0,15,30);
e_list = [];

```

```
for t = t_list
  T      = t*sigma;
  cD     = C((L(1)+1):end);
  CT     = [C(1:L(1));HardThresh(cD,T)];
  y_hat  = waverec(CT,L,waveletname);
  err    = calculate_error(y,y_hat,err_type);
  e_list = [e_list,err];
end

% Now calculate/plot the best denoised version of signal
[tmp,i] = min(e_list); T = t_list(i)*sigma;
figure
plot(t_list,e_list); title('Error vs T/sigma');
xlabel('Threshold in units of sigma');
ylabel('MSE in db');
print('-deps','Fig3_1_MSEvsThreshold');

cD     = C((L(1)+1):end);
CT     = [C(1:L(1));HardThresh(cD,T)];
y_hat  = waverec(CT,L,waveletname);
errA   = calculate_error(y,y_hat,'a');   %% Quantify the error
errM   = calculate_error(y,y_hat,'m');   %% Quantify the error
errP   = calculate_error(y,y_hat,'p');   %% Quantify the error

figure
plot(y_hat);
title(['Denoised signal with ',waveletname, ...
       'wavelet (L=',num2str(DecLevels), ') abs. err. =  ',...
        num2str(errA),' mse = ',num2str(errM),' psnr = ',...
        num2str(errP)]); axis tight;

```

```
print('-deps','Fig3_1_DenoisedWavelet');
```

```
%% Generate 1D Denoising example for Thesis report.
%% Uses load_signal function from PyreToolBox.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Some global variables that control how this program is run.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
n            = 1024;
DecLevels    = 6;
Wavelets     = char('db1','db2','db3','db4','db9','sym2',...
                 'sym3','sym4','sym8','coif1','coif4','coif5');
err_type     = 'm';    %% m -> mse error, p -> psnr error,
                       %% a -> absolute error
                       %% This is used to find optimal threshold
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Load piece wise regular signal
randn('state',1001);
y      = load_signal('piece-regular',n); %% Clean signal
sigma = 0.06 * (max(y)-min(y));          %% Noise level
yn     = y + sigma*randn(n,1);           %% Noisy signal
err    = calculate_error(y,yn,err_type); %% Quantify the error

% Plot the clean and noisy signal
figure('Name','1-D Denoising Examples');
plot(y); title('Original clean signal');
axis tight;
print('-deps','Fig3_2_CleanSignal');
```

```
figure
plot(yn);
title(['Noisy signal with err. =  ',num2str(err)]);
axis tight;
print('-deps','Fig3_2_NoisySignal');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Wavelet transform and hard threshold denoising
% Try different thresholds and pick the best.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i = 1:length(Wavelets)
 waveletname = strtrim(Wavelets(i,1:end))
 [C,L]    = wavedec(yn,DecLevels,waveletname);

 % Iterate over different thresholds
 t_list = linspace(0,15,30);
 e_list = [];
 cD     = C((L(1)+1):end);

 for t = t_list
   T      = t*sigma;
   CT     = [C(1:L(1));HardThresh(cD,T)];
   y_hat  = waverec(CT,L,waveletname);
   err    = calculate_error(y,y_hat,err_type);
   e_list = [e_list,err];
 end

 % Now calculate/plot the best denoised version of signal
 [tmp,j] = min(e_list); T = t_list(j)*sigma;

 CT     = [C(1:L(1));HardThresh(cD,T)];
```

```
 y_hat  = waverec(CT,L,waveletname);
 err    = calculate_error(y,y_hat,err_type);

 figure
 plot(y_hat);
 title(['Denoised signal with ',waveletname, ...
           'wavelet, MSE = ',num2str(err)]);
fname = strcat('Fig_3_2_DenoisedSignal_',waveletname);
 axis tight;
print('-deps',fname);

end
```

```
%% Generate 2D Denoising example for Thesis report.
%% Uses load_image functions from PyreToolBox.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Some global variables that control how this program is run.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DecLevels   = 2; %
waveletname = 'db10';
err_type    = 'm';    %% a -> abs, m -> mse, p -> psnr
name        = 'lena'; %% picture name
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Load the image
I   = load_image(name);
n   = length(I);

```

```
randn('state',1001); % to have repeatability in result

% Add noise
sigma = 30;
In    = I + sigma*randn(n);

% Calculate error
errP  = calculate_error(I,In,'p'); %% Quantify the error

figure
subplot(111); image(I); axis image; axis off;
title('Original Image'); colormap gray(256);
print -deps Denoising2DExample_1.eps
figure
subplot(111); image(In); axis image; axis off;
title(['Noisy Image, psnr = ',num2str(errP), 'db']);
colormap gray(256);
print -deps Denoising2DExample_2.eps

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Running average and plot it's result
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
windowrange = [1:2:15];
e_list      = [];

for w = windowrange
 I_hat  = filter2(ones(1,w)/w,In);
 err    = calculate_error(I,I_hat,err_type);
 e_list = [e_list,err];
end

```

```
% Plot the error vs window
figure
subplot(111); plot(windowrange,e_list);
title('Error vs averaging window'); axis square;
print -deps Denoising2DExample_3.eps

% Plot the optimal filtered image
[tmp,i] = min(e_list); w = windowrange(i);
I_hat = filter2(ones(1,w)/w,In);
errP  = calculate_error(I,I_hat,'p'); %% Quantify the error
figure
subplot(111); image(I_hat); axis image; axis off;
title(['Running average of window ',num2str(w), ...
       ' psnr = ',num2str(errP), ' db']);
colormap gray(256);
print -deps Denoising2DExample_4.eps

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Wiener2 filter
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fI = fft2(I); fIn = fft2(In);
pf = abs(fI).^2; % spectral power
% fourier transform of the wiener filter
hwf = pf./(pf+ n^2*sigma^2);
% perform the filtering over the fourier
I_hat = real( ifft2(fIn .* hwf) );
errP  = calculate_error(I,I_hat,'p');  %% Quantify the error
figure
subplot(111); image(I_hat); axis image; axis off;
title(['Wiener2 psnr = ',num2str(errP), ' db']);
colormap gray(256);
```

```
print -deps Denoising2DExample_5.eps

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Wavelet transform and hard threshold denoising
% Try different thresholds and pick the best. (OracleShrink)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[C,L]    = wavedec2(In,DecLevels,waveletname);

% Iterate over different thresholds
t_list = linspace(0,5,10);
e_list = [];
index  = L(1,:); m = index(1); n = index(2);
cD     = C((m*n+1):end);

for t = t_list
  T      = t*sigma;
  CT     = [C(1:m*n),SoftThresh(cD,T)];
  I_hat  = waverec2(CT,L,waveletname);
  err    = calculate_error(I,I_hat,err_type);
  e_list = [e_list,err];
end

% Now calculate/plot the best denoised version of signal
[tmp,i] = min(e_list); T = t_list(i)*sigma;
figure
subplot(111); plot(t_list,e_list); title('Error vs T/sigma');
axis square; colormap gray(256);
print -deps Denoising2DExample_6.eps

CT     = [C(1:m*n),SoftThresh(cD,T)];
I_hat  = waverec2(CT,L,waveletname);
```

```
errP   = calculate_error(I,I_hat,'p'); %% Quantify the error

figure
subplot(111); image(I_hat); title([waveletname, ...
 'wavelet (L=',num2str(DecLevels), ') psnr = ',num2str(errP), ' db']);
axis image; colormap gray(256); axis off;
print -deps Denoising2DExample_7.eps

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Wavelet transform, using bayes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure
str.repres1     = 'fs';
str.repres2     = '';
str.parent      = 1;
str.Nor         = 8;
str.Nsc         = 2;
options(1).name = 'blsgsm';
options(1).params = str;
f = denoise_image(In, options,sigma, 'p',1,I,name,0);
print -deps Denoising2DExample_8.eps
```

```
%% Generate 2D Denoising example for Thesis report.
%% Uses load_signal,load_image function from PyreToolBox.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Some global variables that control how this program is run.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DecLevels   = 2;
```

```
Wavelets    = char('db1','db4','db9', 'db13', 'sym2',...
                   'sym4','sym8','coif1','coif4','coif5',...
                   'bior4.4','dmey');
err_type    = 'm';   %% m -> mse error, p -> psnr error,
                     %% a -> absolute error
                     %% This is used to find optimal threshold
name        = 'lena'; %% picture image
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Load the image
I   = load_image(name);
n   = length(I);

randn('state',1001); % to have repeatable results

% Add noise
sigma = 0.12*(max(I(:))-min(I(:)));
In    = I + sigma*randn(n);

% Calculate error
err  = calculate_error(I,In,'p');  %% Quantify the error
figure
image(I); axis image; axis off;
title('Original Image'); colormap gray(256);
print('-deps','Fig3_4_LenaCleanImage');
figure
image(In); axis image;
title(['Noisy Image, psnr = ',num2str(err), ' db']);
colormap gray(256);
print('-deps','Fig3_4_LenaNoisyImage');

```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Wavelet transform and hard threshold denoising
% Try different thresholds and pick the best.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure
figcnt = 1;
epsfilecnt = 1;

for i = 1:length(Wavelets)
 waveletname = strtrim(Wavelets(i,1:end))
 [C,L]    = wavedec2(In,DecLevels,waveletname);

 % Iterate over different thresholds
 t_list = linspace(0,6,12);
 e_list = [];
 index  = L(1,:); m = index(1); n = index(2);
 cD     = C((m*n+1):end);

 for t = t_list
   T      = t*sigma;
   CT     = [C(1:m*n),HardThresh(cD,T)];
   I_hat  = waverec2(CT,L,waveletname);
   err    = calculate_error(I,I_hat,err_type);
   e_list = [e_list,err];
 end

 % Now calculate/plot the best denoised version of signal
 [tmp,j] = min(e_list); T = t_list(j)*sigma;

 CT     = [C(1:m*n),HardThresh(cD,T)];
 I_hat  = waverec2(CT,L,waveletname);
```

```
  err    = calculate_error(I,I_hat,'p');

  subplot(1,1,figcnt); image(I_hat);
  title([waveletname, ' psnr  = ',num2str(err), ' db']);
  axis image; colormap gray(256); axis off;
  if (figcnt == 1)
    fname = strcat('Denoising2DEffectOfBasis_',num2str(epsfilecnt));
    print('-deps', fname);
    figure
    figcnt = 1;
    epsfilecnt = epsfilecnt + 1;
  else
    figcnt = figcnt + 1;
  end

end
```

```
% Load an image
name    = char('lena','barbara','boat','house');
%sigma    = [10 15 20 25 30];
%name      = char('lena');
sigma    = 0;
errtype = 'p';
n = 128;

%% tetrom parameters
options.method = 'l1';
options.L      = 1;
MaxC = 117;
```

```
randn('state',0);

[NumImages,temp] = size(name);
for i = 1:NumImages
  iname = strtrim(name(i,1:end));
  I = load_image(iname,n);
  % Add noise
  for sig = sigma
     In = I + sig*randn(n);
     options.sigma   = sig;
     options.T      = sqrt(2*log(n*n))*sig;
     disp(['Threshold is ',num2str(options.T)]);

    % call the denoise function (tetrom)
    [f c_tetrom] = perform_tetrom_denoising(In,options,I);
    % call the denoise function (haar)
    options.MaxC = 1;
    [fhaar c_haar] = perform_tetrom_denoising(In,options,I);
    options.MaxC = MaxC;

    % compute histogram and plot
    LL1_t  = c_tetrom(1:end/2, 1:end/2);
    LH1_t  = c_tetrom(end/2+1:end, 1:end/2);
    HL1_t  = c_tetrom(1:end/2    , end/2+1:end);
    HH1_t  = c_tetrom(end/2+1:end, end/2+1:end);

    LL1_h  = c_haar(1:end/2, 1:end/2);
    LH1_h  = c_haar(end/2+1:end, 1:end/2);
    HL1_h  = c_haar(1:end/2    , end/2+1:end);
    HH1_h  = c_haar(end/2+1:end, end/2+1:end);

```

```
% Quantize all coefficients to finite precision
T = 3; % bins for histogram
[tmp,LL1q_t] = perform_quantization(LL1_t,T);
[tmp,LH1q_t] = perform_quantization(LH1_t,T);
[tmp,HL1q_t] = perform_quantization(HL1_t,T);
[tmp,HH1q_t] = perform_quantization(HH1_t,T);

[tmp,LL1q_h] = perform_quantization(LL1_h,T);
[tmp,LH1q_h] = perform_quantization(LH1_h,T);
[tmp,HL1q_h] = perform_quantization(HL1_h,T);
[tmp,HH1q_h] = perform_quantization(HH1_h,T);

% Generate histogram
[LL1h_t,LL1x_t] = compute_histogram(LL1q_t);
[LH1h_t,LH1x_t] = compute_histogram(LH1q_t);
[HL1h_t,HL1x_t] = compute_histogram(HL1q_t);
[HH1h_t,HH1x_t] = compute_histogram(HH1q_t);

[LL1h_h,LL1x_h] = compute_histogram(LL1q_h);
[LH1h_h,LH1x_h] = compute_histogram(LH1q_h);
[HL1h_h,HL1x_h] = compute_histogram(HL1q_h);
[HH1h_h,HH1x_h] = compute_histogram(HH1q_h);

% Plot the results
figure
subplot(221); image(In);  colormap gray(256);

subplot(222); plot(LH1x_t,LH1h_t,'r');
title(['LH1 coefficients histogram']);
hold on
subplot(222); plot(LH1x_h,LH1h_h);
```

```
    title(['LH1 coefficients histogram']);
    hold off

    subplot(223); plot(HL1x_t,HL1h_t,'r');
    title(['HL1 coefficients histogram']);
    hold on
    subplot(223); plot(HL1x_h,HL1h_h);
    title(['HL1 coefficients histogram']);
    hold off

    subplot(224); plot(HH1x_t,HH1h_t,'r');
    title(['HH1 coefficients histogram']);
    hold on
    subplot(224); plot(HH1x_h,HH1h_h);
    title(['HH1 coefficients histogram']);
    hold off

  end
end
```

```
% Tetrolet vs Haar Denoising performance
% Plot tetrolet transform PSNR vs number of tetrom partitions
% averaged.

name  = char('lena','barbara','boat','house');
sigma = [5 10 15 20 25 30];
%sigma = 10;
errtype  = 'p';
plot_img = 1;
```

```
n        = 128;
randn('state',1001); %% Set randomization to have repeatable answer.
NumberOfIterations = 117;

%% Tetrolet options
options.method = 'T1';
options.L = 1;
options.PrintStatistics = 0;
options.PrintStatFname = '';

[NumImages,temp] = size(name);
for sig = sigma
  figure
  hold on
  options.sigma = sig;
  T0 = sqrt(2*log(n*n))*sig*0.68;
  options.T = T0;
  k = 0;
  for i = 1:NumImages
    iname = strtrim(name(i,1:end));
    I = load_image(iname,n);
    Error = [];
    In = I + sig*randn(n);
    i_hat = zeros(n);
    for j=1:NumberOfIterations
      options.Tiling = j;
      % call the denoise function
      [f c_tetrom] = perform_tetrom_denoising(In,options,I);
      i_hat = i_hat + f;
      Error = [Error; calculate_error(I,i_hat./j,errtype)];
    end
```

```
    switch k
      case 0, plot(1:NumberOfIterations,Error,'ko:');
      case 1, plot(1:NumberOfIterations,Error,'kx:');
      case 2, plot(1:NumberOfIterations,Error,'k+:');
      case 3, plot(1:NumberOfIterations,Error,'k*:');
      case 4, plot(1:NumberOfIterations,Error,'ks:');
      case 5, plot(1:NumberOfIterations,Error,'kd:');
    end
    k = k + 1;
  end
  legend(name);
  hold off;
end
```

```
% Plot tetrom denoising performance vs threshold

name  = char('lena','barbara','boat','house');
sigma = [10 20];
%sigma = 10;
errtype = 'p';
dna     = 0;
n   = 128;
MaxC = 117;
decl = 1;
opt.L               = decl;
opt.PrintStatistics = 0;
opt.PrintStatFname  = 'none';
randn('state',1001);

```

```
[NumImages,temp] = size(name);

for sig = sigma
  figure
  hold on
  T0 = sqrt(2*log(n*n))*sig;
  thres_list = [1/8:1/8:3/2];
  opt.sigma  = sig;

  for i = 1:NumImages
     iname = strtrim(name(i,1:end));
     I    = load_image(iname,n);
     % Add noise
     In     = I + sig*randn(n);

     Error = [];

     for thres = thres_list
       opt.T = thres*T0;

       %% Now do tetrom based denoising
       i_hat_sum = zeros(n);
       for j=1:117
         opt.Tiling = j;
         % call the denoise function (tetrom)
         [f c_tetrom] = perform_tetrom_denoising(In,opt,I);
         i_hat_sum = i_hat_sum+f;
       end
       im_hat = i_hat_sum./j;
       clear i_hat_sum;
       Error = [Error, calculate_error(I,im_hat,errtype)];
```

```
      end
      switch i
        case 1, plot(thres_list,Error,'ko:');
        case 2, plot(thres_list,Error,'kx:');
        case 3, plot(thres_list,Error,'k+:');
        case 4, plot(thres_list,Error,'k*:');
      end
     xlabel('Threshold (T/T0) where T0 is universal threshold');
     ylabel('Psnr in db');
   end
   legend(name);
   title(['Tetrolet performance vs threshold with sigma = ',...
           num2str(sig), ' T0 = ',num2str(T0)]);
end
```

```
% Program to generate performance table.

%% Load images
name  = char('boat','house');
%name = char('barbara');
sigma = [10 15 20 25 30];
%sigma = [10];
NumIterations = 10;
seed = 1001;
n = 128;

%% Information about what algorithms we are using
str.wnam = 'db1';
str.decl = 1;
```

```
options(1).name   = 'visu';
str.type          = 'hard';
options(1).params = str;

options(2).name   = 'visu';
str.type          = 'soft';
options(2).params = str;

options(3).name   = 'sure';
options(3).params = str;

options(4).name   = 'bayes';
options(4).params = str;

options(5).name   = 'michak1';
options(5).params = str;

options(6).name   = 'michak2';
options(6).params = str;

options(7).name   = 'blsgsm';
options(7).params = 'null';

options(8).name   = 'tetrom';
options(8).params = str;

options(9).name   = 'redun';
str.wnam = 'haar';
options(9).params = str;

```

```
algonames = char(options(1).name,options(2).name,...
         options(3).name,options(4).name,...
         options(5).name,options(6).name,...
         options(7).name,options(8).name,...
         options(9).name);

result = [];
f0 = zeros(NumIterations,4);
f1 = zeros(NumIterations,4);
f2 = zeros(NumIterations,4);
f3 = zeros(NumIterations,4);
f4 = zeros(NumIterations,4);
f5 = zeros(NumIterations,4);
f6 = zeros(NumIterations,4);
f7 = zeros(NumIterations,4);
f8 = zeros(NumIterations,4);
f9 = zeros(NumIterations,4);


[NumImages,temp] = size(name);
randn('state',seed);

for i = 1:NumImages
  iname = strtrim(name(i,1:end));
  I = load_image(iname,n);
  for sig = sigma
    display([iname, ' ( sigma = ',num2str(sig), ...
        ')    ABS      MSE     PSNR    SNR']);
    for itr = 1:NumIterations
      In = I + sig*randn(n);
      f = denoise_image(In, options, sig, 'p', 0, I, iname, 0);
```

```
    if itr == 1
      result = f;
    else
      result = result + f;
    end
    f0(itr,:) = f(1,:);
    f1(itr,:) = f(2,:);
    f2(itr,:) = f(3,:);
    f3(itr,:) = f(4,:);
    f4(itr,:) = f(5,:);
    f5(itr,:) = f(6,:);
    f6(itr,:) = f(7,:);
    f7(itr,:) = f(8,:);
    f8(itr,:) = f(9,:);
  end
  % calculate standard deviation and print
  err    = zeros(9,4);
  err(1,:) = std(f0);
  err(2,:) = std(f1);
  err(3,:) = std(f2);
  err(4,:) = std(f3);
  err(5,:) = std(f4);
  err(6,:) = std(f5);
  err(7,:) = std(f6);
  err(8,:) = std(f7);
  err(9,:) = std(f8);
  % calculate average and print
  result = result./NumIterations;
  result = [algonames,num2str(result)];
  display(result);
  display(err);
```

```
    display(f0);
    display(f1);
    display(f2);
    display(f3);
    display(f4);
    display(f5);
    display(f6);
    display(f7);
    display(f8);

    figure
    hold on
    x = [1:NumIterations];
    plot(x,f0(:,3),'bo:');
    plot(x,f1(:,3),'gx:');
    plot(x,f2(:,3),'kd:');
    plot(x,f3(:,3),'c*:');
    plot(x,f4(:,3),'ms:');
    plot(x,f5(:,3),'yd:');
    plot(x,f7(:,3),'r+:');
    xlabel('Iterations'); ylabel('psnr in db');
    title([iname,' Image']);
    legend('VisuHard', 'VisuSoft', 'Sure', ...
       'Bayes', 'Michak1', 'Michak2', 'Tetrom');

    figure
    hold on
    plot(x,f6(:,3),'bo:');
    plot(x,f8(:,3),'gx:');
    plot(x,f7(:,3),'r+:');
    xlabel('Iterations'); ylabel('psnr in db');
```

```
    title([iname,' Image']);
    legend('BLS-GSM', 'Redundant', 'Tetrom');

  end

end
```

Appendix C

Acronyms

ADC	Analog to Digital converter
AWGN	Additive White Gaussian Noise
CWT	Complex Wavelet Transform
DT-CWT	Dual Tree Complex Wavelet Transform
DWT	Discrete Wavelet Transform
GGD	Generalized Gaussian Distribution
GSM	Gaussian Scale Mixture
HH	High High (output of high pass followed by high pass filter)
HL	High Low (output of high pass followed by low pass filter)
IDWT	Inverse Discrete Wavelet Transform
LCD	Liquid Crystal Display
LH	Low High (output of low pass followed by high pass filter)
LL	Low Low (output of low pass followed by low pass filter)
MAP	Maximum A Posteriori Probability
ML	Maximum Likelihood
MMSE	Minimum Mean Square Error
MRA	Multiresolution Analysis
MSE	Mean Square Error
PSNR	Peak Signal to Noise Ratio
QMF	Quadrature Mirror Filters
SNR	Signal to Noise Ratio
SURE	Stein's Unbiased Risk Estimate

TIWT	Translation Invariant Wavelet Transform