UNIVERSITY OF NAIROBI

SCHOOL OF ENGINEERING

WAVELET BASED DENOISING METHODS FOR MAGNETIC RESONANCE IMAGES

KAGOIYA KENNETH GICHOHI

REG NO F56/ 8645/ 2005

A Research Thesis Submitted in Partial Fulfillment of the Requirements for the award of Degree of Master of Science in Electrical and Electronic Engineering of University of Nairobi.

JULY 2019

DECLARATION

I declare that this thesis is my own original work except where explicitly stated otherwise in the text, and that this work has not been submitted for the award of any degree or professional qualification in any other institution.

Signature **……...……..** **Date ……...…….**

KENNETH KAGOIYA

CERTIFICATION

I certify that this thesis is work was carried out under my supervision and this thesis has been submitted for examination with my approval as the University supervisor.

Signature **……...……..** **Date ……...…….**

PROF. ELIJAH MWANGI

DEDICATION

This work is dedicated to my parents Peris and Dickson Kagoiya.

ACKNOWLEDGEMENT

A lot of thanks go to my project supervisor Prof Elijah Mwangi who led me during my research on the thesis. His ideas, advice and guidance have helped me a lot.

I would also like to thank Prof Ouma for his encouragement and support. I would also like to thank all lecturing staff of Electrical and Information Engineering department who took time to teach me as a lone student during my course work.

I would also like to thank the Technical University of Mombasa for the financial support and study leave while doing my coursework and project work at KTH (Royal Institution of Technology). In addition I would like to thank Mr Victor Kyalo and Prof Bjorn Pehrson for supporting my application and admission to the senior Masters Communication Design Project at KTH(Stockholm, Sweeden)

ABSTRACT

Magnetic resonance medical imaging is one of the diagnostic methods used today in cardiograph, mammography and brain tumor analysis as well as many other applications. They are acquired using gradient coil when magnetic moments in body tissues resonate with a very high magnetic field of the Magnetic Resonance Imaging (MRI) scanner. To obtain the image, the raw data acquired is inverse Fourier transformed which results in a complex image which has Rician distributed noise. Most conventional noise removal methods are not suited for MRI denoising since they do not take in consideration the Rician nature of this noise .Various wavelet based methods that have modeled Rician noise have been developed but with short comings .The objective of this research is to develop an efficient and effective solution for denoising a Magnetic Resonance Image. A detailed analysis of MRI formation is presented and then a mathematical model is formulated for an image that is corrupted by Rician noise. Using statistical data of the images, a number of wavelet based filter algorithms have been developed to denoise the images. The bilateral filter in its adaptive form is used to enhance image features and edges to minimize smoothing effects of noise filtering process. Other processes include signal and noise estimation using various methods including Chi square unbiased risk estimator (CURE), Poisson unbiased Risk Estimator with Linear Estimation of Thresholds (PURE-LET),Steinbeck unbiased Risk Estimator with Linear Estimation of Thresholds (SURE-LET), Linear Minimum Mean square error (LMMSE) and also parameter estimation for various filters used as building blocks in different combination filters. The experimental investigation involves structural MRI, functional MRI and Diffusion weighted MRI images of brain, torso, cranium, hip and knee as test images. Four combinational denoising methods have been developed, these are the wavelet based Haar denoising method using non-local means filter and bilateral enhancement, A wavelet based MRI denoising method using LMMSE estimation and bilateral filter enhancement. Others are a total variational wavelet based structural MRI denoising with bilateral feature enhancement and a Haar wavelet magnetic resonance image denoising with optimized chi-square Rician estimation and bilateral filter enhancement. MATLAB simulation platform has been used in testing their effectiveness against Rician noise. Results obtained show that most of the new methods perform better than the stand alone filters and wavelet thresholding in all forms. This is for both subjective comparison and using various objective measures of quality such as the Peak Signal to noise Ratio (PSNR) and Structural Similarlity Index Measure (SSIM). For example in Table 5.1 and Table 5.2 where *SNR* is very high, *MSE* relatively low, UQI almost 1, SSIM 0.984 and EPI 0.89 which is an improvement from 0.70 of the noisy image. It also shows that edge preservation is very sensitive even for low noise.

ACRONYMS AND ABBREVIATIONS

4 PDE	Fourth order Partial Differential Equation
ADF	Anisotropic Diffusion Filter
BF	Bilateral Filter
BOLD	Blood Oxygenation Level Dependent
CALIPIRINHA	Controlled Aliasing in Parallel Imaging Results in Higher Acceleration
CNR	Contrast to Noise Ratio
CT	Computed Tomography
CURE	Chi square unbiased Risk estimator
DFT	Discrete Fourier Transform
DT-MRI	Diffusion Tensor MRI
DWT	Discrete Wavelet Transform
EPI	Echo planar imaging
FID	Free induction decay
FMRI	Functional Magnetic Resonance Imaging
FT	Fourier Transform
HHH	High High High
HHL	High High Low
HLH	High Low High
HLL	High Low Low
LHH	Low High High
LHL	Low High Low
LLH	Low Low High
LLL	Low Low Low
LMMSE	Linear Minimum Mean Square Error
LMSE	Laplacian Mean Square Error
MAP	Maximum A-Posterior
MD	Maximum Difference
M_O	Magnetization at Equilibrium
MRBF	Multi Resolution Bilateral Filter

MRI	Magnetic Resonance Imaging
MSE	Mean Square Error
M_T	Trasverse Magnetization
M_X	Vertical Magnetization
M_Y	Horizontal Magnetization
NCCS	Non Centric Chi-Square
NLM	Non-local Means
PDF	Probability Density Function
PET	Positron Emission Tomography
PSNR	Peak Signal to Noise Ratio
PURE	Poisson Unbiased Risk Estimator
RMSE	Root Mean Square Error Sense
SENSE	Sensitivity encoding method
SM	Similarity Measure for NLM
SNR	Signal to Noise Ratio
SPACE RIP	Sensitivity Profiles From an Array of Coils for Encoding and Reconstruction in Parallel.
SPECT	Single Positron Emission Computed Tomography
SURE	Steinbeck unbiased Risk Estimator
T_1	Longitudinal relaxation time
T_2	Exponential decay time constant
T_2^{inh}	Inhomogeinity sensitive time constant
T_2^*	Total time constant
T_E	Echo time
TV	Total Variation
US	Ultra Sound
WT	Wavelet Transform

TABLE OF CONTENTS

DECLARATION ..... **ii**
DEDICATION ..... **iii**
ACKNOWLEDGEMENT ..... **iv**
ABSTRACT ..... **v**
ACRONYMS AND ABBREVIATIONS ..... **vi**
LIST OF FIGURES ..... **xiii**
LIST OF TABLES ..... **xv**

CHAPTER ONE: INTRODUCTION **1**
1.1 Background 1
1.2 Acquisition and Nature of MRI 2
1.3 Problem Statement 3
1.4 Objectives 3
1.5 Scope of Work 4
1.6 Organization of Thesis 4
1.7 Note on Publication 4

CHAPTER TWO: LITERATURE REVIEW **5**
2.1 Merits of Wavelet Transforms 5
2.2 The Discrete Wavelet Transform 5
2.3 Wavelet Transforms in Two Dimensions 6
2.4 Related Works 6
2.5 Knowledge Gaps 10

CHAPTER THREE: MAGNETIC RESONANCE IMAGE THEORY **11**
3.0 Introduction 11
3.1 Nuclear Magnetic Resonance 11
3.1.1 Nuclei Behaviour 11
3.1.2 Energy Levels 12
3.1.3 Excitation 13

3.1.4 Net Magnetization 14
3.1.5 Longitudinal Relaxation (T_1) 14
3.1.6 Oscillation 15
3.1.7 Frame of Reference 15
3.1.8 T_2Relaxation 16
3.1.9 Excitation by a Radio-Frequency Pulse 16
3.1.10 RF-pulse 16
3.2 Nuclear Magnetic Resonance Signal 18
3.2.1 Free Induction Decay 18
3.2.2 Quadrature Reception of NMR signal 19
3.2.3 Spin Echo Process 19
3.3 Magnetic Resonance Imaging Processes 21
3.3.1 Gradient Magnetic Fields 22
3.3.2 Characterizing the Magnetic Vector 22
3.3.3 Slice Selection 23
3.3.4 Fourier Imaging 24
3.3.5 Frequency Encoding 24
3.3.6 Phase Encoding 24
3.3.7 Spin Echo Imaging Sequence 25
3.3.8 Alternative Imaging Sequences 28
3.3.9 Signal Representation 28
3.3.10 Field of View 31
3.3.11 Image Contrast settings 31
3.3.12 Coil Array 32
3.4 Aliasing 32
3.4.1 Parallel MRI and Aliasing 34
3.4.2 Reconstruction 34
3.4.3 Estimation 35
3.4.4 Coil Sensitivities 35
3.5 Sensitivity Encoding Method 39
3.5.1 Cartesian SENSE 40

3.6 SPACE RIP 42
3.7 Partially Parallel Imaging with Localized Sensitivities 43
3.8 Parallel Magnetic Resonance Imaging with Adaptive Radius in *k*-space 43
3.9 Pre-Enhancement 43
3.9.1 Handling Motion Artifacts 43
3.9.2 CAIPIRINHA 44
3.10 Diffusion Magnetic Resonance Imaging 44
3.10.1 The Einstein Approach 44
3.10.2 Diffusion Tensor Imaging 46

CHAPTER FOUR: IMAGE DENOISING APPROACHES 50
4.1 Wiener Filter 50
4.1.1 Other Linear Filters 51
4.1.2 Non-linear Filters 51
4.1.3 Bilateral Filter 51
4.1.4 Patch based Approaches 52
4.1.5 Variational Approaches 53
4.1.6 Total Variation Minimization 53
4.2 Transform Domain Approaches 55
4.3 Wavelet-Domain Filtering 57
4.3.1 The Discrete Wavelet Transform 57
4.3.2 Wavelet-Domain Filtering 60
4.3.3 Wavelet based Wiener Filter 62
4.4 Image Signal and Noise Modelling and Estimation 66
4.4.1 Rician LMMSE Estimator 66
4.4.1.1 Conventional Approach 67
4.4.1.2 Maximum Likelihood Estimator 67
4.4.1.3 Expectation-maximization (E M) Method 67
4.4.1.4 The Analytical Exact Solution 68
4.4.1.5 LMMSE Estimator for Rician Model 68

4.5 A CURE: Chi-Square Unbiased Risk Estimation 69
4.5.1 Cure-optimized denoising via Unnormalised HAAR Wavelet Transform 71
4.5.2 Magnitude Image 72

CHAPTER FIVE: MATERIALS, METHODS AND EXPERIMENTAL RESULTS ... 74
5.1 Combination Denoising Schemes 74
5.2 Parameter Selection for the Bilateral Filter 74
5.3 Parameter Selection for the non-local means Filter 75
5.4 Parameter Selection in CURE and CURELET 76
5.5 Conventional Measures of Quality 78
5.6 Experimental Procedure 79
5.7 Method I: A hybrid and adaptive MRI denoising method involving a bilateral filter enhancement and Non-local Means wavelet based method 79
5.8 Proposed Approach for method I 79
5.9 Measures of Quality 81
5.9.1 Signal to Noise Ratio 81
5.9.2 Mean Square Error 81
5.9.3 Correlation Coefficient 81
5.9.4 Edge Preservation Index 81
5.9.5 Universal Quality Index 81
5.9.6 The Structural Similarity Index Measure (SSIM) 82
5.10 Experimental Results 82
5.10.1 Effect of Noise Addition 82
5.10.2 Denoising using Wavelet Thresholding, Median, Bilateral and Non-local methods 84
5.10.3 Denoising using the Proposed Method 84
5.10.4 Residue noise 87
5.11 Result analysis and discussion 87
5.12 Method II: Wavelet Hybrid MRI Denoising Scheme using Chi Square unbiased Risk Estimate with Bilateral Filter Preprocessing and Enhancement 88
5.12.1 Proposed Approach 88

5.12.2 Result Analysis and Discussion 94
5.13 Method III: An LMMSE diffusion weighted MRI Image Denoising Wavelet based Algorithm with bilateral feature Enhancement 94
5.13.1 Proposed Method 94
5.13.2 Selection of Parameters in LMMSE 94
5.13.3 Denoising Algorithm 97
5.14 Method IV: A Total Variational Wavelet based Structural MRI Denoising Method with Bilateral Feature Enhancement 100
5.14.1 Selection of Parameters in Variation Denoising 102
5.14.2 Experimental Results 106
5.14.3Result Analysis and Discussion 107

CHAPTER SIX: CONCLUSION AND RECOMMENDATION FOR FURTHER WORK 108
6.1 Conclusion 108
6.2 Recommendations 108
REFERENCES 110

APPENDICES 117
Appendix 1: Measures of quality 117
Appendix 2: Stand Alone Filters 123
Appendix 3: Chi Square Combination 155
Appendix 4: Combination Non-Local Means Filter 166
Appendix 5: LMMSE Combination 190
Appendix 6: Total Variational Combination 204

LIST OF FIGURES

Figure 3.1: Non Aligned Spins 12
Figure 3.2: Aligned Spins 13
Figure 3.3(a) Rotating Frame of Reference (b) Static Frame of Reference 15
Figure 3.4: Effect of the RF pulse 17
Figure 3.5: Free Induction Decay 18
Figure 3.6: Receiver Coil and Signal 19
Figure 3.7: Spin Echo Sequence 20
Figure 3.8: Spin echo Magnetic Vector 21
Figure 3.9: Slice Selection 23
Figure 3.10: Timing Diagram of Spin Echo Sequence 25
Figure 3.11: Spin Echo Signal Dephasing and Rephrasing 26
Figure 3.12: MRI Signal Formation 27
Figure 3.13: SMASH Estimation and Reconstruction 36
Figure 3.14: Motion Correction on MRI 49
Figure 4.1: HAAR Wavelet Transform 71
Figure 5.1: A Flowchart of the Proposed Method 80
Figure 5.2: Noise-free MRI images (a) *Torso1*, (b) *Torso2*, (c) *Hip* 83
Figure 5.3: Noisy images of *Torsol1* at given noise levels 83
Figure 5.4: Denoised Images of *Torso1* using other Methods 84
Figure 5.5: Denoised Images of *Torsol1* using Combinational Non Local Algorithm 86
Figure 5.6: Residual Noise 87
Figure 5.7: Algorithm for Chi-square Method 90
Figure 5.8: Relatively clean MRI Images 91
Figure 5.9: Noisy Images of *Torso2* at Various Levels 91
Figure 5.10: Residual Noise 91
Figure 5.11: Denoised images of *Torsol2* using Combinational Chi-square algorithm 92
Figure 5.12: Denoised images of *Torsol2* using other METHODS 92
Figure 5.13: Algorithm for LMMSE method 96
Figure 5.14: Relatively clean *Hip* MRI image 97

Figure 5.15: Noisy *Hip* Images at various levels 97
Figure 5.16: Residual noise for *Hip* 98
Figure 5.17: Denoised *Hip* images using combinational LMMSE algorithm 98
Figure 5.18: Denoised *Hip* images using other methods 99
Figure 5.19: Algorithm for the Proposed method (Total variational combination method). 101
Figure 5.20: Relatively clean MRI image(cranium) 102
Figure 5.21: Noisy Cranium Images at various levels 102
Figure 5.22: Residual noise for Cranium image 105
Figure 5.23: Denoised Cranium images new algorithm(Combinational total variational)... 105
Figure 5.24: Denoised Cranium images using other methods 105

LIST OF TABLES

Table 5.1: Quality Measures at 2% noise for combinational Non local means algorithm 85
Table 5.2: Quality Measures at 5% noise for combinational Non local means algorithm 85
Table 5.3: Quality Measures at 10% noise for combinational Non local algorithm..... 86
Table 5.4: Quality Measures at 20% noise for combinational Non local algorithm..... 86
Table 5.5: Quality Measures at 2% for combinational Chi-square algorithm 92
Table 5.6: Quality Measures at 5% for combinational Chi-square algorithm 93
Table 5.7: Quality Measures at 10% noise for combinational Chi-square algorithm..... 93
Table 5.8: Quality Measures at 20% noise for combinational Chi-square algorithm..... 93
Table 5.9: Quality Measures at 2% noise for combinational LMMSE algorithm 99
Table 5.10: Quality Measures at 5% noise for combinational LMMSE algorithm 99
Table 5.11: Quality Measures at 10% noise for combinational LMMSE algorithm 99
Table 5.12: Quality Measures at 20% noise for combinational LMMSE algorithm 100
Table 5.13: Quality Measures at 2% noise for combinational total variational algorithm ... 106
Table 5.14: Quality Measures at 5% noise for combinational total variational algorithm ... 106
Table 5.15: Quality Measures at 10% noise for combinational total variational algorithm . 107
Table 5.16: Quality Measures at 20% noise for combinational total variational algorithm . 107

CHAPTER ONE

INTRODUCTION

1.1 Background

Medical imaging as a diagnosis method in soft tissue monitoring like foetus in ultrasound, cardiology using Magnetic Resonance Imaging, Computed Tomography and also vascular imaging has made tremendous advancement in the last two decades. Many imaging algorithms have been developed and implemented in medical imaging by various manufacturers however these may not be versatile for different types of images and adaptive to various categories of noise thus making medical equipment expensive especially for developing countries.

Magnetic resonance imaging is a medical imaging method of choice due to the fact that it does not use radiations that have the possibility of affecting the patient such as X-rays.MRI image are also superior in terms of visual quality in comparisson to other like ultra sound. MRI based methods have been advanced and together with structural images can also be used for functional images.

The methods of acquisition of MRI use a specific combination of physical and chemical properties of hydrogen protons of water in different body tissues whose behaviour can give functional, diffusion or structural information of a living tissue. The MRI technology has undergone notable improvements in aspects such as speed of acquisition, spatial resolution, signal strength and overall cost. However, there exist challenges in acquired signal in terms of signal to noise ratio and other aspects of quality. Therefore denoising methods to enhance usefulness and performance are required, these when well implemented will improve extraction of organ boundaries and shape, physiological parameter estimation including contrast and tissue perfusion and also pixel based tissue classification.

In order to solve the denoising problem, a set of new adaptive wavelet based medical image denoising methods have been proposed. The emphasis has been MRI in which case noise models have been formulated and using image statistics to estimate parameters and optimize expectations ,denoising algorithms have been developed and their performance tested.

1.2 Acquisition and Nature of MRI

The patient is made to slide through a magnetised chamber. The electromagnetic field lines cut through his body and in return the body tissues create their field. The strength and origin of the signal induced in the coil of the MRI instrument is used to represent the structure and composition of the body tissues. A large permanent magnet produces an magnetic field of upto 6 Tesla.This aligns the nuclear spins present in the body tissue.To transfer a group of spins from alower to a higher energy state electromagnetic energy need lto be supplied at resonance frequency.by transmitting an RF signal at the resonance frequencyenergy is added to the system and when sufficient the magnetic vector is flipped from longitudinal to transverse planewhich in tern generates an electrical signal that contains information from which the image can be generated.

Magnetic resource images are acquired in the Fourier domain and they are referred to as k-space data. Spatial domain representation of the data is produced by applying Inverse Fast Fourier Transform (IFFT) to the k-space data which results in complex image with real and imaginary parts, this also introduces phase error. There is noise that arises due to random fluctuation in the signal acquisition coil, electronic circuits and Brownian motion in patient which also has a real and imaginary part each independent being Gaussian distributed[1] [2].

In [1][3] noise in magnetic resonance magnitude images is shown to obey a Rician distribution. Rician noise is signal-dependent and therefore extracting signal from noise is not an easy task. Rician noise removal is difficult where signal-to-noise ratio (SNR) is not high. Its presence results in random fluctuations and a signal-dependent offset to the data .This reduces image contrast. In magnetic resonance imaging (MRI), there is a relationship between the noise level, image resolution and time required to pick the signal. This therefore results in a SNR threshold. The SNR in many clinical applications of MRI is quite high. Two types of averaging takes place in MRI data acquisition. The discrete nature of the data acquisition process causes spatial volume averaging. In some applications it is common to acquire several measurements at the same location and average them to reduce noise [1].

Rician distribution has the form:

$$f\left(x\right)=\left\{\frac{x}{\sigma^2}\;\; I_0\left(\frac{mx}{\sigma^2}\right)\;\; \exp\left(\frac{-x^2\;+m^2}{2\sigma^2}\right)\right\} \tag{1.1}$$

Where $$I_0(y) = \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{y\cos t} \, dt \quad (1.2)$$

$$x \geq 0;$$

is Bessel's function of the first kind

and $$m^2 = m_i^2 + m_q^2 \quad (1.3)$$

σ is the standard deviation m_i and m_q are the mean values of two independent Gaussian components.

1.3 Problem Statement

The primary task is to develop a set of algorithms that will be able to acquire, decompose, and determine local statistics and correlation characteristics which will be used together with enhanced processing in multi-resolution wavelets coefficients filtering for optimal noise removal of various MRI images. Effective denoising of a real magnetic resonance image is still an issue and the research study has developed adaptive and combination filters which take into account Rician distribution nature of MRI images. This will improve on the performance of existing filters without introducing artifacts and intensity bias.

1.4 Objectives

The main objective is to develop a low cost and effective solution for restoration of degraded magnetic resonance medical images

The thesis work also had the following specific objectives:

i. Investigation into various wavelet based image restoration algorithms and identify those which with modification can be used for Magnetic Resonance images at specific levels of noise.
ii. Develop noise filtering models and algorithms for MRI denoising
iii. Identify and specify the parameters or steps that require enhancement for improved performance.
iv. Application and performance evaluation of the algorithms developed using structural images.

1.5 Scope of Work

The goal is to show that adaptive algorithms achieve acceptable performance in medical MRI image denoising. The objective of the various subroutines is to develop computationally fast, spatially adaptive, wavelet based methods for efficient reduction of Rician noise from the medical images. Special emphasis has been given to application of Haar wavelet in noise suppression of Rician noise in Magnetic Resonance Images (MRI).This is in combination with other Rician noise adapted signal and noise estimation and filtering mechanisms taking into consideration the type of MRI image. For structural MRI bilateral filter enhancement has been used before the wavelet thresholding. For DW-MRI used in neuroimaging an LMMSE estimator is used this is because DW-MRI results in diffusion direction dependent image intensities.

1.6 Organization of Thesis

The rest of the thesis is organized as follows. In chapter two, a literature review of previous work in MRI denoising is carried out. In chapter three detailed description of MRI theory, signal generation and various acquisition methods is given. In chapter four a survey of denosing methods and their merits of MRI are discussed. It also includes mathematical methods of modelling presence of Rician noise in MRI signals.

In chapter five combination wavelet based MRI denoising methods are developed. Also discussed are measures of quality appropriate for MRI denoising effectiveness assessment. Results obtained when the denoising methods are applied to magnetic resonance images with varying degree of noise are tabulated and discussed.

The conclusions of the investigations made in this thesis and recommendation for further work are given in chapter six.

1.7 Note on Publication

The research work in this thesis has also resulted in one paper entitled “A hybrid and adaptive Non-local means Wavelet based MRI Denoising Method with Bilateral Filter Enhancement”. Which has been published in the International Journal of Computer Applications, Vol 166 no 10, 0975-8887 May 2017.

CHAPTER TWO

LITERATURE REVIEW

2.1 Merits of Wavelet Transforms

Although the Fourier Transform has been the main method for image processing, the Discrete Wavelet Transform has been noted to possess superior qualities in image compressing, encoding and enhancement. These properties can be exploited in the processing of medical images such as different types of MRI.

One of the challenges in MRI is the suppression of Rician noise that is acquired during the image formation process. In this chapter, a review of the theoretical aspects of the Discrete Wavelet Transform and its application in noise suppression is provided. A presentation of various methods that have been previously suggested or developed are also given.

2.2 The Discrete Wavelet Transform

The wavelet function results in an ordered pattern of coefficients. If the function at hand is a sequence, the result is called the discrete wavelet transform (DWT) of $f(x,y)$. The DWT transform pair is:

(2.1)

(2.2)

For $j \geq j_o$ and

(2.3)

Where, $f(x)$, $\varphi_{j_o}(x)$ and $\psi_j(x)$ are functions of the discrete variable $x = 0, 1, 2 \ldots, M$. These wavelets coefficient functions represent variations of intensity of images along various directions: ψ^H denotes levels variations along columns, ψ^V denotes levels along rows and ψ^D denotes levels along diagonals.

2.3 Wavelet Transforms in Two Dimensions

Two dimensional transforms may be obtained in a similar manner. In which a scaling function, $\varphi(x, y)$, and the three categories of two-dimensional wavelet functions, $\psi^H(x,y)\ \psi^V(x,y)$ and $\psi^D(x,y)$ are used. These are depicted as follows:

$$\varphi(x,y)=\varphi(x)\varphi(y) \tag{2.4}$$

And "directionally sensitive" wavelets functions:

$$\psi^H(x,y)=\psi(x)\varphi(y) \tag{2.5}$$

$$\psi^V(x,y)=\varphi(x)\psi(y) \tag{2.6}$$

$$\psi^D(x,y)=\psi(x)\psi(y) \tag{2.7}$$

The approximation and wavelet coefficients are given by:

$$w_\varphi(j_0,m,n)=\frac{1}{\sqrt{MN}}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}f(x,y)\varphi_{jo,m,n}(x,y)) \tag{2.8}$$

$$w_\varphi^i(j,m,n)=\frac{1}{\sqrt{MN}}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}f(x,y)\Psi^i{}_{j,m,n}(x,y)) \qquad i=H,V,D \tag{2.9}$$

The inverse wavelet transform in two dimensions is therefore:

$$f(x,y)=\frac{1}{\sqrt{MN}}\sum_m\sum_n W_\varphi(j_0,m,n)\varphi_{jo,m,n}(x,y))$$

$$+\frac{1}{\sqrt{MN}}\sum_{i=H,V,D}\sum_{j=j_0}^{\infty}\sum_m\sum_n W_\varphi^i(j,m,n)\psi^i_{j,m,n}(x,y)) \tag{2.10}$$

2.4 Related Works

The noise in MRI is found to be Rician distributed, however many of the traditional approaches to filters such as mean, median, Wiener do not take into consideration the Rician nature of MRI noise. Although several methods have been suggested in the removal of Rician noise, it still remains a formidable task due to the signal dependency nature especially at low SNR where it reduces image contrast and causes random fluctuation in intensity.

In [1] noise present in MRI was modeled as Rician distributed and also wavelet based filter algorithms for both magnitude and square image signals by application of multi-resolution techniques were developed. A method of removing the bias in the image was also developed. This resulted in improved image denoising and lead to increased interest in both Rician noise models and multi-resolution thresholding.

An application of phase error correction method in MRI denoising has been reported in [2] in which the phase corrected noise has been assumed to have the same distribution as original noise and the real component of the image consists of both signal and noise and imaginary component consisting of just noise. So, discarding the imaginary part and using Gaussian filter would give satisfactory results however there are practical difficulties in estimating this noise due to complexities in evaluation of some of the expressions involved.

Other methods that have used the Rician distribution include [3] in which a linear minimum error estimates using a Rician noise model are developed for the denoising of diffusion weighted images. A recursive LMMSE filter was also developed for 3D images.

In [4] an adaptive multi-resolutions, non-local means filter is developed for 3D MRI images. This involves adaptive wavelet coefficient mixing. The method was shown to effectively remove noise from a degraded image while preserving details of the image.

In [5] a denoising method based on a combination of the total variation minimization and wavelet is developed which involves solution of time evolution partial differential equation by time marching the image using gradient flow resulting in most noise being removed.

In [6] [7] [8] thresholding is used together with bilateral filter, in [6] a multi-resolution bilateral filter together with wavelet transform is presented. Analysis based on residual noise and also using a generic static image model showed improved performance in comparison to stand alone filters such as Wiener, anisotropic, non-local means, total variation, bilateral and wavelet thresholding.

In [9] an analysis of wavelet denoising for MRI brain imaging is done and two MRI filters are discussed. Two filters were developed for MRI denoising, one for noise suppression for magnitude MRI and an additional one on blood oxygen level (BOLD) MRI. The results showed relatively clean image in terms of visual inspection.

In [10] [11] [12] Non-local means filters (NLM) are used. The Non-Local Means filters estimate a pixel in a denoised image as weighted combination of image pixels. The main measure of performance is the similarity measure. In [10] a set of new similarity measure for NLM filtering are described in which statistics of MRI noise are used in image denoising and bias removal.

In [13] a new scheme that applies a series of filters, Kernel and Sobel each used to modifying the noise free image estimate and the final output converging to a stable estimate. A method that uses 2D spatial wavelet filtering enhanced with 1D temporal Karhunen-Loeve Transform (KLT) is developed in which KLT is used to produce a series of Eigen images in this approach the signal information can be obtained with Eigen images and a 2D spatial wavelet filter is applied to each of the individual Eigen images. The denoised Eigen images are transformed back into the image space [14].

In [15] multi-component non-local means filters were developed in which the filtering process was enhanced by using additional information provided by MRI of different types and different acquisition times. In NLM the similarity of pixels is more robust to the noise level since region comparison is used in addition to optimization and principal component analysis (PCS). The method has good results but it is limited to applications in MRI where many images are acquired.

Common objective measures of quality include MSE and SNR. The MSE however is a poor visual quality measure, due to its non-adaptability to local signal specificities (intensity and correlation).New measures of quality have been developed one main one is SSIM described in [12] which combines luminance similarity, contrast similarity and structural similarity and is suited for MRI analysis. It has been shown that the image with the lowest MSE is usually the one with the highest SSIM [12]. Other objective quality assessment methods include

method noise which is the effectiveness of a denoising algorithm. This is determined by analyzing the noise expectation of the method and contrast to noise ratio which shows the strength of a feature of interest in relation to its environment, which may be viewed as the effective spatial resolution.

In [16] an expression for the mean-squared error of a chi-square random variable is developed. The main consideration being the process of reducing noise of image data, as independent non central chi-square random variables on two degrees of freedom. Two categories of linearly parameterized estimators which are optimized using the risk estimate were developed. One in the general context of un-decimated filter bank transforms, and another in the specific case of the un-normalized Haar wavelet transform. The methods were shown to have performed better than stand alone thresholding and Wiener filters.

In [17] a method to minimize a data-adaptive estimate of the mean-squared error (MSE) between the enhanced and the noise-free data which is obtained from "Stein's Unbiased Risk Estimate" (SURE) is developed. The SURE method is taken further and proposal for a fast and efficient multidimensional image denoising made, this is termed the SURE-LET approach. SURE allows the quantitative analysis and the denoising quality and low computational complexity are achieved. Various thresholding functions were applied to input data. Also signal dependent use of the Poisson statistics lead to "Poisson's unbiased risk estimate" (PURE) with more adaptive transform-domain thresholding rules. Application of PURE-LET framework to the Haar wavelet transform was demonstrated.

In [18] Wells used various methods to denoise MRI and remove intensity inhomogeneity effects and went on to show that denoising improved structural MRI to enable detection of Alzheimer disease**.** He used standard quantification model of a range of signal to noise ratio to investigate the bias of a celebral blood flow. Another method set mirrors in the in-vivo protocol and used simulated images of a rat brain which were denoised using independent component analysis.

In [19] Sebastián used image denoising method to reduce errors in APLF (Arterial Spin Labelling Perfusion) and Arterial Transit time and improve the precision of CBF (Cerebral Blood Flow) measurements and precision of transit time maps. He developed a method of

correction of the IIH artifacts by eliminating the IIH field and removing the estimates. There were two methods including parametric estimation and non-parametric using Baysian image filtering and fuzzy clustering.

In [20] a method to denoise multiple-coil acquired MR images was proposed. It took into account both the non-centric chi-square distribution and the spatially varying nature of the noise. Experimental results on both simulated and real data sets demonstrated that the effectiveness of denoising was improved by a combination algorithm.

Advanced non-local means methods were developed in [21][22]. In [21] a proposal was made to improve on timing of the non-local maximum likelihood methods (NLM) and its variants, because though effective in denoising, implementation of the algorithms in real time is a challenge. The main focus was on the parallelization and acceleration of one computationally intensive section of the algorithm so as to demonstrate the execution time improvement through the application of parallel processing concepts on a GPU. Experimental results showed possibility of practical implementation of parallelization.

2.5 Knowledge Gaps

In most of the methods developed in previous works, some noise models were not robust. Therefore an investigation into ways of overcoming unfavourable characteristic of weights used in MRI denoising is required. The effectiveness of denoising on real MRI images is still an issue for methods that appear satisfactory with simulated images. Another challenge is the removal of intensity bias without lowering Signal to Noise ratio especially for square images.

Research in this thesis is therefore an effort to improve the overall effectiveness and robustness of MRI denoising by combining multi-resolution wavelet methods with Bilateral filters, Total Variation and Non-Local Means filters. Other adaptive estimation of image and noise and enhancing techniques have also been explored as means to improve the main denoising process. More suitable effectiveness measures have also been adopted for different aspects of denoised image quality.

CHAPTER THREE

MAGNETIC RESONANCE IMAGE THEORY

3.0 Introduction

The MRI technology has its foundations on the presence of magnetic forces on the atom. The scientific theory and its development is described in this chapter. The basic principles of nuclear magnetic resonance (NMR) spin properties and the way to use them for NMR signal acquisition are discussed. The technique used to acquire images using NMR signal is also described. This chapter also contains details about the imaging sequences and techniques used to accelerate the acquisition process.

3.1 Nuclear Magnetic Resonance

Magnetic resonance uses the fact that if any element with nuclear spin is placed in a magnetic field, it acquires energy from an incident photon at a specific frequency. The excited elements return back to the energy equilibrium after some time and release the absorbed energy as photons. The emitted electromagnetic signal provides information about physical and chemical properties of the excited elements [25, 26].

Magnetic resonance can be also used to generate a signal from hydrogen atoms which are the main constituent of a human body. Therefore, it is possible to make images of human tissue that represent the spatial arrangement of the hydrogen atoms. In this case, the radio-frequency (RF) excitation pulse is not selective. The coils in the instrumentation system also acquire an e.m.f from all other hydrogen atoms. Therefore, all hydrogen atoms in the examined tissue are excited at the same instant and the integral of the signal over the whole volume is measured. The important step from NMR to MRI is to add spatial encoding of the signal. In MRI, it is possible to obtain 2D and 3D images of the tissue.

3.1.1. Nuclei Behaviour

All particles in an atomic nucleus have a spin which is the presence of angular momentum in each individual particle. Spins can be positive or negative and therefore spins of opposite signs cancel out. Only nuclei with an unpaired protons or neutrons contribute to NMR. Such nuclei behave like small magnetic dipoles with a random orientation as shown in Figure 3.1.

The arrows represent spin orientation. The result is that the magnetization of a high number of nuclei cancels out on the average and the net magnetization is zero.

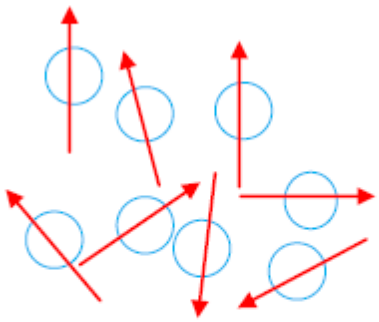


Figure 3.1: Non Aligned Spins

In MRI the primary focus is on the hydrogen atoms. Human tissues are known to mainly consist of water and fat that are primarily hydrogen compounds. It has been estimated that in the human body 63% of atoms are hydrogen [25].It is for this reason that the presentation in this thesis is based on hydrogen nuclei *1H*.

3.1.2 Energy Levels

The nuclei behaviour is explored using quantum mechanics principles. When particles are placed in a magnetic field B_0 (oriented in the z-direction) their spin orientation will tend to align with this magnetic field. Some spins will align parallel and others anti-parallel with respect B_0. The anti-parallel orientation has more energy than parallel orientation. The energy ΔE between the states is:

$$\Delta E = \frac{h\gamma}{2\pi}|B_0||J| \tag{3.1}$$

where γ is a gyro magnetic ratio depending on properties of the atom (for hydrogen ^1H have $\gamma/2\pi$ =42.58 MHz/T), h = 6.6 x10^{-34} J is Planks' constant , $|B_0|$ is intensity of the external magnetic field and J current density). When an external magnetic field is applied the spins are aligned as shown in Figure 3.2.

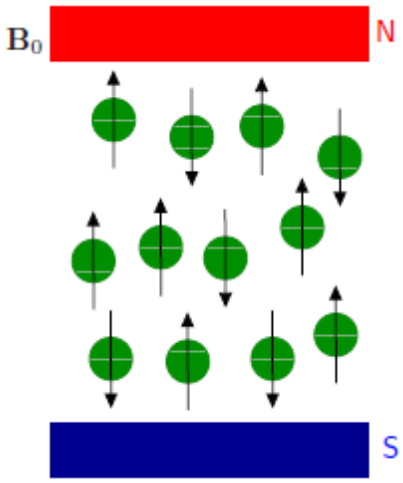


Figure 3.2: Aligned Spins

The energy difference results in the number of protons in the parallel ($n+$) and in the anti-parallel ($n-$) state to differ. The difference in the concentration is approximately 6 protons per million at ambient room temperature. The ratio is given by the Boltzmann distribution law.

$$\frac{n-}{n+} = e^{-\Delta E / k_b T} \tag{3.2}$$

Where $k_b = 1.38 \times 10^{-23}$ J/K [25].

3.1.3Excitation

Photon energy absorption takes the spins to a higher energy position. The energy E of the photon is a function of its frequency f as $E = hf$. The photon energy must completely match the energy difference $\Delta\varepsilon$ hence only a photon with the exact frequency (Larmor frequency or resonance frequency) causes the transition of a spin to the higher energy state. The resonance frequency f_o of a spin is proportional to the external magnetic field B_0 and it is derived from its energy (3.1) [25]. It is therefore expressed as:

$$f_o = \frac{\gamma}{2\pi}|B_0| \tag{3.3}$$

The resonance frequency of ^1H atoms in 1.5T magnetic fields typical for MRI is approximately 63 MHz and increases with the strength of magnetic field. Today for 6T MRI equipment, the resonance has increased to 200MHz.

3.1.4 Net Magnetization

The concept of an energy packet is used in determining the strength of magnetization. The net magnetization vector M_0 that describes the total equilibrium state magnetization of a spin packet is defined as a sum of magnetizations of all spins as follows:

$$|M_0| = \frac{\gamma\hbar}{4\pi}(n+ - n-) \approx \frac{\gamma^2 h^2 |B_0| N_s}{16\pi^2 kT} \tag{3.4}$$

Where N_s is the number of spins in the packet [22]. The vector M_0 is aligned with the orientation of the main magnetic field B_0, as shown in figure 3.2. From equations (3.1), (3.2) and (3.4) the magnitude of the vector M_0 can be increased by lowering the temperature (which may not be very versatile because the patient can only be comfortable in certain range of temperature) or by increasing the intensity of the main magnetic field $|B_0|$ [23].

The system is not always in the equilibrium state, in which case the net magnetization is composed of the longitudinal component M_z which is aligned with the orientation of B_0, and the transversal part M_T that refers to the magnetization in the plane perpendicular to the longitudinal direction.

3.1.5 Longitudinal Relaxation (T_1)

To transfer a group of spins from the lower to the higher energy state electromagnetic energy needs to be supplied at resonance frequency so that the longitudinal part of the net magnetization M_z is lower than it was in the equilibrium state $M_z < |M_0|$ .The system will fall back to the stable state with a time constant T_1. This effect is called the T_1 relaxation; it describes the decay of the M_z magnetization due to the interaction with any surrounding tissue. The longitudinal component gets back to the equilibrium in exponential fashion with a time constant T_1 that is of the order of 100 to 1000 ms and is specific for every tissue [23][24]. The z component $M_z(t)$ of the magnetization vector M_0 at time t from the excitation moment is:

$$M_Z(t) = M_z(0) + (|M_0|) - M_z(0)\left((1 - e^{-t/T_I}\right)$$

$$= \quad |M_z| - |M_0|) - M_z(0)\left((1 - e^{-t/T_I}\right) \tag{3.5}$$

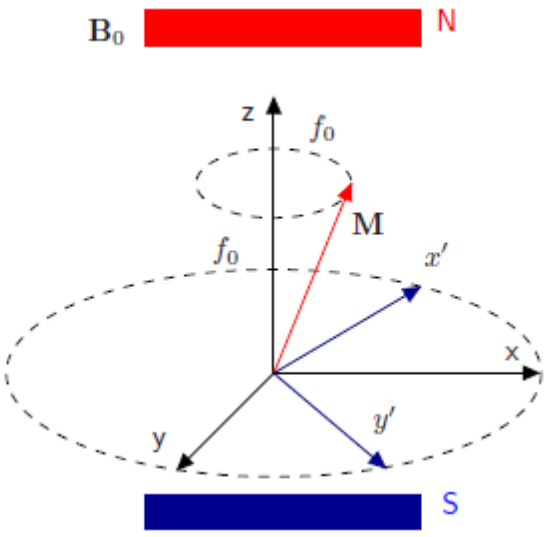
where $|M_0|$ is the intensity value of stable state and T_1 is the relaxation time [25].

3.1.6 Oscillation

The proton magnetic moments are not aligned with the external magnetic field B exactly but at a certain angle. So a torque orthogonal to the magnetic field B and the proton magnetic moment is exerted. This causes an oscillating motion of the proton magnetic moments, and subsequently of the magnetization M, around the z-axis at frequency f which is equal to the Larmor frequency (3.3).

3.1.7 Frame of Reference

Taking M deflected from the z-axis by exciting the magnetization vector M_0 into consideration. For B_0 oriented along the z-axis, the vector M oscillates around the z-axis at f_o as shown in figure 3.3(a).



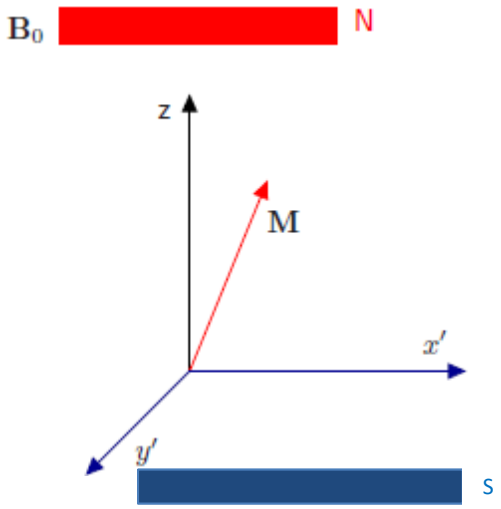


Figure 3.3(a) Rotating Frame of Reference **(b) Static Frame of Reference [25]**

Let [x', y', z] be a new rotating coordinate system termed the frame of reference. It rotates with f_o around the z-axis. In the rotating frame, the oscillating vector M seems to the static.

3.1.8 T_2Relaxation

The transverse component of M describes the magnetization in the transverse plane

$$M_T = \left| \bar{x} M_{x'} + \bar{y} M_{y'} \right| \tag{3.6}$$

Where $\bar{x}$ and $\bar{y}$ are unit vectors in respective directions. The transverse magnetization decays because there is phase difference introduced in oscillating nuclei by spin-spin and molecular interactions. The transverse magnetization decay is hence described by:

$$M_T(t) = M_T(0) - e^{-t/T_1} \tag{3.7}$$

A typical T_2 relaxation time is between 40 and 100 ms and it is always shorter than the T_1 time.

The second factor influencing the T_2 decay is the inhomogeneity caused by the susceptibility variations for the tissue. These inhomogeneities cause spins to oscillate at different frequencies. The time constant of the decay of the transversal magnetization caused by the inhomogeneity is called T_2^{inh} .The total decay time constant T_2^* is $1/T_2^* = 1/T_2 + 1/T_2^{inh}$ and it is often more than two times the pure T_2 decay [25].

3.1.9 Excitation by a Radio-Frequency Pulse

By transmitting an RF pulse at the resonance frequency energy is added to the system. When the energy is sufficient the magnetization vector M_0 is flipped from the longitudinal orientation to the transverse plane (*xy*-plane). The frequency of the magnetization vector in this plane is equal to the oscillating electromagnetic field it generates.

3.1.10 RF-pulse

The RF signal is used to shift M_0 from the equilibrium state in the longitudinal direction to the transverse *xy*-plane. This is done by creating a magnetic field B_1 rotating in the *z*-axis. This is equivalent to a field in the *x'*-direction that seems stationery. However, *M* will start oscillating around the *x*-axis at a frequency f_i because of the presence of the magnetic field B_1. This is shown in Figure 3.4.

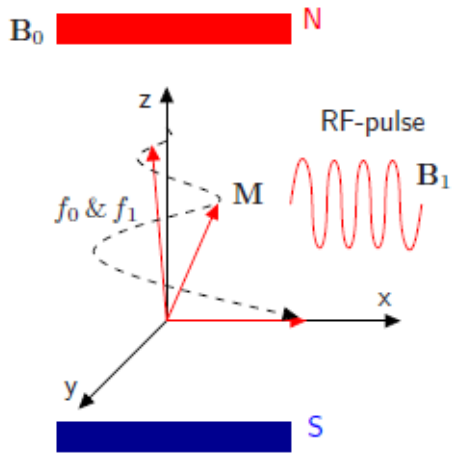


Figure 3.4 Effect of the RF pulse [25]

Source: J. Petr "Parallel Magnetic Resonance Imaging Reconstruction"

The rotating magnetic field with the intensity $|B_1|$ is turned on for time t_{90^o} whose value is given by:

$$t_{90^0} = \frac{1}{4f_i} = \frac{\pi}{2\gamma|B_1|} \tag{3.8}$$

This is called a 90° pulse. Smaller flip angles less than 90°, as well as a 180° flip angles are used for more advanced imaging sequences and are generated in the same way as the 90^0 pulse by using a longer or stronger pulse. For a typical magnetic field B_1, of order of μT, the time is of the order of milliseconds. Alignment of spins with the magnetic field B_1can be neglected because the field is turned on only for a short time and $|B_0| \geq |B_1|$.

The magnetic field B_1 which rotates along the z-axis at the resonance frequency f_0 is created by a linearly polarized field with the frequency f_0. The linearly polarized field can be broken down in two fields rotating against each other with frequencies f_0. One of the circularly polarized fields will appear stationary producing the desired field B_1. The other field will be rotating in the opposite direction with the frequency $2f_0$ and will have a negligible effect on the spins as it is far from the resonance frequency.

3.2 Nuclear Magnetic Resonance Signal

This section describes how an MRI signal is generated from NMR principles and also how the spin echo process is carried out. The description includes the spin echo process and how the gradient fields make the generated electromagnetic field position dependent which ensures that pixel intensity is properly matched to the respective tissue position.

3.2.1 Free Induction Decay

After the injection of a 90° pulse, the marginalization component M starts oscillating in the transversal xy-plane creating an electromagnetic field as shown in figure 3.5 a,b.

Considering repetitive excitation by a 90° pulse with an interval T_R between the excitations, the value of the M_z component is affected by the length of the T_R interval because of the T_1 relaxation effect in the previous excitation. For a time T_R the magnetization vector M does not achieve equilibrium, thus, the NMR signal is weaker and also dependent on the T_1 time of the imaged tissue. The transverse magnetization on the application of the 90° pulse can be described as

$$M_T(0) \alpha Q - e^{-t/T_1} \tag{3.9}$$

where Q is the number of protons in the imaged volume.

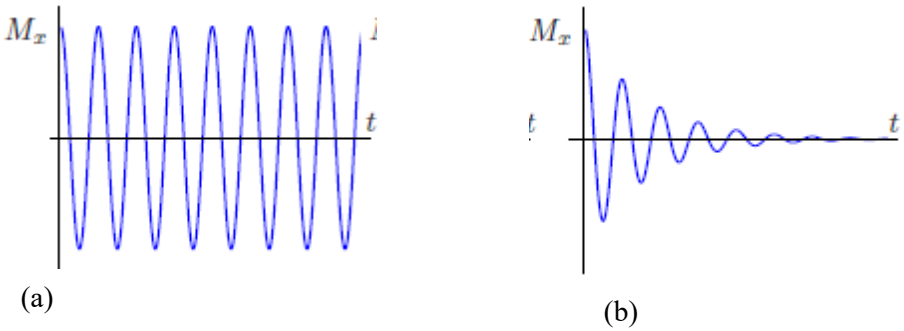


Figure 3.5: Free Induction Decay

After the 90° pulse, the oscillating motion of M generates a current in a receiver coil as depicted in Figure 3.6. The measured signal is referred to as free induction decay (FID). The FID signal is an exponential T_2^* as shown in Figure 3.5(b).

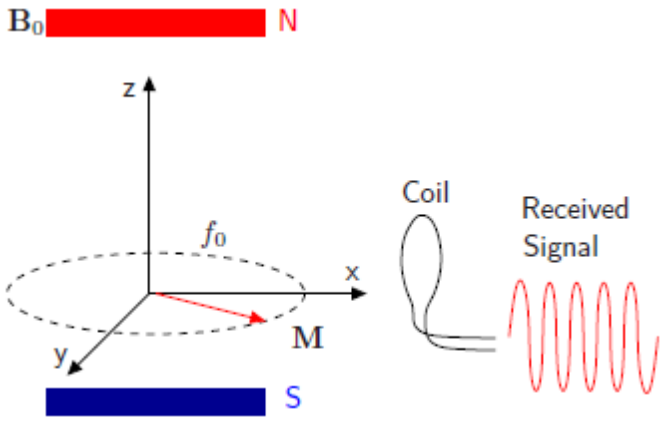


Figure 3.6: Receiver Coil and Signal

3.2.2 Quadrature Reception of NMR signal

The quadrature reception of the NMR signal enables separation of the $M_{x'}$ and $M_{y'}$ components from the acquired signal *s(t)* in the static frame. A product of the received signal with the sinusoidal wave gives $M_{x'}$ the part of the magnetization. The$_{'}$ part is obtained as a product with a sinusoidal wave shifted in phase by 90° relatively to the first one. The two components are low-pass filtered and the detected signal M_T is then treated as a complex number with $M_{x'}$ as the real part and $M_{y'}$ as the imaginary part.

$$M_T = m_{x'} + jM_{y'} \tag{3.10}$$

3.2.3 Spin Echo Process

A sequence begins with a 90º pulse that flips the magnetization vector to the transverse plane. At time $T_E/2$ a 180º refocusing pulse is applied. The signal is measured at the echo time T_E. The excitation is repeated after the repetition time 'PR > T_E. Small inhomogeneities of the core field cause the oscillation frequency of the spins to vary locally, therefore, the spins begin to diphase. To compensate for this negative effect a 180º refocusing pulse is applied after time $T_E/2$. The 180º pulse causes the dephasing spin vectors to flip in the

transverse plane, effectively changing the sign of the phase lag for each pixel. The oscillation of the spins continues in the same direction and the spins begin to rephase. After T_E, the spins are in phase again emitting the maximal signal. This signal is hence referred to as the echo top as shown in figure 3.7and figure 3.8.

Spin echo compensates for the T_2^{inh} effect caused by the inhomogeneities in the main magnetic field, thus, at the echo time the signal decay is affected only by the pure T_2 making the magnitude of the signal higher. After the echo top, the transverse magnetization relaxes with the T_2^* time constant.

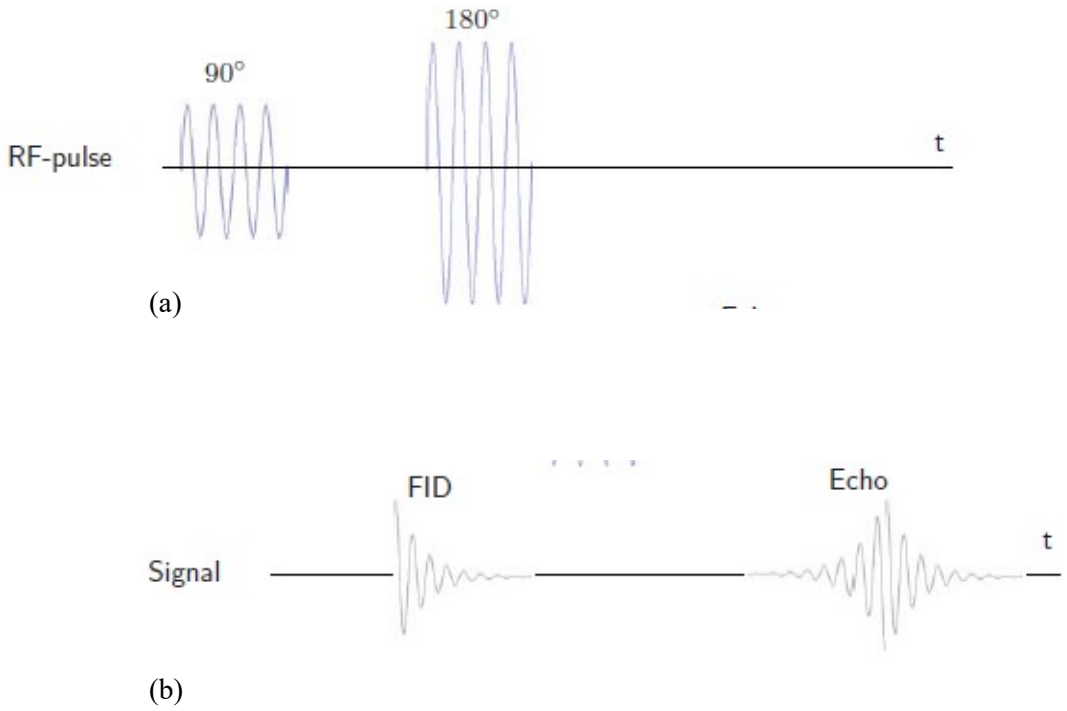


Figure 3.7: Spin Echo Sequence

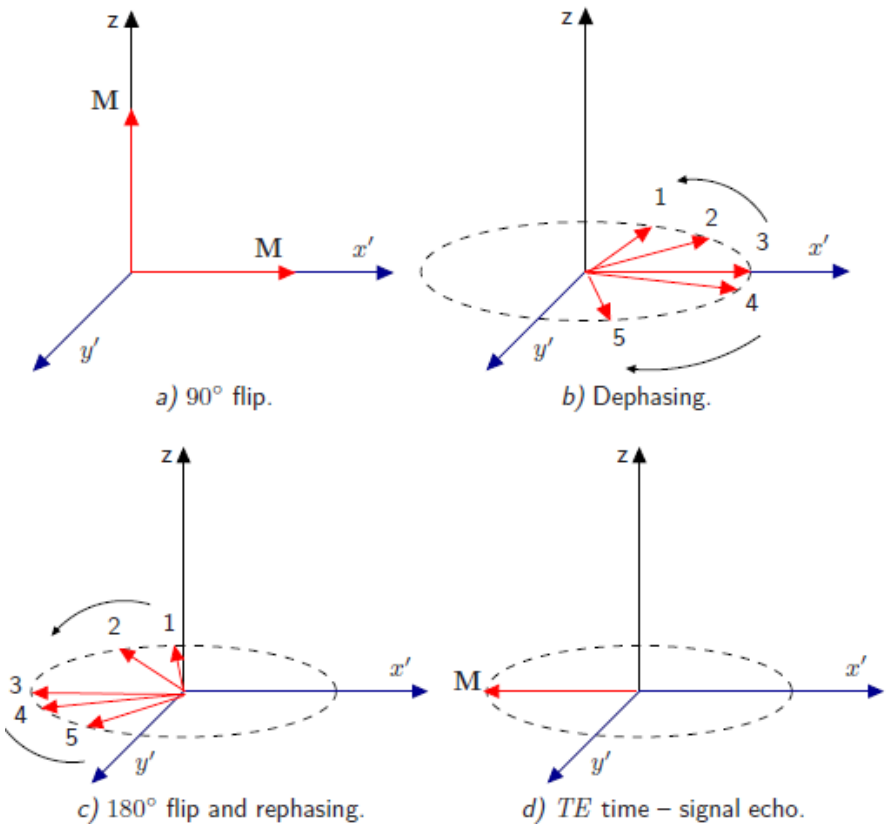


Figure 3.8: Spin echo Magnetic Vector [25]

Source: J. Petr "Parallel Magnetic Resonance Imaging Reconstruction"

The advantage of this mechanism is that it makes the spin-echo sequence less susceptible to inhomogeneities and, thus, improves the quality of the retrieved signal. Even a medical image with small inhomogeneities is then suitable for imaging with the spin-echo sequence. Magnets have been manufactured in the recent years to be much more homogeneous which makes it possible to use advanced imaging sequences with the emphasis on the speed of the acquisition without having to compensate for tissue inhomogeneities.

3.3 Magnetic Resonance Imaging Processes

The techniques that result in MR images involve spatial encoding of the NMR signal. These techniques and associated process are described. These include:

3.3.1 Gradient Magnetic Fields

Gradient magnetic fields G are oriented along the z-axis in the direction of the core field. Intensity of the gradient fields increase linearly along various axes. The total intensity of the field in the z-direction is given by

$$|B(x,y,z)| = |B_0(x,y,z)| + |G(x,y,z)| = |B_0(x,y,z)| + xG_x + yG_y + zG_z \quad (3.11)$$

where G_x, G_y and G_z are the gradient strengths.

The gradient fields make the total magnetic field intensity position dependent. Hence the resonance frequency is also space dependent

$$f(x,y,z) = \frac{\gamma}{2\pi}|B(x,y,z)| \quad (3.12)$$

The gradient fields have intensity usually several μT which are alternately switched on and off to encode the location of the NMR signal.

3.3.2 Characterizing the Magnetic Vector

The relationship between the magnetization vector M in time and the magnetic field strength is represented by the Bloch equation.

$$\frac{\partial M}{\partial t} = \gamma(M \times B) \quad (3.13)$$

where B is the total field which constitutes B_0 the core field, B_1 the excitation pulse and the gradient fields G. The total field $B = B_0 + B_1 + G$. When M is in the rotating frame, the oscillation is not visible.

The T_1 relaxation, affects the longitudinal component of the magnetization as follows:

$$\frac{\partial M_z(t)}{\partial t} = \frac{(M_z(t) - |M_0|}{T_1} \quad (3.14)$$

The T_1 relaxation affects the transversal part of the magnetization as follows:

$$\frac{\partial M_T(t)}{\partial t} = \frac{M_T(t)}{T_1} \tag{3.15}$$

where M_0 is the magnetization in the stable state and M_T is the transversal magnetization as defined in (3.6). The M in the reference frame including the effect of T_1 and T_2 relaxation, the excitation pulse $B_1$and the gradient fields G is given in equation 3.16 [25].

$$\frac{\partial M}{\partial t} = \begin{pmatrix} -1/T_2 & \gamma Gr & -\gamma B_{1y'} \\ \gamma Gr & 1/T_2 & -\gamma B_{x'} \\ -\gamma B_{1y'} & -\gamma B_{x'} & -/T_1 \end{pmatrix} \begin{pmatrix} M_{x'} \\ M_{y'} \\ M_z \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ |M_o|/T_1 \end{pmatrix} \tag{3.16}$$

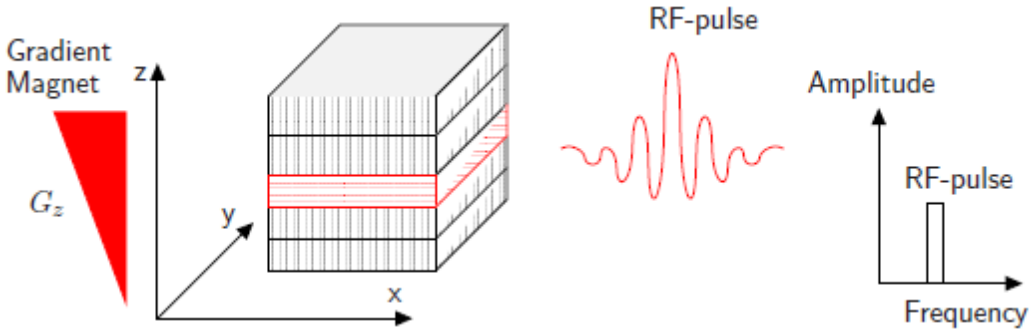
Where r is the location of the measured pixel [25].

3.3.3 Slice Selection

In MRI one slice of the imaged tissue is excited at a particular instance. This is done by imposing a gradient G to ensure the resonance frequency becomes linearly dependent on the spatial position.

$$f(z,t) = \frac{\gamma}{2\pi}|B_0| + G_z(t).z \tag{3.17}$$

An RF-pulse with a narrow bandwidth thus excites only a thin slice of the imaged object as shown in Figure 3.9. Typical slice thickness is 5 millimeters [25].



(a) Slices (b) RF-pulse (c) Conditioned RF pulse

Figure 3.9: Slice Selection [25]

Source: J. Petr "Parallel Magnetic Resonance Imaging Reconstruction"

The Spectrum of an ideal RF-pulse is a box function. Such pulse excites a slice with ideal sharp borders. However, a box function in Fourier domain corresponds to the *sinc* function with infinite duration in the time domain. In practice, an approximate RF-pulse with finite duration is used with typical duration shorter than 5ms [25]. This does not significantly degrade the Fourier profile of the slice.

3.3.4 Fourier Imaging

Fourier imaging was introduced in 1975 by Richard Ernst [27] and it is used for MR imaging even today. Fourier imaging of MRI is achieved through a process that involves a number of image processing steps. These steps include frequency encoding and phase encoding among others.

3.3.5 Frequency Encoding

The location along the *x*-axis is encoded by use of the frequency of the spins. It is therefore called frequency encoding. The *x*-direction is referred to as a read-out direction because the g gradient is switched on at the start the signal read-out. It is applied during the acquisition of the signal hence the oscillating frequency becomes dependent on *x* and is given by:

$$f(x) = \frac{\gamma}{2\pi}(B_0) + x\frac{\gamma}{2\pi}G_x \tag{3.18}$$

The frequency encoded signal excluding the relaxation phenomena is given by:

$$s(t)\alpha \iint_{slice} \vartheta(x,y)e^{-i2\pi fxt}dxdy = \iint_{slice} \vartheta(x,y)e^{iyGxxt}dxdy \tag{3.19}$$

Where $\vartheta(x,y)$ is magnetic field density.

The received signal contains components with different amplitudes. The amplitudes represent the magnetization level of the tissue. The desired components of the signal are generated by Fourier transformation of the receive signal.

3.3.6 Phase Encoding

The spatial location along the *y*-axis is phase-encoded. This is done by imposing a gradient field *nGy*, is for a short time T_y prior to the readout, *n* increases with each acquisition. The oscillation frequency altered for tan instant when the gradient is switched on and the spins

oscillate at various other frequencies. Their phase is changed depending on the y position by introducing a phase shift $y_n G_y$. The signal after the phase encoding is the signal has the form:

$$s(n,T_y)\alpha \iint_{slice} \vartheta(x,y)e^{-jynG_y yT_y}\,dxdy \tag{3.20}$$

Many phase-encoding steps with different gradient intensity nG_y have to be performed in order to get the desired image resolution.

3.3.7 Spin Echo Imaging Sequence

The sequence using spatial signal encoding by application of appropriate gradient fields is as illustrated in figure 3.10.

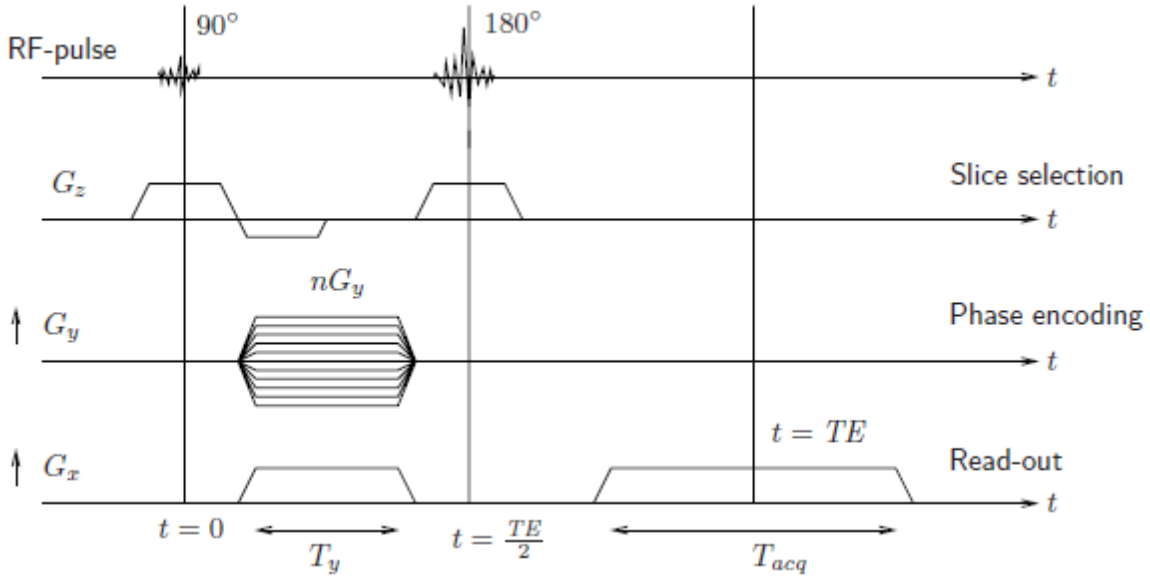


Figure 3.10: Timing Diagram of Spin Echo Sequence

Source: J. Petr "Parallel Magnetic Resonance Imaging Reconstruction"

The spin echo imaging may be described using the timing diagram in Figure3.10.At time *t=0*, the 90° pulse is used to rotate the magnetization vector *M* to the transversal plane. The slice selection gradient *G* comes on during the excitation in order to excite only a thin slice of the imaged object. The slice selection gradient *G* imposed on the z-axis causes the spins with different position on *z*-axis to oscillate with different frequencies. After the gradient is switched off, the spins with different z-coordinates are not in phase. The dephasing is compensated for by a reverse gradient (called refocusing gradient) with the total energy equal

to the half of the energy of the slice selection gradient G which is sufficient because the spins are flipped to the transverse plane after the 90° pulse and, therefore, only the second part of the slice selection gradient causes dephasing. As a result, the slice selection gradient applied during the 90° pulse does not dephases the spins and resulting in the following expression.

$$\int_{0}^{TE/2-t_0} G_z(t)dt = 0 \tag{3.21}$$

where time constant t_o is the duration of the slice-selection pulse.

After the 90° impulse, the signal is phase-encoded in the y-direction. The 180° impulse is emitted at time $T_E/2$ after the 90° impulse so as to reach the echo top at time T_E. The process is shown in figure 3.11.

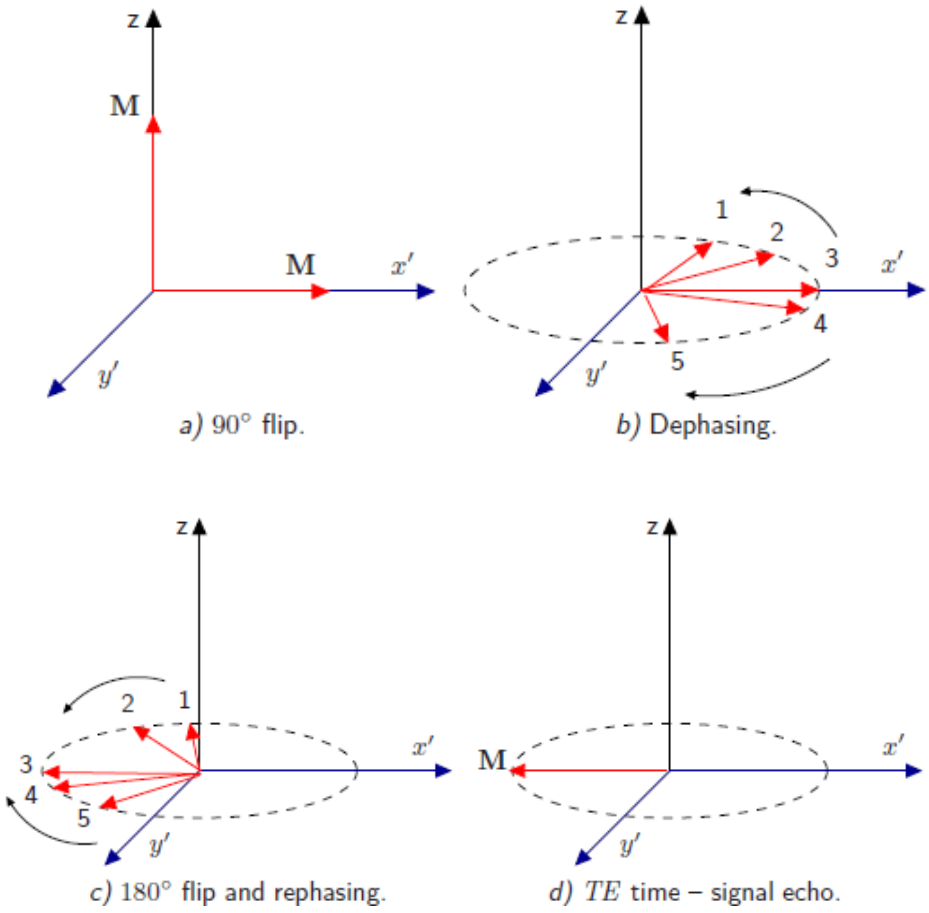


Figure 3.11: Spin Echo Signal Dephasing and Rephrasing [25]

Source: J. Petr "Parallel Magnetic Resonance Imaging Reconstruction"

The 180° pulse flips the magnetization vector midway of gradient G and, therefore, dephasing in the first half and rephasing in the second half of the second slice-selection gradient zeros out. On the average it can be expressed as:

$$\int_{TE/2-t_0}^{TE} G_z(t)dt = 0 \tag{3.22}$$

To minimize the dephasing effect of read-out gradient G_x, a gradient with the same intensity as half of the read-out gradient is turned before the 180° pulse so that evaluation becomes:

$$\int_{TE/2}^{TE} G_x(t)dt = \int_{0}^{TE/2} G_x(t)dt$$

$$\int_{0}^{TE} G_x(t)dt = 0 \tag{3.23}$$

This ensures that the spins are in phase with respect to the frequency encoding gradient at the echo time.

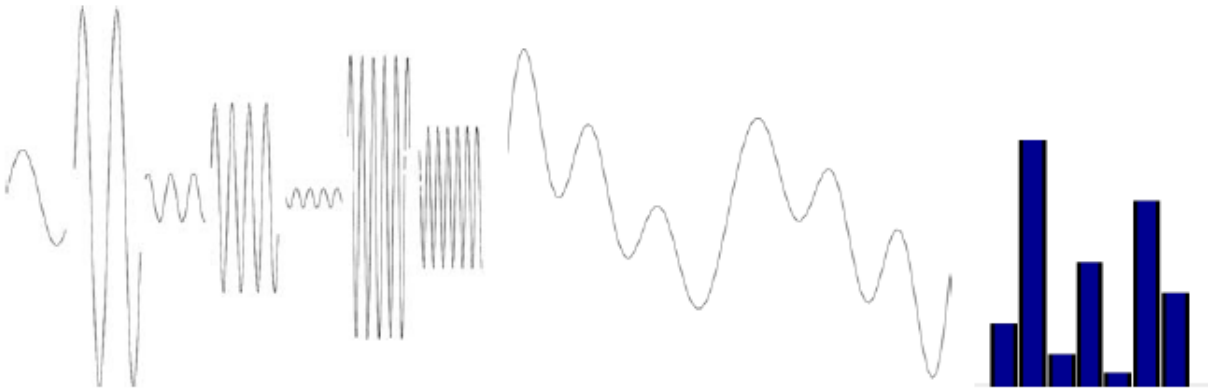


Figure 3.12: MRI Signal Formation

Dephasing resulting from the *nGy* gradient is intentional. The *nGy* gradient is switched on for time T and the integer i is changed for each excitation in order to get various phase-steps.

The signal is acquired during a time period T_{acq} centered on the echo time *TE* to ensure the maximum magnitude of the signal. The frequency-encoding gradient that modifies the oscillating frequency according to the location on the *x*-axis is turned on during the whole acquisition process.

3.3.8 Alternative Imaging Sequences

Spin-echo has been used to illustrate imaging sequence. There are several alternative techniques such as the gradient echo and the echo planar imaging (EPI) [25].

The gradient-echo method is related to the spin-echo method; however, it lacks the 180° refocusing pulse. The method is thus more sensitive to in homogeneities in the main magnetic field. Shorter echo times than in spin-echo are necessary to avoid deterioration of the signal. Gradient-echo lifts the limitation to work with 90° excitation pulses only. Smaller values of flip angle are used to reach faster imaging times.

The spin-echo and gradient-echo acquisition times are long because it is required that the system return to equilibrium between the subsequent phase-encoding steps. More echoes during a single excitation are introduced and by effectively controlling the gradients the whole image can be acquired during a single excitation. EPI is one of such single-shot imaging sequences. EPI acquisition times are many times faster than for spin-echo (on the order of 100 ms). But there is a disadvantage in EPI due to high susceptibility to main magnetic field inhomogeneities and resulting image distortion.

3.3.9 Signal Representation

The magnetization M_T is complex dependent on $M_{x'}$ and $M_{y'}$ expressed as $M_T = M_{x'} + jM_{y'}$. The complex signal *s(t)* is acquired using quadrature detection, the signal is obtained by integrating the Bloch equation as follows:

$$s(t) = M_T(0)e^{-jyr\int_0^t G(t')dt'r^{-t/T_2}} \tag{3.24}$$

Allowing *M*(x, y) be the magnetization immediately after energizing the system. The output signal *s*(t) is given by the integral of the magnetization of that slice [25]. The signal *s*(t) neglecting the relaxation phenomenon is.

$$s(t) = \iint_{x,y} m(x,y) e^{jry.f_0^t G(t')dt'} dx\ dy \tag{3.25}$$

giving

$$s(t) = \iint_{x,y} m(x,y) e^{j\theta(t,r)} dx\ dy \tag{3.26}$$

where $\theta(t,r) = yrf_0^t G(t')dt'$ is the spin phase. For simplicity, it can be assumed that the RF-signals are as impulses thus, they have no effect on the phase. The phase at time t_a=t- TE (t_a is relative time after the echo time) is,

$$\theta(t_a,r) = \int_0^{TE} \gamma r G(t')dt' + \int_{TE}^{TE+ta} \gamma r G(t')dt' \tag{3.27}$$

$$= -\gamma z \int_0^{TE/2} G_z(t')dt' - \gamma yn \int_0^{TE+2} G_y(t')dt' - \gamma x \int_0^{TE+2} G_x(t')dt' \tag{3.28}$$

$$+ \gamma z \int_{TE/2}^{TE} G_z(t')dt' + \gamma x \int_{TE/2}^{TE} G_x(t')dt' - \gamma r \int_{TE}^{TE+ta} G(t')dt' \tag{3.29}$$

The first three terms describing the gradients have negative sign. This is because of the 180^0 flip that reverse the phase θ of oscillation. The equation (3.18) can be simplified according to the properties of the spin-echo sequence. Using equations (3.13, 3.14, 3.15) the phase θ (t_a,r) at time t_a then is:

$$\theta(t_0,r) = \gamma yn \int_0^{TE/2} Gy(t')dt' + \gamma r \int_{TE}^{TE+ta} G(t')dt' \tag{3.30}$$

The signal is measured during the time period T_{acq} around the echo top (i.e., the interval $T_{acq/2}<t_a<T_{acq/2}$). In this time span, the slice-selection G_z and phase-encoding G_y gradients are off. Only the frequency encoding gradient G is active. The phase $\theta(t_a, r)$ is therefore

$$\theta(t_a,r) = -\gamma yn \int_0^{TE/2} G_y(t')dt' + \gamma r \int_{TE}^{TE+ta} G(t')dt' \tag{3.31}$$

$$= -\gamma y n \int_{0}^{TE/2} G_y(t')dt' + \gamma x \int_{TE}^{TE+ta} G_x(t')dt' \tag{3.32}$$

Assuming rectangular gradient pulses, gives:

$$\theta(t_a, r) = -y_n G_y T_y + \gamma_x G_x T_a \tag{3.33}$$

where T_y is the duration of gradient. From (3.17), the measured signal is

$$s(t_a, n) = -\iint_{x,y} m(x,y) e^{j(k_x x + k_j y)} dx\ dy \tag{3.34}$$

Where $k_y = -y_n G_y T_y$

$k_x = y G_x t_a$

The values of k_y and k_x are functions of the time t_a and of the number of phase-encoding steps n. The signal $s(t_a,n)$ is represented as a function $S(k_x, k_y)$. The domain of s is the $k_x k_y$,-plane. The magnetization $m(x, y)$ can be obtained from the measured values $S(k_x, k_y)$,(k-space domain) by performing an inverse Fourier transformation.

$$m(x,y) = \frac{1}{2\pi} \iint_{kx,ky} s(k_x, k_y) e^{j(k_x x + k_j y)} dk_x dk_y \tag{3.35}$$

In practice, to obtain k-space values of $s(k_x, k_y)$ it involves sampling the signal $s(k_x, k_y)$. The discrete time-domain image S is generated as a discrete inverse Fourier transform.

$$s(x,y) = \frac{1}{XY} \sum_{kx=-\frac{x}{2}}^{x/2} \sum_{ky=-\frac{y}{2}}^{y/2} s(k_x, k_y) e^{j(k_x x + k_y y)} \tag{3.36}$$

where X is the number of the sampling steps ($T_{acq} = t_s X$).

3.3.10 Field of View

This refers to the part of the tissue that is focused. The factors that determine FOV are the strengths of the magnetic gradients, their durations and the acquisition time T_{acq}.

Values of the real object *m(x, y)* are represented in a harmonic basis with wavelength $\lambda = 2\pi / k_x$. The unit step $\Delta k_x = yG_x t_s$ determines the wavelength and the magnitude of the field-of-view (FOV)

$$FOV_x = \frac{2\pi}{\Delta k_x} = \frac{2\pi}{\gamma G_x t_s} \tag{3.37}$$

$$FOV_y = \frac{2\pi}{\Delta k_y} = \frac{2\pi}{\gamma G_y t_y} \tag{3.38}$$

It is important to set the FOV to cover the whole imaged object. The encoding function is periodic and, thus components of the imaged tissue outside of the FOV are misinterpreted by aliasing as being inside the FOV. This effect is called the fold-over artifact [25, 28].

3.3.11 Image Contrast settings

MRI is a very versatile technique allowing displaying various features of the imaged tissue and tailored to a specific aims of the examination such as emphasize contrast agents, tumor and brain structures. There are imaging sequences that allow the acquisition of images with different properties contributing to the contrast, in the acquired image when the effect of relaxation is not taken into account and the retrieved signal is proportional to the magnetization in each pixel i.e., to the proton density.

The proton density is not the only tissue property that can be displayed. Tissues differ also in T_1 and T_2 times. By varying the sequence parameters, the echo time *TE* and the repetition time *TR*, it is possible to emphasize various properties of the imaged tissue.

The main contrast settings include [25]:

(i) Proton density; The *RF* signal is measured at time *TE* after switch on. The magnitude of the signal is proportional to $e^{-TE/T2}$ because of T_2relaxation. For echo time $TE << T_2$, the exponential term $e^{-TE/T2}$ is nearly unity for all tissues, thus, the T_2 differences

between tissues is not visible in the image. This affects the magnitude of the transverse magnetization M_T. By setting $TE << T_2$ and $T_1 << TR$ the relaxation phenomenon does not affect the signal intensity and the output image displays only the proton density of the tissue.

(ii) T_1 weighting; if the recursion time TR is short compared to T_1 then the M magnetization does not get back to the equilibrium state and the signal. Assuming a series of 90° pulses, the intensity of the magnetization shortly after the excitation is

$$M_T = |Mo|\left(1 - e^{-TR/T_1}\right) \quad (3.39)$$

By setting $TR << T_1$ the magnitude of the signal is dependent on the T_1 time and the image becomes T_1 weighted.

(iii) T_2 weighting; the variance in T_2 among tissues is emphasized by setting TE> T_2. The factor $e^{-TE/T2}$ and subsequently also the measured signal becomes dependent on the T_2 time and the images become T_2 weighted.

The effects of the T_1 and T_2 weighting can be combined to produce images that are both T_1 and T_2weighted.

3.3.12 Coil Array

For basic MRI, a receiver coil with a homogeneous sensitivity is used and is referred to as a body coil. For parallel MRI a number of receiver coils with varying sensitivities are used [31]. This avails more information on the location of the signal. For a multiple receiver, the coils are in parallel and hence, the acquisition duration is as that of a single coil. The image S_i from a particular array is given by.

$$S_i(x, y) = S(x,y)\, C_i\,(x,y) \quad (3.40)$$

where $C_i(x, y)$ is coil sensitivity and $S(x, y)$ is the resulting image retrieved by the homogeneous-sensitivity coil,

3.4 Aliasing

For parallel MRI which is an efficient and robust method of acquisition, the distance between the lines I encoding depends on a factor M, therefore reducing the steps Y while maintaining gradient intensity. This reduces the FOV and so causes aliasing. The consequences of skipping a line in the phase-encoding direction is that high frequency features are lost.

The *k*-space image s^{A} retrieved with an acceleration factor *M* is identical to the complete *k*-space image except that only every *m-th* phase encoding line is retrieved. The *A* in S^{A} is an image acquired by the accelerated acquisition and so contains aliasing. FOV is reduced only in the phase-encoding direction. Thus, inverse Fourier transformation is performed in advance to simplify the analysis. The aliased image S^{A} retrieved with an acceleration factor *M* is an inverse Fourier transform in the *j*-direction [32].

$$S^{A}(x,y) = DTF_{Y}^{-1}\left\{s^{A}(x,k_{y})\right\} \tag{3.41}$$

$$= \frac{M}{Y} \sum_{k_y=0,M,2M....}^{Y-M} s(x,k_{y}) e^{jk_{y}y} \tag{3.42}$$

Every *M*-th line is taken into account. The image *S* is transformed back to the image-domain by.

$$S^{A}(x,y) = \frac{M}{Y} \sum_{ky=0,M,2M..}^{Y-M} e^{jk_{y}y} \sum_{y'=0...}^{Y-M} s(x,y') e^{jk_{y}y'} \tag{3.43}$$

$$= \frac{M}{Y} \sum_{y'=0...}^{Y-M} S(x,y') \sum_{k_y=0,M,2M...}^{Y-M} e^{jk_{y}y'} e^{-jk_{y}y'} \tag{3.44}$$

$$= \frac{M}{Y} \sum_{y'=0...}^{Y-M} S(x,y') \sum_{k_y=0,1.2...}^{Y/M-1} e^{jk_{y}y'} e^{-jk_{y}y'} \tag{3.45}$$

The functions $e^{jk_{y}y}$ and $e^{-jk_{y}y'}$ are orthogonal and the summed over k_y for $M = 1$ produces zero for all $y \neq y'$. For $M>1$ the exponential functions are summed to a sum of *M* Kronecker delta functions in an aliasing equation as in equation 3.46.

$$S^{A}(x,y) = \sum_{u'=0}^{Y-1} \sum_{m=0...}^{M-1} \delta_{(y',y \bmod \frac{Y}{M} + m\frac{Y}{M})} S(x,y') \tag{3.46}$$

$$= \sum_{m=0...}^{M-1} S_{(x,y \bmod \frac{Y}{M} + m\frac{Y}{M})} \tag{3.47}$$

Each value in the aliased image S^{A} is a superposition of *M* values from the complete image.

3.4.1 Parallel MRI and Aliasing

Parallel MRI uses multiple receiver coils each having different spatial sensitivity to speed up the acquisition process. The factor determining the MRI acquisition duration is the number of steps. This is because each step is applied in a single excitation whose duration is determined by the recurrence time TR.

In parallel MRI, the acquisition time is decreased *M*-times by reducing the encoding steps *M*-times. Since the total *k*-space coverage is constant, this causes aliasing. The decrease of the data amount per coil is compensated by multiple coils with different spatial sensitivities. This provides more information on the spatial position of the signal. The objective of parallel MRI is to obtain an unaliased image from many aliased images using the sensitivity information of the used coils.

The process has two consecutive steps. First, the reconstruction transformation is estimated using sensitivity information extracted from unaliased images. In the second step the known reconstruction transformation is applied to the input images.

3.4.2 Reconstruction

A series of *L* aliased images S_l^A with given acceleration factor *M* are retrieved by the coil array as the input for the reconstruction step. The images S_l^A ($l = I...., L$) are modified by the sensitivity function of the array coils and they also contain aliasing.

$$S_l^A(x,y) = \sum_{N=0}^{M-1} S(x, y \bmod \frac{x}{m} + m\frac{x}{m} +) Ct(x, y \bmod \frac{x}{m} + m\frac{x}{m}) \tag{3.48}$$

where *S* stands for an ideal image acquired by a homogeneous sensitivity coil. The input images S_l^A are used to obtain a un-aliased image $\hat{S}$ using the reconstruction transformation *R*.

$$R\left(S_l^A\right)(x,y) = \hat{S}(x,y) \tag{3.49}$$

The design task is to determine optimum reconstruction operator *R* [34].

3.4.3 Estimation

A set of L unaliased array-coil images S_t and an unaliased ideal image S are acquired in the estimation step. The ideal image S is acquired using a body-coil that has an approximately homogeneous sensitivity. It may also be approximated using the unaliased images S_t. The reference images S_t, and S are used in the encoding process (3.27). These images are used to set parameter in the method and hence determine the reconstruction transformation R.

The unaliased images do not have the same characteristics as the input images with aliasing. However, it is assumed that the values of the sensitivities C_t does not change between the estimation and the reconstruction step. Another solution is by simultaneously acquiring low resolution unaliased images used for the estimation and high-resolution aliased images used for the reconstruction.

The main task of parallel imaging is to ensure the acquisition process in fast, but it has higher computation cost than the Fourier transformation used in normal imaging. A parallel based algorithm, therefore, should be designed with the reconstruction speed taken into consideration. The speed is necessary especially in the case when teleimaging and on-line interaction with the acquisition process are carried out.

3.4.4 Coil Sensitivities

When multiple receiver coils with spatially varying sensitivities are used the map is obtained as a ratio.

$$\hat{C}_l(x,y) = S_l(x,y) / S(x,y) \tag{3.51}$$

Where $S_l(x,y)$ array is the coil image and $S(x,y)$ is an image with homogeneous coil sensitivity.

The image with homogeneous sensitivities is obtained by use of sum of squares methods because in summing up of individual images pixel by pixel phase cancellation artifacts may be introduced [25, 30, 31, 32].This is shown in equation 3.52.

$$S^{SoS}(x,y) = \sqrt{\sum_{l=1}^{L} \left| S_1(x,y) \right|^2} \tag{3.52}$$

(i) SMASH

This is Simultaneous Acquisition of Spatial Harmonics the most common parallel MRI method, in which the missing lines are generated directly into the *k*-space by a combination of the neighbouring lines. This method uses a weighted composition of the array-coil images to generate the harmonic modulation produced by phase encoding gradients. Figure 3.13 explain the process graphically.

$$C_0^{comp}(x,y) = \sum_{l=1}^{L} w(l,o)C_l(x,y) \tag{3.53}$$

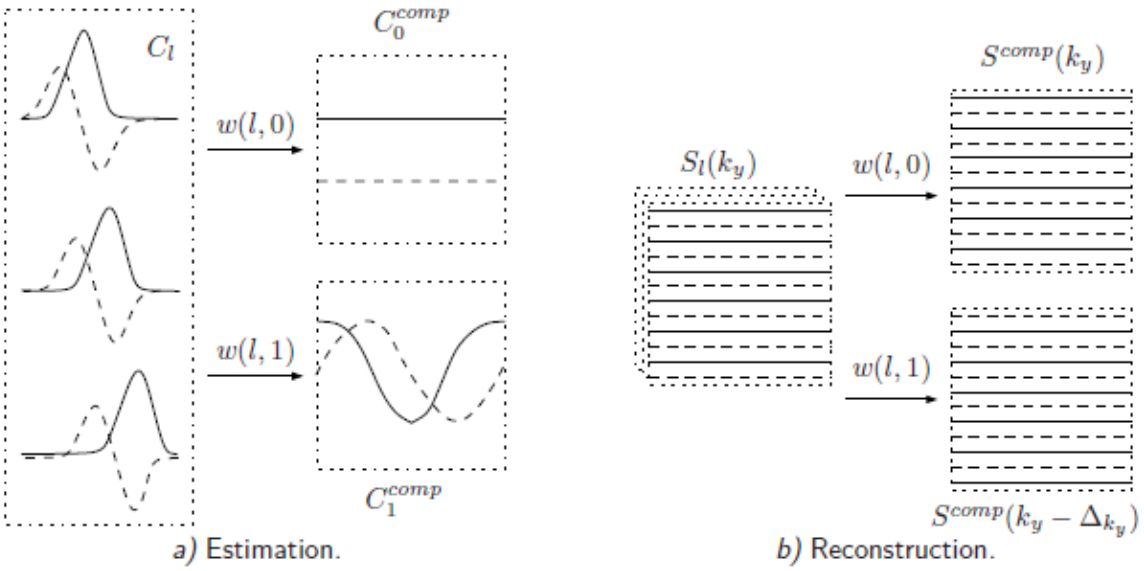


Figure 3.13: SMASH Estimation and Reconstruction

Then the harmonic spatial modulation on the top of the composite profile is produced by using the weights $w(l,m)$ therefore:

$$C_0^{comp}(x,y) = C_0^{comp}(x,y)e^{jmy\Delta k_y}\sqrt{\sum_{l=1}^{L} w(l,m)C_l(x,y)} \tag{3.54}$$

Where Δk_y is the size of step in the *k*-space.

All *k*-space lines in the composite image are reconstructed from the under-sampled coil images using the estimated weights.

$$S^{comp}\left(k_x k_y - m\Delta k_y = \sum_{l=1}^{L} w(l,m) S_t(k_x, k_y)\right) \tag{3.55}$$

This is shown to be true as follows

$$\sum_{l=1}^{L} w(l,m) S_t(k_x, k_y) = \sum_x \sum_y \sum_{l=1} w(l,m) C_t) x, y) S(x,y) e^{-jk,x - jk,y}$$

$$= \sum_x \sum_y C_0^{comp} e^{jmy\Delta k_y} S(x,y) e^{-jk_x x - jk_y y}$$

$$= \sum_x \sum_y C_0^{comp} S(x,y) e^{-jk_x x - j(k_y - m\Delta k_y) y}$$

$$= \sum_x \sum_y C_0^{comp} (x,y) e^{-jk_x x - j(k_y - m\Delta k_y) y}$$

$$= S^{comp}\left(k_x, k_y - m\Delta k_y\right) \tag{3.56}$$

When compensated

$$\hat{S}(x,y) = \frac{S^{comp}(x,y)}{C^{comp}(x,y)} \tag{3.57}$$

with

$C_0^{comp}(x,y) = \sum_{l=1}^{L} C_l(x,y)$ a sum of the coil sensitivities

Or $C_0^{comp}(x,y)$=1 for homogeneous case

And then using the equation (3.55)

$$C_0^{comp}(x,y) = \sum_{l=1}^{L} w(l,0) C_l(x,y) \tag{3.58}$$

(ii) Auto Calibration

There is need for an auto-calibration method in SMASH to avoid errors associated with the need for precise estimates of sensitivity maps. Raw images are used for calibration instead of coil sensitivities. In this case:

$$C_0^{comp}(x,y)S(x,y) = \sum_{l=1}^{L} w(l,0)C_l(x,y)S(x,y) \quad \text{and}$$

$$S_0^{comp}(x,y) = \sum_{l=1}^{L} w(l,0)S_l(x,y) \tag{3.59}$$

Auto calibration lines $S_l^{AC}\left(k_x, k_y^0 - m\Delta k_y\right)$ are acquired for single coordinates and are linearly combined with the weights $w(l,0)$ to form composite images described by:

$$\sum_{l}^{L} w(l,m)S_l^{AC}(k_x, k_y^0 - m\Delta k_y) = S^{comp}(k_x, k_y^0 - m\Delta k_y) \tag{3.60}$$

And the composite image used to find weights as follows:

$$\sum_{l}^{L} w(l,m)S_l\,(k_x, k_y^0) = S^{comp}(k_x, k_y^0 - m\Delta k_y) \tag{3.61}$$

The weights w in (3.61) are estimated for a set of k-space lines B

$$\forall m = 1..M-1; \forall k_y \in B \sum_{i=1}^{L} w(l,m)s_l(k_x, k_y) = S^{comp}(k_x, k_y - m\Delta k_y) \tag{3.62}$$

And the estimated values for each parameter $w(l,m,k_y)$

(iii) GRAPPA

This is Generalized Auto-calibration Partially Parallel Acquisitions**.** This method uses the following equation to generate the missing k-space lies of the j^{th} coil as follows:

$$s_j(x, k_y - m\Delta k_y) = \sum_{L=1}^{L}\sum_{b=B} w(j,b,l,m)s_l\left(x, k_y - bM\Delta k_y\right) \tag{3.63}$$

Images used in GRAPPA to allow both estimation and reconstructions when they are of variable density without need for additional scan. . The value of the weights $w(j,b,l,m)$ is found by solving the following equation.

$$S_j^{AC}(x,k_y - m\Delta k_y) = \sum_{l=1}^{L}\sum_{b\in B} w(j,b,l,m)s_t(x,k_y - bM\Delta k_y) \tag{3.64}$$

(iv) Temporal GRAPPA

Another MRI reconstruction method that does not need variable density scan and used in dynamic imaging is Temporal GRAPPA which has an effect of lowering the acquisition time. In the *k-t* GRAPPA the auto-calibration is in the centre of k-space lines for the reconstruction and is described by

$$S_j^t(k_y - m\Delta k_y) = \sum_{l=1}^{L}(\sum_{b\in B} w_b(j,b,l,m)s_t^t(k_y - bM\Delta k_y) + \sum_{v=t-m,t+A-m} w^v(j,l,m)s_l^v(k_y - m\Delta k_y)) \tag{3.65}$$

3.5 Sensitivity encoding method

Sense works in the image domain and uses a matrix E to describe the generation of the *k*-space values $s_l(k_x,k_y)$ including the coil sensitivity modulation and aliasing.

$$s_l\left(k_x,k_y\right) = \sum_{k_x,k_y} S(x,y)E_{lk_x,k_y}(x,y) \tag{3.66}$$

And the image is reconstructed using a matrix F, the reconstructed image given as

$$\hat{S}\left(x,y\right) = \sum_{l,k_z,k_y} F_{l,k_x,k_y}\left(x,y\right)s_l\left(k_x,k_y\right) \tag{3.67}$$

While F is

$$F = \left(E^H\Psi^{-1}E\right)^{-1}E^H\Psi^{-1} \tag{3.68}$$

3.5.1 Cartesian SENSE

In matrix form the Cartesian coordinates S are given by:

$$\begin{pmatrix} S_1^A(x,y) \\ S_2^A(x,y) \\ . \\ . \\ S_L^A(x,y) \end{pmatrix} = C \begin{pmatrix} S(x,y) \\ S(x,y+Y/M) \\ . \\ . \\ S(x,y)+(M-1)Y/M) \end{pmatrix} \tag{3.69}$$

Where coil sensitivity matrix C is given by:

$$\begin{bmatrix} C_1(x,y)... & C_1(x,y)+(M-1)Y/M \\ C_2(x,y)... & C_2(x,y)+(M-1)Y/M \\ . & \\ C_L(x,y)... & C_L(x,y)+(M-1)Y/M \end{bmatrix} \tag{3.70}$$

It is required that C is not singular in any x or y and the reconstruction becomes:

$$C_+(x,y) = \left(C^H \Psi^{-1} C + \Lambda\right)^{-1} C^H \Psi^{-1}(x,y) \tag{3.71}$$

C^H is the Hermitian transpose of matrix C.

The final image is computed from the input aliased images using the following process.

$$\begin{bmatrix} S(x,y) \\ \hat{S}(x,y+Y/M) \\ . \\ . \\ \hat{S}(x,y)+(M-I)Y/M \end{bmatrix} = C_+(x,y) \begin{bmatrix} S_l^A(x,y) \\ S_2^A(x,y) \\ . \\ . \\ S_l^A(x,y) \end{bmatrix} \tag{3.72}$$

Regularization

Tikhonov regularization and minimization is used to get the unbiased image

$$\hat{S} = \arg\min_s \left\{ \left\| CS - S_L^A \right\| + \lambda \left\| S - S^0 \right\| \right\} \tag{3.73}$$

where λ is the regularization parameter, $\| \ \|$ is the L_2 norm and S^0 is the expected solution

A different regularization approach is introduced by Kellman in [35]. The solution to the SENSE equation (3.66) is modified by adding a regularization term to the equation (3.71)

$$C+(x,y) = A\left(C^H \Psi^{-1} C + \Lambda\right)^{-1} C^H \Psi^{-1} (x,y) \tag{3.74}$$

A is a gain matrix and **Λ** is a conditioning matrix. The matrix ***A*** is computed to make the diagonal elements of ***C****+(x,y)*equal to 1. The values in the matrix **Λ** regularize the process by controlling the trade-in between SNR and the artifact minimization in the reconstructed image. The non diagonal elements of ***C****+(x,y)*specify the artifacts suppression. The matrix ***C****+(x,y)*becomes identity for ***Λ*** and equal zero hence the artifacts are fully suppressed at the cost of relatively high noise in the image. The values in ***Λ*** are optimized to meet the desired artifact suppression level in the reconstructed image. By changing some values in ***Λ*** the reconstruction SNR may be improved while retaining the artifact suppression at an acceptable level [25].

Iterative solution

When the noise correlation matrix is taken to be diagonal using equations 3.63

and 3.65 gives $E_l^H E_l \hat{S} = E_l^H S_l$ (3.75)

where the vector S_l is defined as $\mathbf{s}[l,k_x,k_y]$.

For a given vector ***z*** matrix multiplication $\boldsymbol{E_z}$ is

$$\left(E_z\right)_{l,kx,ky} = \sum_{x,y} e^{j\pi\left(k_x x + k_y y\right)} C_l(x,y) z(x,y) \tag{3.76}$$

Re-writing as an inverse discrete Fourier transformation gives

$$(E_z)_{l,kx,ky} = \iint_{x',y'} e^{j\pi(k_x x' + k_y y')} \sum_{x,y} C_l(x,y) z(x,y) \delta(x - x', y - y') dx' dy' \tag{3.77}$$

$$= FT^{-1}\left[\sum_{x,y} C_l(x,y) z(x,y) \delta(x - x', y - y') \right]_{x',y'} (k_x, k_y) \tag{3.78}$$

There is also the iterative SENSE, where *k*-space is allowed to be sampled using arbitrary sampling. Therefore, an additional transformation is required to transform the data to a regular Cartesian grid prior to the discrete Fourier transformation.

3.6 SPACE RIP

Sensitivity Profiles from an Array of Coils for Encoding and Reconstruction in Parallel, generates the un-aliased image directly from the acquired *k*-space values, which may be represented as an inverse Fourier transform.

$$s_1(x, k_y) = \sum_y S(x,y) C_l(x - y) e^{-\pi j l_y y} \tag{3.79}$$

In matrix form it is possible to express the encoding process for all the coils and all phase-encoding coordinates in both image domain and *k*-space.

$$\begin{bmatrix} S_1(x, k_y^1) \\ \\ S_1(x, k_y^{N_k}) \\ S_2(x, k_y^1) \\ . \\ . \\ S_1(x, k_y^{N_k}) \end{bmatrix} = C \begin{bmatrix} C_1(x,1) e^{-\pi j k_y^1} \ldots C_1(x,Y) e^{-\pi j k_y^1 y} \\ \cdots \\ C_1(x,1) e^{-\pi j k_y^{N_k}} \ldots C_1(x,Y) e^{-\pi j k_y^{N_k} y} \\ C_2(x,1) e^{-\pi j k_y^N} \ldots C_2(x,Y) e^{-\pi j k_y^1 Y} \\ . \\ C_L(x,1) e^{-\pi j k_y^{N_k}} \ldots C_L(x,Y) e^{-\pi j k_y^{N_k} Y} \end{bmatrix} \begin{bmatrix} s(x,1) \\ S(x,Y) \end{bmatrix} \tag{3.80}$$

3.7 Partially parallel imaging With Localized Sensitivities

Partially parallel imaging With Localized Sensitivities (PILS) is a simple and fast reconstruction method that utilizes the high spatial localization of sensitivity in smaller coils. Such coils acquire images with a limited field of view according to coil sensitivity.

3.8 Parallel Magnetic Resonance Imaging with Adaptive Radius in *k*-space

Parallel Magnetic Resonance Imaging with Adaptive Radius in *k*-space (PARS) does not work independently on the *x* coordinate like SMASH or GRAPPA; it uses neighbourhoods of each pixel to contribute to its reconstruction. Each un-acquired pixel in the *k*-space is reconstructed using pixels that have the Euclidean distance from lower than k_R

$$s_1(k_x k_y) = \sum_{\Delta x \Delta y} \sum_{t-1}^{L} w_l, l', \Delta_x \Delta_y \; sl'(k_x - \Delta_x, k_y - \Delta_y) \tag{3.81}$$

With *w* being the weighting coefficients and the sum is over all $\Delta x, \Delta y$ which gives the relationship $\sqrt{\Delta_x^2 + \Delta_y^2} \le k_R$

It was shown in [25] that weights for which a perfect reconstruction can be obtained and given by:

$$C_1(x, y) = \sum_{\Delta x \Delta y} \sum_{l'-1}^{L} w_l, l', \Delta_x \Delta_y e^{pi(\Delta_x \Delta_y y)} \; C_l'(x, y) \tag{3.82}$$

3.9 Pre-Enhancement

There are very many applications of the various Parallel MRI methods in pre-enhancing the image before final interpretation to solve specific issues.

3.9.1 Handling Motion Artifacts

Motion artifacts mainly result from patient movement during acquisition. In [25] a method is described which involved taking fully sampled *k*-space images with motion artifacts and dividing the *k*-space to several sets each containing a distinct subset of the acquired *k*-spaces. In each set the missing lines are generated from the rest using SENSE. Images reconstructed are the same except for errors. The *k*-space lines that differ from the others are discarded and replaced by those reconstructed from neighbours using SMASH which is better localized in the frequency domain. This method ensures motion artifacts are minimized in the final image.

3.9.2 CAIPIRINHA

Controlled aliasing in Parallel Imaging Results in Higher Acceleration for multi-slice imaging is used for speeding up multiple acquisitions using parallel MRI principles. Since two or more slice are acquired together it is not easy to distinguish the pixel values from each slice by means of a standard signal encoding. A solution is proposed in [36] where the phase of each slice is modified by a value Δy .The acquired signal $S(y)$ for two slices is then:

$$S_l^A(y) = \sum_{ky} \left(s_l(k_y)C_1^1(k_y) + s_2(k_y)C_1^2(k_y)e^{j\pi k_y \Delta_y}\right)e^{j\pi k_y \Delta_y}$$

$$= S_l(y)C_1^1(y) + S_2(y-\Delta_y)C_2^1(y-\Delta_y) \tag{3.83}$$

Another method used to accelerate a dynamic MRI is UNFOLD (Un-aliased by Fourier-Encoding the Overlaps using the Temporal dimension) which uses a single coil receiver. The acquisition process time is speeded up by under-sampling the *k*-space which causes aliasing which cannot be removed without prior knowledge. The temporal information is hence used to minimize the aliasing and time dependent phase shift is generated. The constructed image is therefore described by equation 3.84.

$$S^A(x,y,t) = \sum_{m=0}^{M-1} S(x, y \bmod \frac{Y}{M} + M\frac{Y}{M}, t)e^{j\pi m f(t)} \tag{3.84}$$

3.10 Diffusion Magnetic Resonance Imaging

3.10.1 The Einstein Approach

Diffusion is a process in which molecules of a given substance try to distribute equally in a fluid.

The rate at which the process takes place dependents on fluid characteristics and the concentration of the molecules. The diffusion process is used in dynamic Magnetic resonance Imaging to track body activities that involve movement of molecules in body fluids. These processes include absorption of chemical medicines by body tissues, blood flow and many others.

To develop a mathematical model for one dimensional diffusion, the concentration is denoted as $C(x, t)$. The flux normal to the concentration gradient is denoted:

$$F = -D\frac{\partial C}{\partial x} \tag{3.85}$$

where D is the diffusivity of the fluid. However the dynamic relationship has the form

$$\frac{\partial C}{\partial t} = D\frac{\partial^2 C}{\partial x^2} \tag{3.86}$$

which is referred to as Flicks' equation.

For n molecules located at $x=0$ and taking all other variables at their initial conditions to be 0 the diffusion process is described by:

$$C(x,t) = \frac{n}{\sqrt{4\pi Dt}}\exp\left(-\frac{x^2}{4Dt}\right) \tag{3.87}$$

For isotropic diffusion in a homogeneous fluid in all directions, the generalization to three dimensions is possible. The distribution is given by

$$P(r/r_0,t) = \frac{1}{(4\pi Dt)^{3/2}}\exp\left(-\frac{(r-r_0)^2}{4Dt}\right) \tag{3.88}$$

The distribution is found to be a vector with $\sigma^2 = 2Dt$. The distance that the molecule will travel can therefore be expressed as:

$$|r-r_0|^2 = \left\langle (x-x_0)^2 + (y-y_0)^2(z-z_0)^2 \right\rangle$$

$$= \left\langle (x-x_0)^2 \right\rangle + \left\langle (y-y_0)^2 \right\rangle + \left\langle (z-z_0)^2 \right\rangle$$

This is equal to the sum of the resulting variances in the three directions this is given as

$$\left\langle (r-r_0)^2 \right\rangle = \sigma_x^2 + \sigma_y^2 + \sigma_z^2 = 6DT \tag{3.89}$$

The Gaussian covariance matrix and displacement distribution is found to be

$$\Sigma = 2Dt = \begin{bmatrix} 2D_{xx}t & 2D_{xy}t & 2D_{xz}t \\ 2D_{sy}t & 2D_{yy}t & 2D_{yz}t \\ 2D_{xx}t & 2D_{yz}t & 2D_{zz}t \end{bmatrix} \tag{3.90}$$

For isotropic diffusion as derived as [20]:

$$D_{xx} = D_{yy} = D_{ZZ} = D \qquad D_{xy} = D_{xz} = D_{yZ} = 0$$

3.10.2 Diffusion Tensor Imaging

Diffusion tensor MRI can be achieved by the addition of diffusion weighting gradients to either side of refocusing pulse to the spin echo MRI method. One gradient will set the phase of the spins which will be proportional to the gradient while other will introduce equal and opposite rephasing for stationary spins. Turner [21] proposed that for a diffusion weighting gradient of magnitude G, applied for a time t, the log-ratio between the signal A, after the full echo time, t_e, and that produced by the initial 90° *rf* pulse can be described by:

$$In\left(\frac{A(b)}{A_0}\right) = y^2\delta^2\left(\Delta - \frac{\delta}{3}\right)G^2 D^{eff} = -bD^{eff} \tag{3.91}$$

Effective diffusivity, D_{eff} is averaged over the diffusion time. In tissue that have an anisotropic diffusion profile, D_{eff} "constant" will also vary with the orientation of the diffusion gradient. In this case, Eq. (3.91) generalises to:

$$In\left(\frac{A(b)}{A_0}\right) = y^2\delta^2\left(\Delta - \frac{\delta}{3}\right)G^2 R^T D^{eff} R = -\sum_i \sum_i b_{ij} D_{ij}^{eff} \tag{3.92}$$

where R is a vector showing the direction of the gradient, and b_{ij} constitute a matrix, b, which is analogous to the scalar weighting factor in Eq. (3.91). The elements b_{ij} encode various interactions between diffusion and imaging gradients [20].The mean diffusivity (MD) given by:

$$MD = (D) = \frac{Tr(D)}{3} = \frac{\lambda_1 + \lambda_2 + \lambda_3}{3} \tag{3.93}$$

Where λ_1, λ_2 and λ_3 are diffusion coefficient in the x, y, z coordinates respectively.

D is used to describe the average diffusivity This quantity does not provide information on isotropy of the tensor, since it uses only the mean of the Eigen values. There is, moreover, no single obvious way to compute anisotropy. Three other measures used include fractional anisotropy (*FA*), relative anisotropy (*RA*) and the volume ratio (*VR*), which are defined as follows [20].

$$FA = \sqrt{\frac{3}{2}} \sqrt{\frac{\left(\lambda_1 - \langle D \rangle\right)^2 + \left(\lambda_2 - \langle D \rangle\right)^2 + \left(\lambda_3 - \langle D \rangle\right)^2}{\lambda_1^2 + \lambda_2^2 + \lambda_3^2}} \tag{3.94}$$

$$RA = \frac{1}{\sqrt{3}} \sqrt{\frac{\left(\lambda_1 - \langle D \rangle\right)^2 + \left(\lambda_2 - \langle D \rangle\right)^2 + \left(\lambda_3 - \langle D \rangle\right)^2}{\langle D \rangle}} \tag{3.95a}$$

$$VR = \frac{\lambda_1 \lambda_2 \lambda_3}{\langle D \rangle^3} \tag{3.95b}$$

Where $\langle D \rangle$ is mean diffusivity

3.10.3 Motion Correction

Magnetic Resonance Imaging has motion artifacts introduced by patient motion during imaging so overall resolution suffers significantly. It is therefore necessary to remove any artifacts caused by motion error. Many methods of overcoming this limitation exist. A common approach [21] involves removal of motion artifacts via use of convex projections. Temporal redundancy available in time-resolved MRI is exploited to remove motion artifacts. Phase filtering is combined with projections in Fourier-space and image space to remove motion artifacts. These projections should protect the features from degradation.

Subtraction of remnant images from post-contrast images is then carried out to obtain an image. Patient motion may lead to small intensity changes so the effect is not only the reduction of image quality but can also obscure important temporal events like the arrival of contrast agent and the temporal resolution of the image. This may lead to the loss of information even for slight patent motion [18, 19].

POCS-based method iteratively applies successive constraints to the corrupted frame, making it more similar to an artifact-free reference frame that is computed from other input frames. The constraints applied consist of four projections, two depend on image space and two on Fourier space (called *k*-space in MR). These projections are designed specifically to prevent obliteration of vascular features, and to ensure stability and convergence of the POCS algorithm. The method improves visual quality of the target image.

Procedure

The technique begins by identifying mask and arterial phase MRI. Motion-corrupted frames are corrected using the POCS algorithm. The process repeats for each corrupted frame in the sequence. It is assumed that an uncorrupted reference image is available from the reference set, the set of n_{ref} frames preceding the corrupted frame by getting their median. The number of reference frames used depends on the level of motion degrading the image. It involves two other steps.

***K*-space Box Constraint**

Patient motion can cause points in *k*-space of the corrupted frame to differ from corresponding points in the reference frame. The solution of these undesirable changes is performed by the projection *k*-restrict, which distorted *k*-values to be within a small range of the reference *k*-values. This results in a box constraint applied to *k*-space values, since it defines a convex polyhedral referred to as spherical "box" around a reference point within which any solution must reside.

Phase: Correction

The second projection, represented by the box, is a phase filter for correcting translational motion artifacts. It is based on the fact that these artifacts vary linearly in *k*-space, whereas the phase information due to contrast arrival and concentration change is fast-varying and erratic.

Intensity inhomogeneity (IIH) correction

The visualized MRI signal results from the measurements of the tissue composition. MRI images pixel levels are taken to be piecewise constant except for tissue boundaries and the underlying noise which is additive. However various imaging conditions introduce an additional multiplicative noise factor, referred to as the intensity inhomogeneity (IIH).

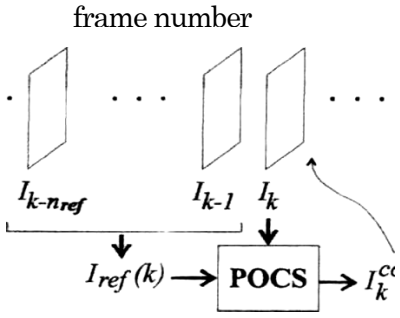


Figure 3.14: Motion Correction on MRI

The sources of IIH are generally divided in two groups [21]:

(i) Arising from properties of the MRI device such as static magnetic field inhomogeneity, RF signal energy spatial distribution and others.

(ii) Arising from the characteristics of image including spatial variations of the permeability and dielectric properties of the object.

The correction of the IIH degradation involves of estimating the IIH field and removing (dividing or subtracting) this it from the given image. This correction is an essential step before other image processing and segmentation. It improves the performance of such algorithms.

For ideal conditions which include no partial volume effect, and no IIH, no variations due to the imaging device the segmentation of MRI would be trivial once the signal intensity for each kind of tissue is known. Where, these artifacts are present there is the need for adaptive and robust segmentation algorithms. Some algorithms perform both the IIH field estimation and the MRI segmentation into the tissue regions of interest.

There are two main categories of correction algorithms. These are parametric and non-parametric ones. The first requires a model of the inhomogeneity field, so that estimated values fit the model. The second algorithms perform estimation of the inhomogeneity field value at every pixel location of the measured MRI. Two approaches have been developed in [21].A parametric algorithm whose assumed model of the IIH field is linear combinations of outer products of Legendre polynomials. Estimation is performed by error function gradient descent for model parameters as well as the tissue class intensity means. Therefore this can be an IIH correction and segmentation algorithm.

CHAPTER FOUR

IMAGE DENOISING APPROACHES

Filtering is one of the most explored methods used to reduce the noise level in an image. Its efficiency is associated with the particular spectral energy distribution in an image. For most images, the noise free spectrum is mainly found in low frequencies. It is often taken to decay like $\left(\frac{1}{f\alpha}\right)$ where f is the oscillation frequency and α is the decay co-efficient close to 2 for many images. For AWGN, the noise spectral density is constant. Therefore, low pass filtering of image can improve the signal-to-noise ratio (SNR). However for MRI images where the noise distribution is Rician distributed adaptation to this kind of distribution is required [1]. In this chapter several denoising methods are presented.

4.1 Wiener Filter

This is one of the filters that has been successively used in image denoising. It is formulated as follows:

$$y=x+b \tag{4.1}$$

Where both the noise free image x and the associated noise b are independent random processes. The problem is to estimate x by $x=w*y$ and to obtain the linear filter w that minimizes the (MSE), i.e.

$$w_{opt}= \arg_{min w} E\{\|w*y-x\|^2\} \tag{4.2}$$

Where E{-} is expectation whose solution is the Wiener filter. Its frequency response is given by

$$H(\omega)=\frac{s_x(\omega)}{s_x(\omega)+s_b(\omega)} \tag{4.3}$$

Where $s_b(\omega)$ is the power spectral density of the noise free signal x and $s_b(\omega)$ of the noise b.

It is optimum for a Gaussian stationary random signal only while MRI noise is not Gaussian. Various methods to estimate the image signal and noise signal power spectral density have been investigated. The most common approaches use some parameterized models. Wiener filter is thus usually out performed by spatially adaptive and non-linear algorithms [17]. Wiener filter is obtained by the product of the centred noisy image signal by a given weight.

$$x = \hat{x}\left(1 - \frac{\sigma^2}{\sigma_y^2}\right)\left(y - \mu_y\right) + \mu_y \tag{4.4}$$

Where σ^2, σ_y^2 and μ_y denote noise variance, image variance and mean of the noisy data respectively. This approach has been found to be versatile and can be extended to transform domain such as the Wavelet transform.

4.1.1 Other Linear Filters

The moving average and Gaussian smoothing are other categories of filters that may be used when noise and signal statistics are not available. However, their performance has limitation of edge blurring and introduction of artifact features.

4.1.2 Non-linear Filters

Another set of filters include the non-linear category such as the median filter. It is mainly used in removing impulsive noise. Combination of morphological operators is another method which has been used where AWGN, salt and pepper, speckle and Rician noises are present [35].

4.1.3 Bilateral Filter

Bilateral filtering in its initial formulation was to use domain filtering and range filtering. Domain filtering takes advantage of the similarity of spatially close pixels which are more correlated to the centered pixel than more distant pixels. Therefore, a weighed averaging of neighbouring pixels reduces the noise level. However, it does not perform well in the vicinity of edges. Improved method that takes into consideration image discontinuities consists of grouping similar pixels without regard to spatial locations. The filter computes a weighted average of samples around a central pixel, where the weights are the products of the spatial

domain and the range domain weights. The three parameters that control the bilateral filter are the neighbourhood size and parameters set the decay of the two weight factors [35][36][37][38][39].

4.1.4 Patch based Approaches

This set of image denoising methods use the redundancy of the various structural features present in natural images. It can particularly be efficient in boundaries where an edge can be taken as a collection of similar binary patches. Of importance in application of this approach is a method of measuring similarity between image patches.

At a given location *n*, the estimate $\hat{x}_n$ of the noisy pixel $y_n = x_n + b_n$, is computed as follows

$$x_n = \frac{1}{c}\sum H_h(y_n, y_k) G_g(n,k) y_k \tag{4.5}$$

$$\text{where} \quad C = \sum H_h(y_n, y_k) G_g(n,k) y_k$$

y_n is a set taken from a neighbourhood around the location *n*,*k* are the indices of any pixels in the neighbourhood y(*n*) of the current pixel location *n*.

The function $H_h(y_n, y_k)$ is a measure of similarity patches y_n and y_k. It is formulated as:

$$y_k H_h(Y_n, Y_k) = \exp\left(\frac{\|y_n - y_k\|^2}{2h^2}\right) \tag{4.6}$$

h is a thresholding parameter derived from noise standard deviation. A function $G_g(n,k)$ generates weights based on distance to the current pixel location. When the patch is taken as the pixel the overall function becomes a bilateral filter.

One of the major challenges in these patch based methods is to have minimum offset for various dimensions of the patches. Hence the design of the two weighting functions H_h and H_g, as well as the value of their respective smoothing parameters *h* and *g* need to be done with certainty [21].

4.1.5 Variational Approaches

Anisotrophic Diffusion

The goal is to obtain successive version of an original image I_o(x,y) at various resolutions. This is done by convolving the noisy image with Gaussian Kernels of increasing variance. It has been shown that this subset of filtered images is a solution of the diffusion equation. It is formulated as [21]:

$$\frac{\partial}{\partial t} I(x, y, t) = div\{c(x, y, t)\nabla I(x, y, t)\}. \tag{4.7}$$

With the initial condition $I(x, y,0) = I_0(x, y).\nabla = \left[\frac{\partial}{\partial x}\frac{\partial}{\partial y}\right]^T$ is the special gradient operator.

When diffusion conductance *c(x,y,t)* is a constant *C*, linear diffusion results in homogenous diffusivity. This solution leads to a Gaussian smoothing of the noisy image. In that case, there is a difficulty in distinguishing the treatment of edges and flat regions of the image.

Perona and Malik [40] used an image dependent diffusivity *c(x,y,t)* $= h(|\nabla I(x, y, t|))$. The diffusion is dependent on image gradient at any given time *t*.

$$\vartheta_n(w, s, a) = \sum_{k=1}^{2} a_k \max\left(1 - \lambda_k \frac{4\gamma_n(s)}{\gamma_n^2}, 0\right) \tag{4.8}$$

The effect is that the edges are clearer, with flat regions being smoothed. This method is a non-linear anisotropic diffusion which is found to be superior to Gaussian smoothing. The tricky aspect of the method is the choice of an appropriate diffusivity function. Various analytical functions used have a conductance parameter. In addition to the category of diffusivity functions and conductance, two other parameters need to be selected. They include the diffusion speed and the number of iterations [21].

4.1.6 Total Variation Minimization

The total variation (TV) method was first proposed by Rudin, and others [21]. In this approach, denoising is formulated as a constrained minimization problem. Letting $l_0(x, y)$ be the noisy image which is described as:

$$I_0(x,y) = I(x,y) + B(x,y) \tag{4.9}$$

Where *I(x,y)* is the noise free image and *B (x,y)* is an additive noise independent from *I (x,y),* such that $E\{B\} = 0$ and $E\{B^2\} = \sigma^2$

The clean image is the solution of

$$\arg_{\min} u \int_{\Omega} |\nabla U(x,y) dx\, dy \;\; subject\ to \quad \Big| \frac{1}{|\Omega|} \|U - I_0\|^2 = \sigma^2 \tag{4.10}$$

where

$$\|U - I_0\|^2 = \int_{\Omega} (u(X,Y) - I_0(x,y))^2 dx\; dy \tag{4.11}$$

With a multiplier λ , the solution for (4.10) takes the form of unconstrained cost functional minimization given by:

$$J(U) = \int_{\Omega} |\nabla U(x,y)| dx\; dy \; + \frac{\lambda}{2} \int_{\Omega} (U(x,y) - I_0(x,y))^2 dx\; dy \tag{4.12}$$

With appropriate value of y

∇J is obtained by use of the Euler- Lagrange equation of (4.12) as

$$\nabla J = \lambda (U(x,y) - I_0(x,y)) - \nabla \left\{ \frac{\nabla U(x,y)}{|\nabla U(x,y)|} \right\} \tag{4.13}$$

Where the minimization of the functional is then obtained by gradient descent

$$\frac{\partial}{\partial t} U(x,y,t) = div \left\{ \frac{\nabla U(x,y,t)}{|\nabla U(x,y,t)|} \right\} + \lambda (I_0(X,Y) - U(x,y,t)) \tag{4.14}$$

Total variational therefore becomes the link between various PDE based approaches, including the anisotropic diffusion, and the various forms of regularized cost functional, Setting λ =0 in Equ. (4.14) gives a case where diffusivity *c (x,y,t)* = $1/|(\nabla U(x,y,t)|)$ [21].

4.2 Transform Domain Approaches

The transform domain approaches have been used singly or in combination for medical image denoising. The development of powerful multi-resolution methods has then contributed a lot to the preference of this denoising approach. Appropriate transform method should have the following properties [21]:

(i) Invertibility: Any transform used in MRI denoising should have a perfect inverse transform. This ensures that the resulting image has minimum distortions.

(ii) Linearity: The major task in MRI denoising is to be able to clearly identify and distinguish components of a transform expression that contain noise and those that mainly contain the image. Linearity ensures this and also enhances invertibility.

(iii) Computational efficiency: Denoising is usually one of the stages in image processing. It is therefore desirable that denoising contributes minimal computational cost. This is due to the fact that other processing stages such as segmentation have relative high computational cost.

(iv) Decorrelation ability: Since the choice of transform domain denoising includes its ability to remove inter pixel dependencies. For a given category of signals, point-wise operations in an appropriate transform domain can be already very efficient such as the Wiener filter for stationary processes. The decorrelation ability of the transform is also necessary in Bayesian denoising. It is used to enable description of the prior on the noise free data.

(v) Energy compaction: For efficiency and effectiveness it is required that transform energy of the important images features is concentrated in a small number of coefficients.

(vi) Shift Invariance: Various transform like DCT and wavelet have Artifacts when spatial shift is used as a step either in transformation or processing of an image. For these and any other transform to be used in MRI image denoising they need to be made shift invariant.

The typical procedure for shift invariance may involve steps as follows:

Step (i)	Shift the input image signal
Step (ii)	Apply the DCT transform
Step (iii)	Denoise the shifted transformed coefficients
Step (iv)	Apply the inverse DCT transform
Step (v)	Shift back the denoised output.
Step (vi)	Repeat steps 1 to 5 for a range of shifts and average the various denoised outputs.

A few shifts are used in practice since a large number may be computational expensive. These include overlapping blocks for block-transform (e.g. sliding window DCT). Where wavelets transform is the choice as the case in this thesis, better shift invariance can be obtained with:

Un-decimated wavelet transforms complex and dual-free wavelets and sharper band pass filters. It has low computational cost in comparison with decimated wavelets.

In rotation invariance the edges and other features present in natural images take different form in random orientations. Rotation invariance can ensure these edges and features will remain constant. The issue with rotation invariance is that it is not easy to be used since it would require interpolation of the input image sampling grid leading to artificial correlations between neighbouring pixels. Interpolation is omitted when rotation is along concentric rectangles.

When the wavelet transform is employed then rotation invariance introduces directional sensitivity in horizontal and vertical axis. It is possible to have higher directional sensitivity by use of steerable multiscale transforms. Directional wavelet transforms have been designed, and there are a number of categories, one involves transforms without orientations, such as isotropic wavelets. Other techniques that are used include data adaptive transformations. The Karhunen- Loeve transform can be used to decorrelate the data has been efficiently used for image denoising algorithms [21].

4.3 Wavelet-Domain Filtering

Wavelet-domain is desired because the orthogonal wavelet transform concentrates the signal energy into a few of coefficients. Therefore, the wavelet transform of the noisy image consists of few coefficients with high SNR and many coefficients with low SNR. On removing the noisy coefficients, the image are obtained using the inverse wavelet transform therefore achieves invertibility. Wavelet transform enables the denosing procedure to adapt to the spatial variations in the signal frequency content hence avoids excessive smoothing. It therefore performs better than the Fourier domain method. Other wavelet based image processes such as noise removal, compression, and signal recovery can achieve optimal performance characteristics and, moreover, do not introduce excessive artifacts in the denoised image [1].

4.3.1 The Discrete Wavelet Transform

The 1D discrete wavelet transform (DWT) represents a real-valued continuous-time signal $f(t)$ in terms of shifts and dilations of a low pass scaling function $\phi(t)$ and band pass wavelet $\varphi(t)$. For refined choices of these functions, the shifts and dilations form an orthonormal basis for L_2 (IR) and the signal representation becomes [1]

$$f(t) = \sum_{k} c^{j}_{k^{2}-\frac{j}{2}} \Phi\left(2^{-J_T-\kappa}\right) + \sum_{j=-\infty}^{J} \sum_{d} d_k^j \, 2^{-j/2} \varphi\left(2^{-j_t} k\right) \tag{4.15}$$

with scaling coefficients

$$c_k^J = 2^{-J/2} \int f(t) \phi\left(2^{-J_t} - \right) dt \tag{4.16}$$

and wavelet coefficients

$$d_k^j = 2^{-j/2} \int f(t) \psi\left(2^{-j_t} - \right) dt \tag{4.17}$$

Another advantage of the wavelet is multiresolution: the first term in (4.15) is an approximation to $f(t)$ at resolution J, and the second term consists of refinements at finer scales $j < J$. The wavelet basis functions are localized in both time and frequency; hence the DWT is superior to many other transform for signals with spatial varying frequency content.

It is also possible to achieve rotational and shift variance as well as many other requirements such as piecewise linearity and computational efficiency.

The scaling and wavelet coefficients may be computed using efficient recursion. Since

$$\phi\left(t/2\right) = \sqrt{2\sum_n h[n]\,\phi(t-n)} \tag{4.18}$$

$$\psi\left(t/2\right) = \sqrt{2\sum_n g[n]\phi(t-m))} \tag{4.19}$$

with *h* and *g* being discrete-time low pass and highpass filters, respectively, the coefficients are computed as follows:

$$c_k^{j+1} = \sum_n h[n]\, c_{2k-n}^j \tag{4.20}$$

$$d_k^{j+1} = \sum_n g[n]\; c_{2k-n}^j \tag{4.21}$$

These computations can be formulated as discrete-time filter bank.

For a length 2^M discrete-time signal $s = [s[1], s[2],\ldots, s[2^M]]^T$, the recursion (4.20) (4.21) beginning with $c^0{}_k = s[k]$ defines a discrete-time DWT of s[5]. Iterating (4.20) (4.21) *J* times, produces an *J*-scale DWT consisting of *J* sets of wavelet coefficients at scales $j = 1,.., J$, and a single set of scaling coefficients at scale *J*. The maximum number of iterations is $Jmax = M$., whose computational cost is linear in the number of signal samples processed. Another reason that the wavelet is the transform of choice is that that the transforms of real world images tend to be sparse, with a few large scaling and wavelet coefficients dominating the decomposed representation, hence tend to compress the images. Wavelet coefficients of polynomial signals are exactly zero so the transform exhibits compression properly due to the "vanishing" moments of the wavelet functions, let ψ_k^j denote the underlying discrete-time wavelet basis function at scale *j* and location *k*. Then:

$$\sum_m m^n\; \psi_k^j\; [m] = 0,\; n = 0,\ldots\ldots, N-1 \tag{4.22}$$

Wavelet functions are found to have N vanishing moments. The number of vanishing moments of ψ_k^j determines the highest degree polynomial signal whose wavelet coefficients are all zero. However, for higher N there are more polynomials which will have non-zero wavelet coefficients. The noise present in an image tends to be evenly distributed over all wavelet and scaling coefficient which are orthonormal so white noise is identically distributed in each coefficient. A simple procedure of denoising an image is therefore removing all the wavelet coefficients.

Real signal and images may be represented by a piecewise polynomial function. In this case coefficients will be zero except those of the wavelets that have large values near the breakpoints of the polynomial pieces. These non-zero wavelet coefficients are required when inverting the multi resolution components back to a complete image so as to retain details like edges. For effective noise removal in the wavelet-domain it is necessary to identify which wavelet coefficients do not have significant signal energy and hence can be discarded wavelet filters.

The two dimensional discrete wavelet transform and inverse transform are given in equations (2.28 and 2.29). The discrete *2D* wavelet functions $\psi_{k,0}^{j,1}$ and scaling functions $\phi\, j,k,l$ have several indices *j* corresponding to the scale, *o* corresponding to wavelet orientation (horizontal, vertical, or diagonal), *k, 1* corresponding to the position. A universal index *I* is used together with spatial index.

$$c_I = \sum_m \phi 1[m] s[m] \tag{4.23}$$

Similarly the I-th wavelet coefficient is computed

$$d_I = \sum_m \psi I\,[m] s\,[m] \tag{4.24}$$

It is normal to refer to the scaling and wavelet coefficients by vectors *c* and *d*, respectively.

4.3.2. Wavelet-Domain Filtering

The process involves computation of the 2D DWT of an image f(*x,*y). The goal is to obtain a better estimate of noise-free signal wavelet coefficients by filtering *d*. Filtering of the coefficient of a particular wavelet b_I to the signal reconstruction by weighting the corresponding coefficient d_1 by a number $0 \le \alpha_1 \le 1$ is carried out. This is represented by:

$$\overset{\iota}{d}_1 = \alpha_1 \, d_1 \tag{4.25}$$

Setting $\alpha_1 = 0$ totally ignores the contribution of the wavelet function ψ_1 setting $\alpha_1 = 1$ leaves it in its original form. Selecting $0 < \alpha_1 < 1$ reduces the contribution of the *I*-th wavelet function. The objective is to eliminate those wavelet coefficients which contain more noise than signal. After this the inverse DWT of the filtered wavelet coefficients *d* and the scaling coefficients *c*, result in improved output image. The new image denoted *s* is a wavelet-domain filtered form of *x*. The weights $\alpha = \{\alpha_1\}$ is the effective wavelet-domain filter.

The main task therefore in wavelet-based filtering is the design. Ideally the wavelet-based filtering procedure can be adapted to the local SNR in each wavelet coefficient, as to be able to remove wavelet coefficients with very low SNR. Assuming that the noisy wavelet coefficient is an estimator of the value of the signal's wavelet coefficient its mean that $\partial_1 = E\left[d_1\right]$ .That is, ∂_1 is the wavelet coefficient of the noise-free signal. The mean squared error filter of each coefficient is:

$$\alpha_1^{MSE} = \frac{\partial_1^2}{\partial_1^2 + \sigma_1^2} \tag{4.26}$$

Where σ_1^2 is the variance of the noisy coefficient d_I. The best weight is given by minimizing $E\left[(\partial_1 - \alpha_1 \, d_1)^2\right]$ with respect to α_1. This results into wavelet-domain analogue of the Wiener filter [1]. However, this optimal filter weight is not practically possible as it requires perfect knowledge of σ_1 *and* σ_1^2 and therefore is not feasible in practice.

This may be approximated by:

$$\alpha_1 = \left(\frac{d_1^2 - \sigma_1^2}{d_1^2} \right) \tag{4.27}$$

where d_1^2 is the square value of the wavelet coefficient and σ_1^2 is the variance of the d_1. Again, assuming that $z(x) = \sum_{jEI} \alpha_j \varphi_j(x)$ is simply an estimate of the numerator in (4.26), and *d* is an estimate of the denominator in (4.26). Therefore, (4.27) provides a reasonable approximation to (4.26), and interpreted (4.26) as adaptive Wiener filter. A threshold action on this expression ensures that α does not at any time have negative values. This is automatically achieved when sum of squared errors is used.

The working of the filter (4.27) is to set small wavelet coefficients, with squared magnitude less than the estimated variance, to zero and to leave larger coefficients unchanged. Other operations including hard and soft-thresholds, can also be employed. But these are inferior to the operation given by (4.27) in MRI and other imaging applications [1]. So this threshold is normally used.

More noise removal is possible by weighting the estimated variance in (4.27). For example,

$$\alpha_1 = \left(\frac{d_1^2 - I\ \sigma_1^2}{d_1^2} \right) \tag{4.28}$$

where $I \leq 1$ is a user-defined parameter that adjusts the amount of noise reduction. Selecting $I \leq 1$ effectively raises the threshold level of the nonlinearity. Universal threshold for Gaussian noise removal that is greater than unity has also been used, although larger threshold levels can result in greater noise reduction, but may also lead to over smoothing. The choice of *T=1* provides a conservative lower bound on the threshold level. In MRI applications, *T=2* provides very good results in terms of squared error and visual quality [1]. In [1] the Haar wavelet was shown to be better at preserving fine image details such as small vascular structure compared to other wavelets. The Haar wavelet also has the most compact spatial support of all wavelets. But wavelets with larger spatial support such as the Daubechies-6 wavelet have better frequency localization and approximation properties.

Filtering with wavelets with large spatial support generally produces smooth images. But they may also lead to excessive smoothing of fine details. The Haar wavelet is the best in MRI applications since it exhibits both large number of vanishing moments and spatial support.

Another aspect is the shift-variant nature of the wavelet-domain filter adapted for MRI. To improve wavelet-domain filtering methods, shift- invariant (undecimated) DWT methods are used. Shift-invariant wavelet transforms have been shown to provide better results in many cases since they are less sensitive to misalignments between the edges in images and the wavelet basis functions.

4.3.3 Wavelet based Wiener Filter

Wiener filter may be derived from the estimation given in equation (4.29)

$$\hat{x}[n] = h[n] * y[n], \tag{4.29}$$

In this case, $h[n]$ is obtained by minimizing the estimation error that defines the Wiener filter. The frequency response of which is described in equation 4.3 as:

$$H(\omega) = \frac{P_x(\omega)}{P_x(\omega) + P_v(\omega)} \tag{4.30}$$

where $P_x[\omega]$ and $P_v[\omega]$ represent the power spectral density of the clear and the noise, respectively. The filtering effect is frequency selective so frequency bands where the signal power is much stronger than that of the noise and are sustained and frequencies where noise is abundant are attenuated.

In MRI application where noise has Rician distribution the filter designed is for denoising individual pixels. When the signal varies from one pixel to another, it is desirable to denoise each pixel by itself; hence the output still reflects this pixel to pixel variation. The SWT may be applied to ensure a stable filter due to the redundancy that exists.

Methods for estimating signal and noise have been developed. One is in which the averaging of measured data is carried out.

(i) Filter modelling

For a number of pixels consisting of *N* samples, as:

$$\langle x_0[0],..x_0[N-1],...,x_k[N-1],..x_{k-1}[0],..x_{k-1}[N-1]\rangle \tag{4.31}$$

The average is:

$$\widetilde{X}[n]=\frac{1}{K}\sum_{k=0}^{k-1}x_k[n] \tag{4.32}$$

The result of SWT on an individual pixel *k* is:

$$\left\{d_k^{j-1}[0],..d_k^{j-1}[N-1],...,d_k^{j-2}[0]\right\},...,d_k^{j-2}[N-1],...,d_k^{l}[0],...,d_k^{L}[N-1],s_k^{L}[0],...,s_k^{L}[N-1] \tag{4.33}$$

And on the average is:

$$\left\{\widetilde{d}_k^{j-1}[0],..\widetilde{d}_k^{j-1}[N-1],...,\widetilde{d}_k^{j-2}[0]\right\},...,\widetilde{d}_k^{j-2}[N-1],...,\widetilde{d}_k^{l}[0],...,\widetilde{d}_k^{L}[N-1],\widetilde{s}_k^{L}[0],...,\widetilde{s}_k^{L}[N-1] \tag{4.34}$$

The Wiener field is the wavelet domain may be obtained from equation (4.30) and expressed as transfer function:

$$H(j,n)=\frac{p_x(j,n)}{P_x(j,n)+P_v(j,n)} \tag{4.35}$$

Where $P_x(j,n)$ is the true signal *x* in location *n* at resolution level *j*. $P_v(j,n)$takes the form:

$$P_x(j,n)+P_v(j,n)\approx\frac{1}{K}\sum_{k=0}^{k-1}\left(d_k^{j}[n]\right)^2 \tag{4.36}$$

With the approximation approaching equality as *K* tends to ∞. To get an estimate of the numerator of Equation (4.35), it is noted that:

$$\left(\widetilde{d}^{\,j}[n]\right)^2\approx P_x(j,n)+\frac{1}{K}P_v(j,n) \tag{4.37}$$

This progression from (4.35) follows the results in (4.38). Combining Eqs. (4.36, 4.37) gives:

$$P_x(j,n) \approx \frac{K}{K-1}(\tilde{d}^{\,j}[n]^2 \quad + \left(\frac{1}{K} - \frac{1}{K-1}\right)\sum_{k=0}^{k-1}(d_k^j[n]^2 \tag{4.38}$$

By substituting Eqs (4.36) and (4.38) into Eq. (4.35), obtain

$$H(j,n) = \frac{\frac{K}{K-1}(\tilde{d}^{\,j}[n]^2 \quad + \left(\frac{1}{K} - \frac{1}{K-1}\right)\sum_{k=0}^{k-1}(d_k^j[n]^2}{\frac{1}{K}\sum_{K=0}^{K-1}(d_k^j[n])^2} \tag{4.39}$$

The denoised wavelet coefficients are denoted:

$$\left\{\hat{d}_k^{j-1}[0],..\hat{d}_k^{j-1}[N-1],...,\hat{d}_k^{j-2}[0]\right\},...,\hat{d}_k^{j-2}[N-1],...,\hat{d}_k^{l}[0],...,\hat{d}_k^{L}[N-1], s_k^L[0],...,s_k^L[N-1] \tag{4.40}$$

Where $\hat{d}_k^j[n] = d_k^j[n]H(j,n)$. This process is not smooth parts of an image pixel, $S_k^L[1],...,s_k^L[N]$. The effect of this operation on the image data is to alter wavelet coefficients according to the ratio noise to signal plus noise in the corresponding component.

(ii) Adaptive Multiresolution Non-local Means Filter

This filter may be formulated [4] as:

$$\breve{u}(x_i) = \sum_{x_j \epsilon V_i} w(x_i, x_j)u(x_j) \tag{4.41}$$

Where $\breve{u}(x_i)$ is restored intensity, the pixel or voxel intensity $u(x_j)$ and $w(x_i, x_j)$ is the weight assigned to intensity values. The weights are a measure of similarity between patch N_i and N_j. the weights are computed as follows:

$$w(x_i, x_j) = \frac{1}{Z_i} e^{-\frac{||u(N_i)-u(N_j)||_2^2}{h^2}} \tag{4.42}$$

Where Z_i ensures $\sum_j w(x_i, x_j)$ =1 and H is a smoothing factor. The Blockwise approach is adapted to reduce computational time in 3D image volumes. The restored values for all elements in a patch are given by:

$$\hat{u}(N_i) = \sum_{N_j \epsilon V_i} w(x_i, x_j)u(N_j) \tag{4.43}$$

The noise distribution in magnitude domain follows Rician distribution

$$p(m) = \frac{m}{\sigma^2} \exp\left(-\frac{m^2+A^2}{2\sigma^2}\right) I_0\left(\frac{Am}{\sigma^2}\right) \tag{4.44}$$

Where A is the amplitude without noise, m is magnitude value with noise, I_0 is the modified zero order Bessel function of the first kind, and $\sigma^2$is standard deviation of noise. The second order moment is denoted as:

$$\mathbb{E}\,(m^2) = A^2 + 2\sigma^2 \tag{4.45}$$

A log-likelihood function of the form:

$$\log L = \sum_{i=1}^{N} \log\left(\frac{\mathrm{m_i}}{\sigma^2}\right) - \sum_{i=1}^{N} \frac{m_1^2+A^2}{2\sigma^2} + \sum_{i=1}^{N} log I_0\left(\frac{Am_i}{\sigma^2}\right) \tag{4.46}$$

is used to optimize the procedure with Maximum Likelihood Estimators(MLE) being:

$$\hat{A}_{MLE} = \arg(A_{\max} \log L) \tag{4.47}$$

Working with squared signal enables intensity bias removal by subtraction bias

$$\hat{A}_{CA} = \sqrt{max\left(\widehat{\mathbb{E}(m^2)} - 2\sigma^2, 0\right)} \tag{4.48}$$

Blockwise conventional approach method of optimization does not required much additional computation time so it is chosen and is formulated as:

$$\hat{u}(N_i) = \sqrt{max\left(\left(\sum_{N_j \epsilon V_i} w(x_i, x_j) u(N_j)^2\right) - 2\sigma^2, 0\right)} \tag{4.49}$$

When multiresolution involving Discrete Wavelet Transforms is used with NLM filters the filtering parameters for various resolutions applied result in different sets of filtering parameters, Sub-band mixing of the denoised coefficients is therefore necessary for appropriate recombination of various image components. Sub-band mixing can be hard or soft.

The procedure for hard sub-band mixing can be written as:

$$\hat{c}_k = \hat{c}_{k,u}$$

$$\hat{d}_k = \hat{d}_{k,u} \text{ for LLH, LHL and HLL}$$

$$\hat{d}_k = \hat{d}_{k,o} \text{ for HHL, HLH, LHH and HHH} \tag{4.50}$$

Where $\hat{c}_k$are scaling coefficients and$\hat{d}_k$wavelet coefficients

$$\widehat{\boldsymbol{T}}_{\boldsymbol{b}}(\boldsymbol{\sigma}) = \frac{\boldsymbol{\sigma^2}}{\widehat{\boldsymbol{\sigma}}\boldsymbol{X}} \quad \text{with} \tag{4.51}$$

$$\hat{\sigma}X = \sqrt{max(\hat{\sigma}_b^2 - \sigma^2)}$$

The resulting coefficients are:

$$\hat{c}_k = \hat{c}_{k,u}$$

$$\hat{d}_k = \emptyset\left(d_{k,}\hat{T}_b(\sigma)\right)\hat{d}_{k,u} + (1 - \emptyset(d_{k,}\hat{T}_b(\sigma)))\hat{d}_{k,o} \tag{4.52}$$

$$\phi\left(d_k, \hat{T}_b(\sigma)\right) = \frac{1}{1+e^{(-\lambda(|d_k| - \hat{T}_b(\sigma)))}} \tag{4.53}$$

$$I_n(x_j) = \sqrt{I_r(x_j)^2 + I_i(x_j)^2} \tag{4.54}$$

4.4 Image Signal and Noise Modelling and Estimation

4.4.1 Rician LMMSE Estimator

Noise in magnitude MRI image has Rician distribution so the PDF of the images is [3]

$$pM(M_{ij} \backslash A_{ij}, \sigma_n) = \frac{M_{ij}}{\sigma_n^2} e^{\frac{M_{ij}^2 + A_{ij}^2}{2\sigma_n^2}} I_0\left(\frac{A_{ij}M_{ij}}{\sigma_n^2}\right) u(M_{ij}) \tag{4.55}$$

For an image background where SNR is low the distribution is Rayleigh

$$pM(M_{ij} \backslash \sigma_n) = pM(M_{ij} \backslash A_{ij} = 0, \sigma_n) = \frac{M_{ij}}{\sigma_n^2} e^{\frac{M_{ij}^2}{2\sigma_n^2}} u(M_{ij}) \tag{4.56}$$

Noise estimation mainly uses background intensities. Hence estimators based on mean and second order moments are:

$$\widehat{\sigma_n^2} = \frac{1}{2n}\sum_{i=1}^{N} M_i^2 \,, \widehat{\sigma_n} = \sqrt{\frac{2}{\pi}}\frac{1}{n}\sum_{i=1}^{N} M_i \tag{4.57}$$

4.4.1.1Conventional Approach

The estimate is expressed in terms of noise standard deviation and second order moment in a Rician distribution as:

$$\hat{A}_c = \sqrt{\max(\langle M^2 \rangle - 2\sigma_n^2, 0)} \tag{4.58}$$

The sample estimator is defined

$$\langle I \rangle = \frac{1}{|\eta|} \sum_{p \epsilon \eta} I_p \tag{4.59}$$

With η a square neighbourhood.

4.4.1.2 Maximum Likelihood Estimator

The maximum likelihood function is used

$$A_{ML} = arg_A^{max}(\log L) \tag{4.60}$$

with

$$\log L = \sum_{i=1}^{N} log\left(I_0\left(\frac{AM_i}{\sigma_n^2}\right)\right) - \frac{NA^2}{2\sigma_n^2} - \sum_{i=1}^{N} \frac{M_i^2}{2\sigma_n^2} \tag{4.61}$$

Where N is the number of samples considered for the likelihood function.

4.4.1.3 Expectation-maximization (E M) Method

In this method noise variance and signal are estimated simultaneously by maximizing the log likelihood expectation.

$$\hat{A}_{k+1} = \frac{1}{N} \sum_{i=1}^{N} \frac{I_1\left(\frac{\hat{A}_k M_i}{\hat{\sigma}_k^2}\right)}{I_0\left(\frac{\hat{A}_k M_i}{\hat{\sigma}_k^2}\right)} M_i \tag{4.62}$$

$$\sigma_{k+1}^2 = \max\left[\frac{1}{2N} \sum_{i=1}^{N} M_i^2 - \frac{\hat{A}_k^2}{2}, 0\right] \tag{4.63}$$

The initial values are computed as:

$$\hat{A}_0 = \left(2\left(\frac{1}{N} \sum_{i=1}^{N} M_i^2\right)^2 - \frac{1}{N} \sum_{i=1}^{N} M_i^4\right)^{1/4} \tag{4.64}$$

$$\hat{\sigma}_0^2 = \frac{1}{2}\left(\frac{1}{N}\sum_{i=1}^{N} M_i^2 - \hat{A}_0\right) \tag{4.65}$$

4.4.1.4 The Analytical Exact Solution

Using Rician noise model and using sample mean *M*

Given the correlation factor as

$$\zeta(\theta) = 2 + \theta^2 - \frac{\pi}{8} e^{-\frac{\theta^2}{2}} \left[(2+\theta^2) I_0\left(\frac{\theta^2}{4}\right) + \theta^2 I_1\left(\frac{\theta^2}{4}\right)\right]^2 \tag{4.66}$$

and the root θ of

$$g\langle\theta\rangle = \sqrt{\zeta(\theta)\left(1 + \frac{\langle M\rangle^2}{\langle M\rangle^2 - \langle M\rangle^2}\right)} - 2 - \theta \tag{4.67}$$

The signal estimate and the noise variance are given by:

$$\sigma^2 = \frac{\langle M\rangle^2 - \langle M\rangle^2}{\zeta(\theta_0)} \qquad and \qquad \hat{A} = \theta_0 \hat{\sigma} \tag{4.68}$$

4.4.1.5 LMMSE Estimator for Rician Model

The estimator for a parameter may be denoted as [3] the LMMSE Estimator for Rician model

The estimator for a parameter may be denoted as:

$$\theta = E\{\theta\} + C_{\theta x} C_{xx}^{-1}(x - E\{x\}) \tag{4.69}$$

Where $C_{\theta x}$ is the cross covariance vector and C_{xx} is the covariance matrix of the data

For the 2D Rician model and using local statistics.

$$\widehat{A_{ij}^2} = E\{A_{ij}^2\} + C_{A_{ij}^2 M_{ij}^2} C_{M_{ij}^2 M_{ij}^2}^{-1}(M_{ij}^2 - E\{M_{ij}^2\}) \tag{4.70}$$

For Rician noise distribution, the square LMMSE estimator for the 2D is therefore

$$\widehat{A_{ij}^2} = E\{A_{ij}^2\} \frac{E\{A_{ij}^4\} + 2E\{A_{ij}^2\}\sigma_n^2 - E\{A_{ij}^2\}E\{A_{ij}^2\}}{E\{M_{ij}^4\} - E\{M_{ij}^2\}^2} \; x \; \left(M_{ij}^2 - E\{M_{ij}^2\}\right) \tag{4.71}$$

When the expectations are substituted with their sample estimators the estimators are expressed as:

$$\widehat{A_{ij}^2} = \langle M_{ij}^2 \rangle - 2\sigma_n^2 + K_{ij}\left(M_{ij}^2 - \langle M_{ij}^2 \rangle\right) \tag{4.72}$$

Where

$$K_{ij} = 1 - \frac{4\sigma_n^2\left(\langle M_{ij}^2 \rangle - \sigma_n^2\right)}{\langle M_{ij}^4 \rangle - \langle M_{ij}^2 \rangle^2} \tag{4.73}$$

For a N dimensional volumes the expectation is given by:

$$\widehat{A_{ijk}^2} = \langle M_{ijk}^2 \rangle - 2\sigma_n^2 + K_{ijk}\left(M_{ijk}^2 - \langle M_{ijk}^2 \rangle\right) \tag{4.74}$$

With

$$K_{ijk} = 1 - \frac{4\sigma_n^2\left(\langle M_{ijk}^2 \rangle - \sigma_n^2\right)}{\langle M_{ijk}^4 \rangle - \langle M_{ijk}^2 \rangle^2} \tag{4.75}$$

And

$$\langle I_{i,j,k} \rangle = \frac{1}{|\eta_{i,j,k}|} \sum_{p \epsilon \eta_{i,j,k}} I_p^2 - \langle I_p \rangle^2 \tag{4.76}$$

In the background where distribution is Rayleigh the noise is estimated as

$$\widehat{\sigma_n} = \sqrt{\frac{2}{\pi}}\ mode\left\{\langle I_{i,j,k} \rangle\right\} \tag{4.77}$$

If the background is not empty the local sample variance is computed as

$$\widehat{\sigma_n^2} = mode\left\{\sigma_{i,j,k}^2\right\} \tag{4.78}$$

$$\sigma_{i,j,k}^2 = \frac{1}{|\eta_{i,j,k}| - 1} \sum_{p \epsilon \eta_{i,j,k}} I_p^2 - \langle I_p \rangle^2 \tag{4.79}$$

4.5 A CURE: Chi-Square Unbiased Risk Estimation

For a vector of *N* samples drawn from non-concentric chi-square distribution with parameters defined as centrality measure $x_n \geq 0$ and $K>0$ as the degrees of freedom [16] the vector *Z* maybe denoted

$$Z \sim X^2 k_{(x)} \quad and \quad Z \epsilon R^{+N}$$

Statistical characterization of the observation model gives:

$$p(y|x) = \prod_{n=1}^{N} p(y_n|x_n) = \prod_{n=1}^{N} \frac{1}{2} e^{-\frac{x_n+y_n}{2}} \left(\frac{y_n}{x_n}\right)^{\frac{K-2}{4}} I_{\frac{K}{2}-1}(\sqrt{x_n y_n}) \tag{4.80}$$

The characteristic function is given by:

$$\hat{p}(w|x) = \prod_{n=1}^{N} \frac{exp\left(-\frac{jw_n x_n}{1+2jw_n}\right)}{(1+2jw_n)^{K/2}} \tag{4.81}$$

$$E\{y\} = x + K.1,$$
$$\text{And } E\{\|y\|^2\} = \|x\|^2 + 2(K+2)1^T x + NK(K+2) \tag{4.82}$$

The expectations are obtained from Taylor development of the characteristic functions and hence:

$$E\{MSE\} = E\left\{\frac{1}{N}\|f(y) - x\|^2\right\} = \frac{1}{N}\sum_{n=1}^{N}(E\{f_n(y)^2\} - 2E\{x_n f_n(y)\} + x_n^2) \tag{4.83}$$

A method was developed in [16] to estimate the expectation without a know value of x. This is achieved when $f(n)$ of the estimator $f(R^N)$is continuously differentiable.

$$\partial f(y) = \left[\frac{\partial}{\partial y_n} f_n(y)\right]_{1\le n\le N}, \partial^2 f(y) = \left[\frac{\partial^2}{\partial y_n^2} f_n(y)\right]_{1\le n\le N} \tag{4.84}$$

With this it was shown in [16] that:

$$E\{x^T f(y)\} = E\{(y - K.1)^T f(y)\} - 4E\left\{\left(y - \frac{K}{2}.1\right)^T \partial f(y)\right\} + 4E\{y^T \partial^2 f(y)\} \tag{4.85}$$

If $\boldsymbol{Z \sim X^2 k_{(x)}}$ with $f(z)$ satisfying regularity conditions then

$$MSE = \frac{1}{N}\left(\|f(y) - (y - K.1)\|^2 - 4^T\left(y - \frac{K}{2}.1\right)\right) + \frac{8}{N}\left(\left(y - \frac{K}{2}.1\right)^T \partial f(y) - y^T \partial^2 f(y)\right) \tag{4.86}$$

The MSE may be expressed as a sum of three terms

$$\|\mathrm{f(y)} - \mathrm{x}\|^2 = \underbrace{\|\mathrm{f(y)}\|^2}_{\text{term 1}} - \underbrace{2\mathrm{x^T f(y)}}_{\text{term 2}} + \underbrace{\|\mathrm{x}\|^2}_{\text{term 3}} \tag{4.87}$$

These can be replaced by statistical equivalents independent of x

$$\|x\|^2 = E\{\|y\|^2\} - 2(K+2)1^T E\{y\} + NK(K+2) \tag{4.88}$$

4.5.1Cure-optimized denoising via Unnormalised HAAR Wavelet Transform

The Cure optimized denoising with HAAR wavelet transform is described in figure 4.1.

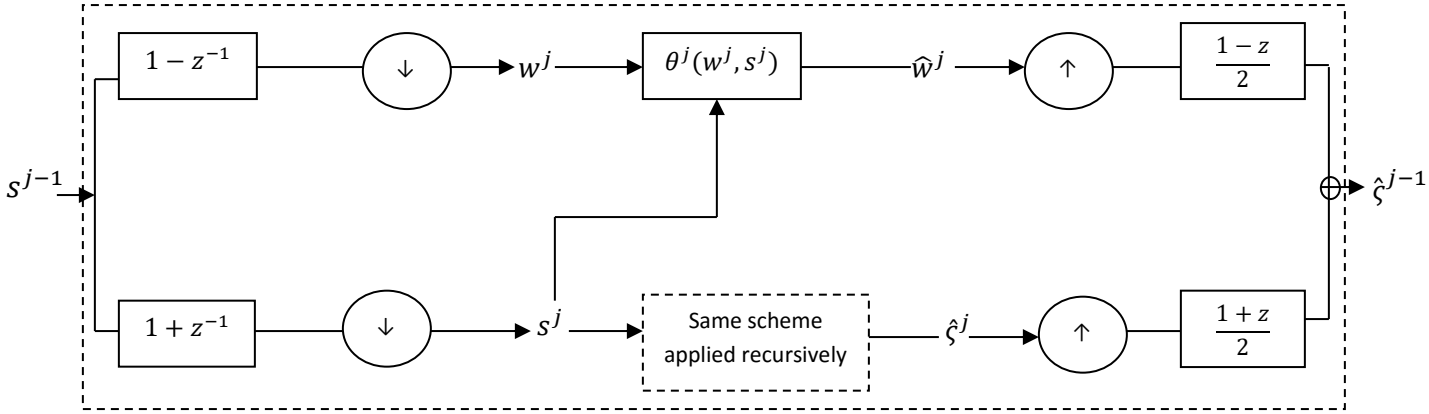


Figure 4.1: HAAR Wavelet Transform

It is possible with the use of un-normalized form to have sub-band dependents on estimation of MSE. The scaling and wavelet coefficients of the observed image can be given as:

$$s_n^j = s_{2n}^{j-1} + s_{2n-1}^{j-1}, \quad w_n^j = s_{2n}^{j-1} - s_{2n-1}^{j-1} \tag{4.89}$$

And those of un-normalized Haar scaling and wavelet coefficients of the non-centrality parameter as:

$$\varsigma_n^j = \varsigma_{2n}^{j-1} + \varsigma_{2n-1}^{j-1}, \quad w_n^j = \varsigma_{2n}^{j-1} - \varsigma_{2n-1}^{j-1} \tag{4.90}$$

It was shown in[16] that:

$$CURE_j = \frac{1}{N_j}\left(\|\theta(w,s) - w\|^2 - 4^T\left(s - \frac{K_j}{2}.1\right)\right) + \frac{8}{N_j}\left(\left(s - \frac{K_j}{2}.1\right)^T \partial_1\theta(w,s) + w^T\partial_2\theta(w,s)\right) - \frac{8}{N_j}\left(w^T\left(\partial_{11}^2\theta(w,s) + \partial_{22}^2\theta(w,s)\right) + 2s^T\partial_{12}^2\theta(w,s)\right) \tag{4.91}$$

Is unbiased estimate of the risk for sub-band. The estimate therefore may be formulated as:

$$E\{\|\theta(w,s) - \omega\|^2\} = E\{\|\theta(w,s)\|^2\} - \underbrace{2E\{\omega^T\theta(w,s)\}}_{\text{(I)}} + \underbrace{\|\omega\|^2}_{\text{(II)}} \tag{4.92}$$

The two expressions I and II are evaluated giving:

$$E\{\omega_n\theta_n(w,s)\}=E\{x_{2n}\theta_n(w,s)\}-E\{x_{2n-1}\theta_n(w,s)\}= E\left\{w_n\left(\theta_n 4\left(\partial_{11}^2\theta_n(w,s)+\right.\right.\right.$$

$$\left.\left.\left.\partial_{22}^2\theta_n(w,s)-\partial_2\theta_n(w,s)\right)\right)\right\}(w,s)+4E\{(s_n-K)\partial_1\theta_n(w,s)-2s_n\partial_{12}^2\theta_n(w,s)\} \quad (4.93)$$

and

$$\omega_n^2=E\{w_nw_n\}=E\{w_n^2-4(s_n-K)\} \quad (4.94)$$

Adopt uniform soft thresholding as:

$$\theta_n(w,s;a)=sign(w_n)max(|w_n|)-a\sqrt{s_n},0) \quad (4.95)$$

With linear expansion of threshold which decrease the artifacts further the resulting expression is:

$$\theta_n(w,s;a)=\sum_{k=1}^{2}a_k max\left(1-\lambda_k\frac{4\gamma_n(s)}{\gamma_n^2(w)},0\right)w_n+\sum_{k=1}^{2}a_{k+2}\,max\left(1-\lambda_k\frac{4\gamma_n(s)}{\gamma_n^2(p)},0\right)w_n+$$

$$\sum_{k=1}^{2}a_{k+4}\max\left(1-\lambda_k\frac{4\gamma_n(s)}{\gamma_n^2(w)},0\right)p_n+\sum_{k=1}^{2}a_{k+6}\,max\left(1-\lambda_k\frac{4\gamma_n(s)}{\gamma_n^2(p)},0\right)p_n \quad (4.96)$$

The optimal parameters is the solution to the equation $Ma=c$ where:

$$\begin{cases} c=\left[w^T\theta_k(w,s)-4\left(s-\frac{K_j}{2}.1\right)^T\partial_1\theta_k(w,s)+8s^T\partial_{12}^2\theta_k(w,s)+\right. \\ \qquad\left.4w^T\left(\partial_{11}^2\theta_k(w,s)+\partial_{22}^2\theta_k(w,s)-\partial_2\theta_k(w,s)\right)\right]_{1\le k\le 8} \\ M=[\theta_k(w,s)^T\theta_l(w,s)]_{1\le k,l\le 8} \end{cases} \quad (4.97)$$

The bias is removed from the low pass sub-band at a given scale *J*:

4.5.2 Magnitude Image

In magnitude magnetic resonance imaging, the acquired image is a composition of the magnitudes $\|m_n\|$ of *N* complex measurements m_n, where:

$$\begin{cases} \Re\{m_n\} & \sim & N(\Re\{\mu_n\},\sigma^2), \\ \Im\{m_n\} & \sim & \mathcal{N}(\Im\{\mu_n\},\sigma^2). \end{cases} \quad (4.98)$$

The task is to estimate the original signal from the measurements. Two *N* dimensional vectors are defined as:

$$\begin{cases} x= & [|\mu_n|^2/\sigma^2]_{1\le n\le N}\epsilon\mathbb{R}_+^N, \\ y= & [|m_n|^2/\sigma^2]_{1\le n\le N}\epsilon\mathbb{R}_+^N, \end{cases} \quad (4.99)$$

Denoising of the image is carried out as follows:

Step I: Estimate the noise variance σ^2

Step II: Rescale the squared-magnitude MR image y ;

Step III: Use a CURE-optimized algorithm for an estimate $\hat{x} = f(y)$ of x;

Fix some $\lambda \epsilon [0; 1]$ and obtain the final estimate $\hat{\mu}$ of the MR Image by using

$$\hat{\mu} = \sigma\left[\lambda\sqrt{|f_n(y)|} + (1-\lambda)\sqrt{max(f_n(y), 0)}\right]_{1\le n\le N} \tag{4.100}$$

A new measure of similarity may be defined as constant invariant PSNR(CIPSNR) is given by:

$$10_{\log 10} \frac{N\|\mu\|\infty^2}{\| \ (a*\hat{\mu}+b*)-\mu \ \|^2} \tag{4.101}$$

where

$$(a^*, b^*) = \ arg_{a,b}^{min}\|(a\mu + b) - \mu\|^2 \Leftrightarrow \begin{cases} a^* = \dfrac{N\,\mu^T\,\mu - 1^T\mu 1^T\hat{\mu}}{N\hat{\mu}^T\,\hat{\mu} - (1^T\,\hat{\mu})^2} \\ b^* = \dfrac{1^T\mu\,\hat{\mu}^T\hat{\mu} - \mu^T\hat{\mu}\,1^T\hat{\mu}}{N\hat{\mu}^T\hat{\mu} - (1^T\hat{\mu})^2} \end{cases}$$

Stand alone filters and other processes used for combinational filter systems include the bilateral filter, the non-local means filter, total variation filter,4th order filter, linear minimum minimization square error, Chi-square unbiased risk estimator, Poisson unbiased risk estimator and Steinbeck unbiased risk estimator wavelet transform based Wiener filter.

CHAPTER FIVE

MATERIALS, METHODS AND EXPERIMENTAL RESULTS

5.1 Combination Denoising Schemes

Various methods have been developed and tested for denoising MRI medical images. These include bilateral filtering, non-local means, phase error corrections and total variational methods. With appropriate methods of signal and noise estimation and optimizing parameter tuning each of these algorithms have improved their performance to near perfection. This is so when the objective measures of performance take care of some predefined aspects such as the SNR, contrast and the MSE. However, when more specific measures are developed to take care of other characteristics that have become more relevant with advancement of diagnostic methods, it becomes apparent that more innovative methods need to be developed.

In this research, the attention has been on the use of combinational filters and denoising mechanisms where each stage or filter optimized a given characteristic without necessarily introducing degradation on other parameters. It also enables finer tuning of parameters that determine the effectiveness of image processing.

Four different combinational algorithms have been developed and each compared to the best performing algorithms for stand alone or other reported combinational filters results. These are the wavelet based Haar denoising method using non-local means filter and bilateral enhancement, A wavelet based MRI denoising method using LMMSE estimation and bilateral filter enhancement. Others are a total variational wavelet based structural MRI denoising with bilateral feature enhancement and a Haar wavelet magnetic resonance image denoising with optimized chi-square Rician estimation and bilateral filter enhancement.

5.2 Parameter Selection for the Bilateral Filter

The bilateral filter is used in all the combinational filters. The performance depends on how well its parameters are tuned (optimized) to the noisy image characteristics as well as to the statistical parameters.

There are two parameters that determine the behaviour of this filter. These are the parameter σ_d that characterizes the spatial domain behaviour and σ_r for the intensity domain behaviour. The optimizations of these parameters as functions of noise are obtained empirically.

Experiments involve adding the zero-mean Rician noise to MRI test images for different values of σ_d and σ_r. This is repeated for different noise variances and the mean square error for each is then noted. On examining the variation of MSE with σ_d and σ_r. It is found that an optimal σ_d is relatively insensitive to noise compared to σ_r [4].

It is also confirmed that σ_d and σ_n are linearly related. The least squares fits to σ_r / σ_n shows that σ_d is between 3 and 5 [4].

5.3 Parameter Selection for the non-local means Filter

Similarity of pixels is done with more precision by using regional comparison instead of pixel comparison. The pixels far away from the pixel to be are also given similar weight to the neighbouring pixels [6].

For an image *I* the filtered intensity at a pixel *p* is computed as a weighted average in search area.

$$NLM(\gamma(P)) = \sum_{\gamma(p)\in\Omega} W(N_p, N_p) I_p \tag{5.1}$$

Where

$$0 \le W(p.q) \le 1$$

$$\sum_{\gamma q=\Omega} W(N_p, N_q) = 1$$

N_i is a window around pixel *I* with radius $R\sin\Omega$, where R is radius of search.

The similarity is denoted as:

$$W(N_p, N_q) = \frac{1}{Z(p)} \ell^{-d}(Np, Nq) / L^2 \tag{5.2}$$

$$Z(p) = \sum_{\nu qE\gamma} \ell^{-ds}(Np, Nq) / L^2 \tag{5.3}$$

L is an exponential decay parameter that control parameter d which is a Gaussian weighted squared Euclidian distance, given by:

$$d(N_p, N_q) = \left\| (Y(N_p) - Y(N_q)) R \sin \Omega \right\| \tag{5.4}$$

For $p = q$ there could be overweighting

A computation to determine:

$D(N_p, N_q)$ the minimum distance of the other pixels is given by:

$$d(N_p, N_q) Min(d(N_p, N_q), q \neq p \tag{5.5}$$

A way of pre-selection to reduce the computations and to enhance filtering results is considering only similar pixels.

Select the pixels that do not have the same first local moment (mean value of a 3 x 3 image patch) which should be less than $k\sigma_i / \sqrt{n}$ no of significant pixels.

k is set to 3 for a single image corresponding to the third quartile of distribution

$$W\left(N_p N_q\right) = \left\{ \frac{1}{N_{(p)}} \ell^{-d(p,q)^*}, if\left(\left| \mu_{pi} - \mu_{qi} \right| < k\sigma_i / \sqrt{n} \right) v_i \in [1, C) \right\} \tag{5.6}$$

with

$$d(p,q)^* = \max \frac{\left(\sum_{i=1}^{C} d\left(N_p, N_q^i\right) \right)}{c} / h^{i^2} - 1{,}0 \tag{5.7}$$

5.4 Parameter selection in CURE and CURELET

The estimate has been shown to be equal to [16]:

$$\text{CURE} = \frac{1}{Nj} \left(\left\| \mathcal{G}(w,s) - w \right\|^2 - 4^T \left(S - \frac{kj}{2} - 1) \right) \right)$$

$$+\frac{8}{Nj}\left(\left\| s-\frac{kj}{2}.\,1\right)^{T}\partial,\vartheta(W,S)+W^{T}\partial_{2}\vartheta(W,S)\quad\right\|\Big)$$

$$-\frac{8}{nJ}\left(w^{T}\left(\partial_{11}^{2}\vartheta\left(w,s\right)+\partial_{22}^{2}\vartheta(w,s)+2S^{T}8_{12}^{2}\vartheta(w,s)\varepsilon\{CURE;\}=\varepsilon\{MSE;\}\right.\right.\text{ at sub } b \text{ and } j \quad (5.8)$$

Rician distributed noise is signal dependent therefore threshold is required.

'Uniform' soft threshold is adapted as:

$$\vartheta n(w,s;a)=Sign\ (w_n)\,max(w_{al}-a\sqrt{S_n},0 \quad (5.9)$$

a is selected to yield minimum cure. A continuously differentiable approximation of soft thresholding is used. A linear estimation of threshold is expressed as follows:

$$\vartheta_n\left(w,s,a\right)=\sum_{k=1}^{2}a_k\ \max\left(1-\lambda_k\frac{4\gamma_n\left(s\right)}{\gamma_n^2\left(w\right)},0\right)w_nT\sum_{k=1}^{2}a_k+2\max\left(1-\lambda\ k\frac{4\lambda_n\left(s\right)}{\gamma^2n\left(p\right)},0\right)w_n \quad (5.10)$$

$$\vartheta_n\left(p,s,a\right)=\sum_{k=1}^{2}a_k\ \max\left(1-\lambda_k\frac{4\gamma_n\left(s\right)}{\gamma_n^2\left(e\right)},0\right)p_n+\sum_{k=1}^{2}a_k\ \max\left(1-\frac{\lambda_k4\gamma_n\left(s\right)}{\gamma_n^2\left(p\right)'0},\right)p_n \quad (5.11)$$

The function $\gamma_n(u)=\frac{1}{2\sqrt{\pi}}\sum_k\left|\mu k\right|\ell^{-(n-k)^2\!/_z}$ (5.12)

implements a Gaussian smoothing this local filtering takes care of similarities between neighbourhood wavelet, scaling and parent coefficients.

The function therefore considers the inter as well as intra-scale dependencies that are presentt in a Haar wavelet.

There are eight parameters optimized via least squares. Parameters $\lambda = 1$ and $\lambda = a$ can be fixed in advance without loss in denoising quality.

Considering that all wavelet coefficients in a sub-band are filtered, the approximation becomes:

$$\vartheta_n(w,s,a)=\sum_{k=1}^{8}a_k\vartheta_k(w,s) \tag{5.13}$$

The optimal set is therefore the solution of *M*a = *C*

$$\text{Where C}=\left[w^T\vartheta_k(w,s)-4(s-\frac{kj}{2}.1\right]\partial^T\vartheta_k(w,s)+\partial AS^T\partial_{12}^2\vartheta_K(w,s))$$

$$+4W\left(\partial_{11}^2\right)'\vartheta k(w,s)+\partial_{22}^2\vartheta(w,s)-\partial_2\vartheta_k(w,s))]1\le k\le 8 \tag{5.14}$$

$$\text{M}=\left[\vartheta_K(w,s)^T\vartheta(w,s)\right]_1\le k,1\le 8 \tag{5.15}$$

The bias is removed from loss pass residual sub-band at scale *J* as $\hat{G}^J=S^J-2^JK.1$

5.5 Conventional Measures of Quality

These include Mean Square Error (MSE), Signal to Noise Ratio (SNR) which shows the value of removing noise and Peak Signal to Noise Ratio (PSNR) used to measure the difference of two images, are global measures of objectives function that will be used to test the success and efficiency of the chosen approach.

$$MSE=10\log_{10}\left\{\frac{1}{N^2(x,y)}\right\}\sum_{n=1}^{N}\left[f(x,y)-\hat{f}(x,y)\right]^2 \tag{5.16}$$

$$SNR=10\log_{10}\left\{\frac{\sum_{n=1}^{N}(x,y)[f(x,y)]^2}{\sum_{n=1}^{N}(x,y)\left[f(x,y)-\hat{f}(x,y)\right]^2}\right\} \tag{5.17}$$

For an *M* ×*N* image where *f (x,y)* and $\hat{f}$*(x,y)* are the original and restored image respectively while;

$$PSNR=\left(\frac{2}{RMS}\right) \tag{5.18}$$

Where n is bits for each pixel in the input image and

$$RMSE = \sqrt{\frac{\sum\left[f\left(x,\, y\right) - \hat{f}\left(x, y\right)\right]^2}{MN}} \tag{5.19}$$

Where M and N are the dimensions the input image matrix

$f(x, y)$ is the clean image and $\hat{f}(x, y)$ is the resulting image.

Also subjective tests through visual inspection of the original image and the restored image have been performed on the test images for comparative purposes.

5.6 Experimental Procedure

Algorithms have been developed to implement adaptive combination wavelet based filters. Various MR images have been denoised using these algorithms and their output images analyzed analytically and visually.

5.7 Method I: A hybrid and adaptive MRI denoising method involving a bilateral filter enhancement and Non-local Means wavelet based method

A bilateral filter is used to diffuse image discontinuities with one image being denoised and the other with all features preserved by edge enhancement. The bilateral filter uses two sets of parameters for the two effects. Wavelet decomposition into various sub-bands at different levels is applied to each of the two images. The Non-local means filtering is carried out at various sub-bands with appropriate coefficients being computed from original and noisy image statistics. Finally a bilateral filter is used for further enhancement. The input to the filter is either magnitude or square noisy images obtained by adding Rician noise at different levels to a relatively clean image. The parameters w, s_d and s_r of bilateral filters are varied over a large number of values as there are no distinct rules that can guide the tuning of these parameters.

5.8 Proposed Approach for method I

The algorithm for the proposed method is shown in Figure 5.1. The image is taken through three different bilateral filters each with a different parameter and then through non-local means filter before wavelength decomposition.

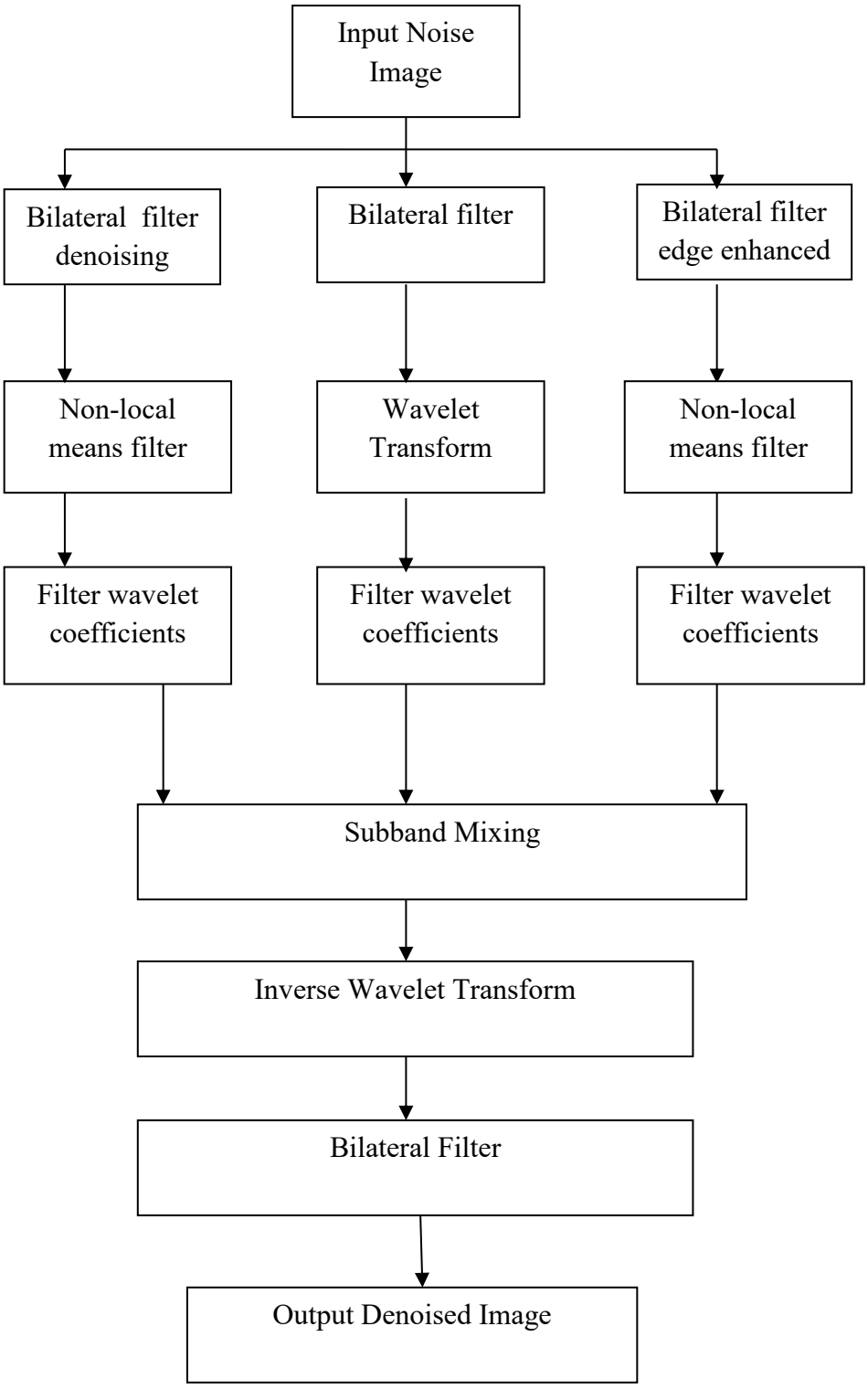


Figure 5.1: A Flowchart of the Proposed Method

5.9 Measures of Quality

To assess the effectiveness of the denoising techniques various measures of quality are used such as the MSE and PSNR. These are the first choice but they do not give sufficient information about visual quality. To evaluate image visual quality, additional measures are used. These measures are defined as follows:

5.9.1 Signal to Noise Ratio

This is a measure of signal purity defined as**:**

$$SNR = 10log_{10}\left(\frac{\sum_{x_i \epsilon \Omega} I(x_i)^2}{\sum_{x_i \epsilon \Omega}(I(x_i)-\hat{I}(x_i))^2}\right) \tag{5.20}$$

5.9.2 Mean Square Error

The MSE on gradient is given by

$$\text{MSE} = \frac{1}{|\Omega|}\sum_{x_i \epsilon \Omega}\left(\Delta I(x_i) - \Delta\, \hat{I}\,(x_i)\right)^2 \tag{5.21}$$

This metric is based on the "ground truth" image I and the denoised image$\hat{I}$.

5.9.3 Correlation Coefficient

Another useful measure used is the corrélation coefficient defined as follows:

$$CoC = \frac{\sum_{x_i \epsilon \Omega}(I(x_i)-\bar{I})\left(\hat{I}(x_i)-\bar{\hat{I}}\right)}{\sqrt{\sum_{x_i \epsilon \Omega}(I(x_i)-\bar{I})^2}\sum_{x_i \epsilon \Omega}\left(\hat{I}(x_i)-\bar{\hat{I}}\right)^2} \tag{5.22}$$

5.9.4Edge Preservation Index

This is a numerical measure of edge profile consistency which is formulated as follows:

$$EPI = \frac{\sum_{x_i \epsilon \Omega}(\nabla I(x_i)-\overline{\nabla I})\left(\nabla \hat{I}(x_i)-\overline{\nabla \hat{I}}\right)}{\sqrt{\sum_{x_i \epsilon \Omega}(\nabla I(x_i)-\nabla \bar{I})^2}\sum_{x_i \epsilon \Omega}\left(\nabla \hat{I}(x_i)-\nabla \bar{\hat{I}}\right)^2} \tag{5.23}$$

5.9.5 Universal Quality Index

The Universal Quality Index (UQI), is defined as a product of three factors: loss of correlation, luminance distortion, and contrast distortion as:

$$UQI = \frac{\sigma_{I\hat{I}}}{\sigma_I \sigma_{\hat{I}}} \cdot \frac{2\bar{I}\bar{\hat{I}}}{(\bar{I})^2+\left(\bar{\hat{I}}\right)^2} \cdot \frac{^2\sigma_I \sigma_{\hat{I}}}{\sigma_1^2+\sigma_{\hat{1}}^2} \tag{5.24}$$

5.9.6 The Structural Similarity Index Measure (SSIM)

This is a measure of closeness in structure of two images. It is defined as:

$$SSIM = \frac{(2\bar{I}\hat{\bar{I}}+C_1)(2\sigma_{I\hat{I}}+C_2)}{\left((\bar{I})^2+(\hat{\bar{I}})^2+C_1\right)(\sigma_I^2+\sigma_{\hat{I}}^2+C_2)} \quad (5.25)$$

where $\bar{I}$ and $\hat{\bar{I}}$ are the means of the noise-free and the denoised image and C_1,C_2 are correlations respectively.

These measures of quality require data on the noisy and noiseless image as well as images gradient for computation of edge preservation index and mean square error on gradient. Appropriate scaling is required so that truncation of high square values of intensity is avoided.

5.10 Experimental Results

The image data used in the experimental investigations have been sourced from three public repositories namely; BrainWeb [47], Luisier [48] and Clunie [49]. These are the main libraries for MRI data and have been used by many investigators due to their ease of availability. These images were acquired using a Siemens Vision scanner echo-planar imaging system with a 1.5T magnetic field. Other parameters of the scanner being field of view (FOV) = 240mm x 240mm, relaxation time (TR) =2000ms, echo time (TE) = flip angle (FA) = 90^0, for 256x256 image sizes. To compare the effectiveness of the adaptive combination non-local means filter to the stand alone state of the art denoising methods, Rician noise at various levels from 2% to 20% is added to noise free magnetic resonance image. The range is chosen because acquired images noise would be in this range. The images used include *Torso1, Torso2* and *Hip* which are structural 2D images.

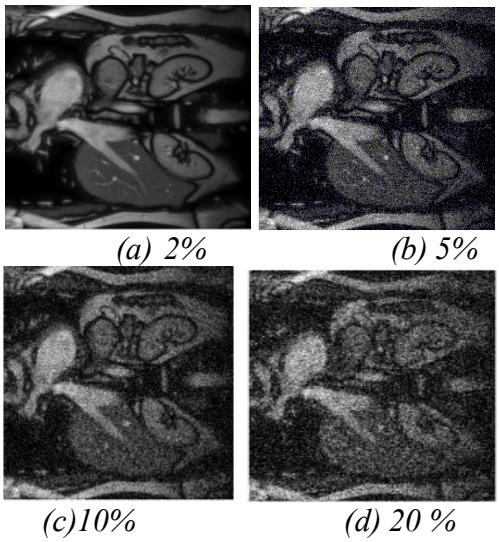
5.10.1 Effect of Noise Addition

The visual effect of noise addition was tested with three structural 2Dimensional MR images taken from the torso and the hip. This image has been referred to as *torso1, torso2* and *hip*. These three images are displayed in Figure 5.2 priori to Rician noise addition. These images have features that are very clear and the edges are also relatively distinct. They also have a wide intensity range from completely dark to complete white that is in the pixel intensity range (0 to 256) for an 8-bit system. They also exhibit high and low frequency regions. For the purpose of algorithm testing, the images have been re-sized from various sizes to 256x256.

(a) (b) (c)

Figure 5.2: Noise-free MRI images (a) Torso1, (b) Torso2, (c) Hip

The effect of addition of Rician noise on the visual quality of the images is illustrated in Fig 5.3(a), (b), (c) and (d) in regard to the image *torso1*. The level of noise in the image was varied from 2% to 5% to 10% up to 20%. It can be noted that for a low level of noise degradation, the image edges are still very clear but contrast can been seen to have been reduced slightly. This could affect the interpretation of small features in the image during diagnosis; the same applies to 5% though the contrast is worse. At 10% it is difficult to distinguish the boundary of some tissue and the random noise is clearly visible. At 20% only the major features are recognizable with minute features buried in the random noise. Similar observations were made with the *torso 2* and *hip* images.



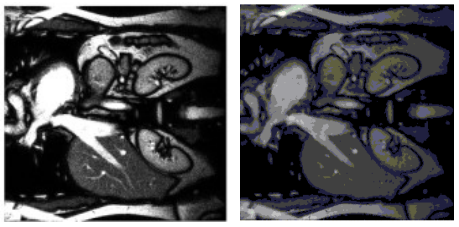
(a) 2% *(b) 5%*

(c)10% *(d) 20 %*

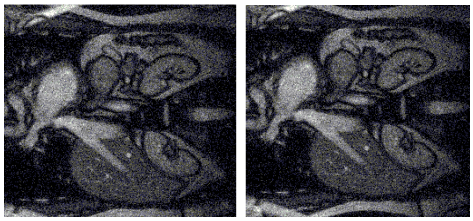
Figure5.3: Noisy images of Torsol1 at given noise levels

5.10.2 Denoising using Wavelet Thresholding, Median, Bilateral and Non-local methods

The results of employing the stand alone methods in noise suppression is illustrated in Figure 5.4.This is without taking advantage of the contribution of each filter in the combination. Using the bilateral alone may lead to over smoothing hence obscuring fine details; using wavelet thresholding on its own also leads to loss of fine details. Figure 5.4(a) shows extreme whitening and darkening when thresholding is used. This is because below a given intensity value all signal is assumed to be noise and truncated and the bias in the magnitude image takes intensity values above a certain value to maximum. This upper threshold is the difference between the maximum value and the bias. In Figure 5.4 (b) it can be observed that the bilateral filter and median filter when used each on its own lead to smoothing of image details. Figure 5.4 (c) shows that the non-local filter on its own gives a relatively better image but the contrast and the boundaries clarity needs further improvement.



(a) Thresholding *(b) Median*



(c)Non-local means (d) Bilateral

Figure 5.4: Denoised Images of Torso1using other Methods

5.10.3 Denoising using the Proposed Method

The proposed method is shown in figure 5.1 where a bilateral filer is used to preserve edges before the main method; Non-local means with soft wavelet thresholding is applied. Figure 5.5 shows the result of using the proposed algorithm. At 2% and 5% noise addition, the recovery process is very efficient as can be noted from a visual inspection of the images. For example, at 2% noise addition the recovered image is seen to be almost the same as the noiseless image. The observations are also analyzed by employing the mathematical quality

measures described in section 3 and given in table 5.1, 5.2, 5.3 and 5.4. For example in Table 5.1 and Table 5.2 where *SNR* is very high, *MSE* relatively low, UQI almost 1, SSIM 0.984 and EPI 0.89 which is an improvement from 0.70 of the noisy image. It also shows that edge preservation is very sensitive even for low noise. The use of bilateral filter in this combination algorithm has been shown to be the main contributor for edge preservation.

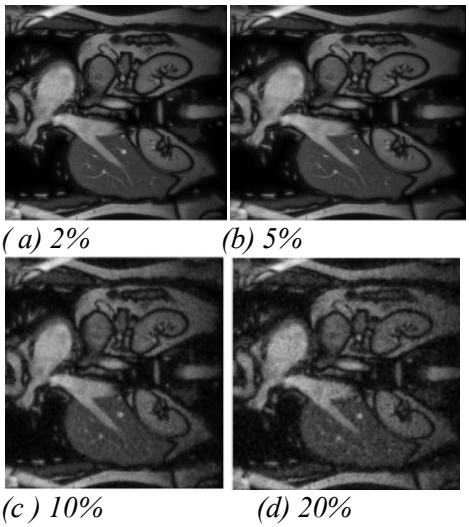
The results obtained when the five measures of quality are used to assess the effectiveness of image *torso1* denoising using the proposed method for relatively low noise levels are shown in Table 5.1 and Table 5.2

Table 5.1: Quality Measures at 2% noise for combinational Non local means algorithm

Noise Filter type	PSNR (dB)	RMSE	UQI	SSIM	EPI
Noisy image	38.04	16.23	0.96	0.86	0.70
Median filter	41.23	13.55	0.97	0.89	0.75
Wiener filter	42.38	11.12	0.97	0.97	0.74
Bilateral filter	44.02	8.654	0.99	0.98	0.85
Thresholding	38.54	22.43	0.95	0.93	0.68
NLM filter	43.54	8.54	0.98	0.90	0.82
Combination filter	**44.43**	**6.45**	**0.99**	**0.98**	**0.89**

Table 5.2: Quality Measures at 5% noise for combinational Non local means algorithm

Noise Filter type	PSNR (dB)	RMSE	UQI	SSIM	EPI
Noisy image	35.02	23.53	0.95	0.84	0.62
Median filter	37.64	18.45	0.98	0.89	0.72
Wiener filter	38.56	12.34.	0.99	0.95	0.71
Bilateral filter	41.23	8.654	0.98	0.95	0.80
Thresholding	36.54	34.12	0.87	0.84	0.71
NLM filter	39.54	14.23	0.99	0.95	0.82
Combination filter	**42.38**	**12.55**	**0.99**	**0.95**	**0.84**



(a) 2% *(b) 5%*

(c) 10% *(d) 20%*

Figure 5.5: Denoised images of Torsol1 using combinational Non local algorithm

The quality measures have also been used to give an assessment of the simulation results for 10% and 20% noise levels.

Table 5.3: Quality Measures at 10% noise for combinational Non local algorithm

Noise Filter type	PSNR (dB)	RMSE	UQI	SSIM	EPI
Noisy image	29.5	35.45	0.80	0.74	0.48
Median filter	31.21	25.34	0.87	0.84	0.57
Wiener filter	33.25	19.95	0.97	0.87	0.51
Bilateral filter	34.53	16.78	0.98	0.89	0.60
Thresholding	28.51	45.67	0.95	0.84	0.52
NLM filter	36.45	17.76	0.98	0.89	0.68
Combination filter	**37.12**	**15.23**	**0.99**	**0.89**	**0.69**

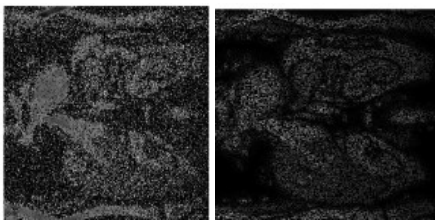
Table 5.4: Quality Measures at 20% noise for combinational Non local algorithm

Noise Filter type	PSNR (dB)	RMSE	UQI	SSIM	EPI
Noisy image	23.45	46.72	0.85	0.65	0.34
Median filter	24.20	30.94	095	0.72	0.37
Wiener filter	26.24	25.43	0.96	0.73	0.44
Bilateral filter	27.54	22.69	0.96	0.80	0.41
Thresholding	23.21	57.31	0.95	0.73	0.38
NLM filter	26.78	23.47	0.96	0.88	0.45
Combination filter	**28.69**	**19.23**	**0.97**	**0.88**	**0.47**

Tables 5.1 to Tables 5.4 show that the adaptive non-local means filter outperform other methods like Thresholding and Wiener filtering alone when used with multi-resolution wavelet soft thresholding. The inclusion of bilateral filter contributes to overall effectiveness not only on edge preservation but also on SSIM and MSE. In Table 5.4, where the added noise is 20%, the quality measures show that the efficiency and effectiveness of all the denoising methods is relatively low. It is therefore necessary that a relatively clear image is reconstructed during the acquisition process.

5.10.4 Residue noise

Figure 5.6 shows the noise image which is the difference between noiseless (reference) image and noisy image. Result show that noise is signal dependent and even after denoising the slight residual noise has a remote relationship to the image.



(a) (b)

Figure 5.6 Residual noise , (a) Before denoising (b) After denoising

5.11 Result analysis and discussion

The denoising results obtained for each of the tested images using the combinational adaptive NLM filters and those of the standalone filters have been obtained by computer simulation and given in figure 5.4 and figure 5.5 and in tables 5.1, 5.2, 5.3, and 5.4. From these figures, it is clear that the proposed combinational algorithm results in a higher-contrast denoising and better visual clarity as compared to the stand-alone methods. The proposed method also suppresses image noise to a higher level that the other methods. The PSNR, RMSE, UQI, SSIM and EPI quality measures have been used to provide a comparison of the noise reduction levels by the stand alone filters and the proposed method. It has been shown that using the proposed method there is some significant improvement over the other methods in image quality while exhibiting higher effective resolution and contrast. The results obtained

in a high noise regime indicate poor recovery of the original image. This has been evidenced by our simulations as given in fig 5.5 (c) and in tables 5.3 and 5.4 where at least 10% noise addition was used. The solution lies in the improvement of the measurement and calibration system of MRI equipment. This may be achieved by use of more gradients, use of parallel acquisition methods, slice time correction and motion correction.

5.12 Method II: Wavelet Hybrid MRI Denoising Scheme using Chi Square unbiased Risk Estimate with Bilateral Filter Preprocessing and Enhancement

5.12.1 Proposed Approach

A new combinational filter consisting of a bilateral filter and wavelet thresholding has been developed. The filter algorithm is implemented in MATLAB and the performance of the model is compared with median filter, Wiener filter, stand alone bilateral filter and wavelet thresholding. In this method a chi-square unbiased risk estimation is applied to MRI using square image. The Rician distribution of noise is used to model the image noise for purposes of not only denoising but also bias removal. The bilateral filter utilizing both spatial and amplitude distances is used to better preserve image structural and other details in a suitable manner.

The algorithm is shown in the figure 5.7. The bilateral filter parameters are set before wavelet transform process is applied. In one case they are set to enhance boundaries and the other case for block discontinuity identification and correction and texture map characterization. This involves filtering the input image with[-1,0,1] and [-1,0,]T which detect both vertical and horizontal boundaries and any discontinuities. To elminate blockiness the bilateral filter is applied to the entire block and diffuse the boundaries into the block. One approach to this diffusion is to set the centre four pixels to zero except the four corner pixels and large values of the discontiuinties are maintained and other values interpolated lineary. Repeating this for all the blocks, block discontinuity map is $M_b(x)$ is obtained from which σ_r is obtained.

$$\sigma_r(x) = \max(\sigma_{r,min}, k_0 M_b(x)) \tag{5.26}$$

Where k_0 is a scaling factor

The standard deviation of each block is computed and is used to state the texture and adjust the σ_d value for preserving the texture information. This enables texture detection and edge regions.To remove the edge regions the bilateral filter is applied with a large σ_d. A 3x3 median filter may be used on 8x8 blocks and resulting image is interpolated and texture map M(t) detected which is used to compute σ_d as follow:

$$\theta_d(x) = \max(\theta_{d,\min}, \frac{k_2}{1+Mt(x)}) \tag{5.27}$$

The wavelet decomposition is carried out to level 4 for the Haar wavelet, which has been found to be sufficient after 8 trials. Cure Unbiased Risk Estimation (CURE) is the main pocess of obtaining the estimates of the wavelet coefficients which will lead to reconstitution of a denoised image after a thresholding level is identified along one part and the edges enhanced in the second path. Before sub-band mixing the bias is removed by subtracting the last component of the noisy image for each pixel (intensiy level) using the scaling coefficient. Sub-band mixing involves taking the filtered subbands and reconstring usin the two dimensional wavelet trasform. After which the final stage of the bilateral filtering is used in this case with optimal parametres for artifact removal, increased contrast and structural details enhancement.

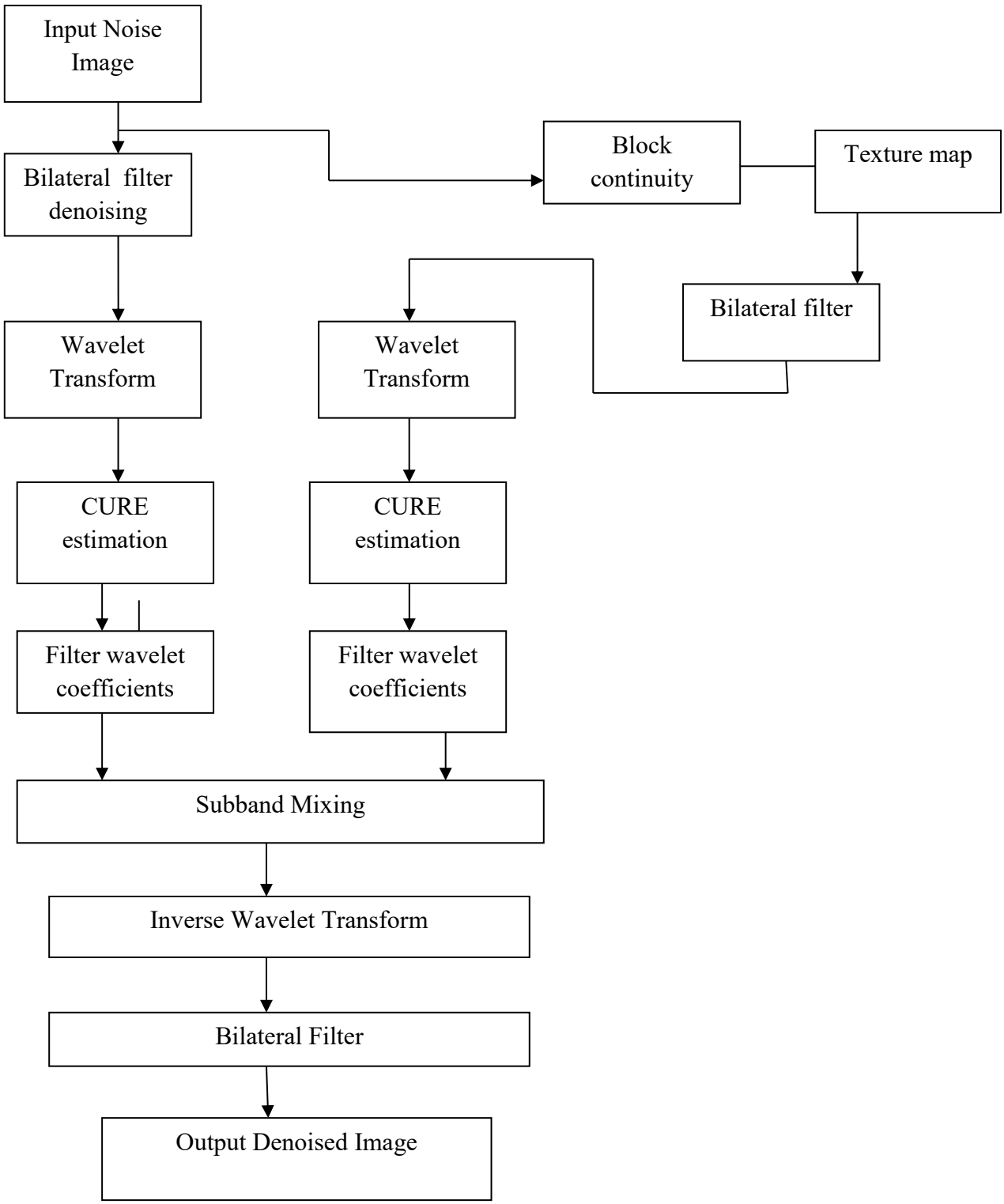
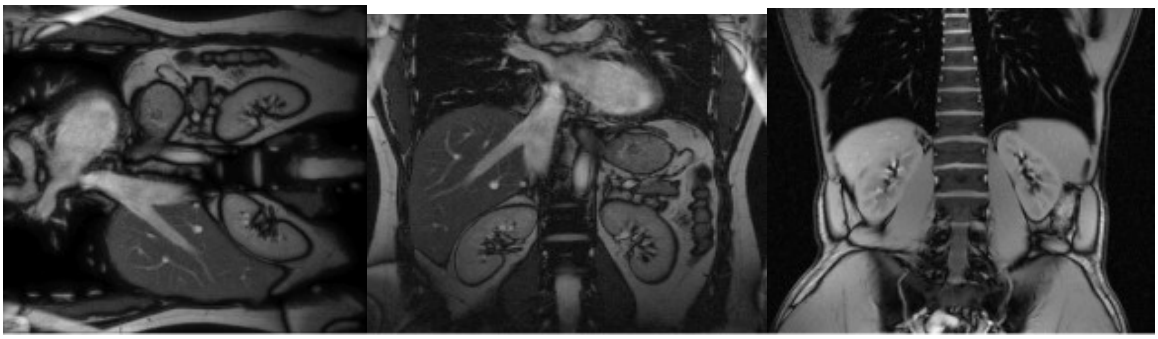


Figure 5.7: Algorithm for Chi-square method

The input to the filter is either a magnitude or a square noisy image obtained by adding Rician noise at different levels to a relative clean image. Some of the images used are shown in figure 5.8, 5.9 and figure 5.10.



Torso 1 *Torso 2* *Hip*

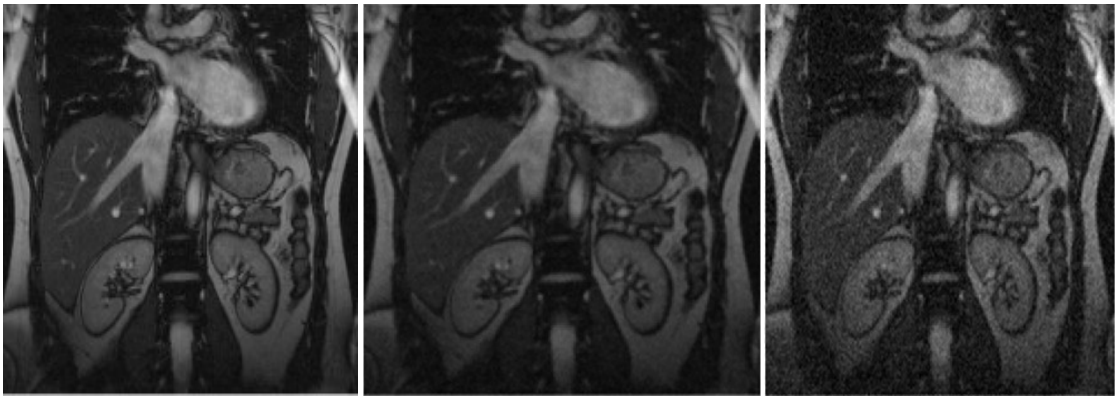
Figure 5.8: Relatively clean MRI images

2% **5%** **10%** **20%**

Figure5.9: Noisy Images of Torso2 at various levels

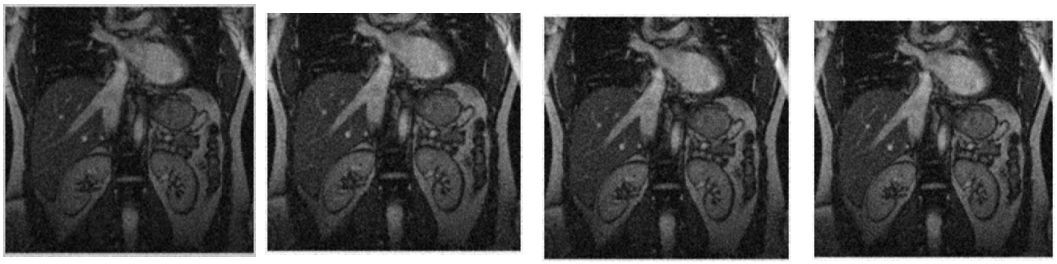
Figure 5.10: Residual noise

The resulting noise images using the new algorithm are shown in figure 5.11 and those from the state of arts algorithms in figure 5.12.



5% 10% 20%

Figure 5.11: Denoised images of Torsol2 using combinational Chi-square algorithm



Wiener Thresholding Bilateral Nowak

Figure 5.12: Denoised images of Torsol2 using other methods

Results of Experiments

The detailed resulting image statistics are shown in table 5.5, 5.6, 5.7 and 5.8.

Table 5.5: Quality measures at 2% for combinational Chi-square algorithm

Noise Filter type	SNR	PSNR (dB)	RMSE
Noisy image	24.56	35.02	23.53
Median filter	36.54	37.642	18.45
Wiener filter	27.92	38.56	12.34.
Bilateral filter	28 .67	41.23	8.654
Thresholding	34.21	36.54	34.12
Nowak filter	26.87	39.54	14.23
Combination filter	29.34	41.65	9.54

Table 5.6: Quality measures at 5% for combinational Chi-square algorithm

noise Filter type	SNR	PSNR(dB)	RMSE
Noisy image	24.56	35.02	23.53
Median filter	36.54	37.642	18.45
Wiener filter	27.92	38.56	12.34.
Bilateral filter	28 .67	41.23	8.654
Thresholding	34.21	36.54	34.12
Nowak filter	26.87	39.54	14.23
Combination filter	29.34	41.65	9.54

Table 5.7: Quality measures at 10% noise for combinational Chi-square algorithm

Filter type	SNR	PSNR(dB)	RMSE
Noisy image	21.45	29.5	35.45
Median filter	22.54	31.21	25.34
Wiener filter	24.28	33.25	19.95
Bilateral filter	25.69	34.53	16.78
Thresholding	20.32	28.51	45.67
Nowak filter	26.54	36.45	22.76
Combination filter	26.72	37.12	15.23

Table 5.8: Quality measures at 20% noise for combinational Chi-square algorithm

Filter type	SNR	PSNR(dB)	RMSE
Noisy image	21.45	29.5	35.45
Median filter	22.54	31.21	25.34
Wiener filter	24.28	33.25	19.95
Bilateral filter	25.69	34.53	16.78
Thresholding	20.32	28.51	45.67
Nowak filter	26.54	36.45	22.76
Combination filter	26.72	37.12	15.23

5.12.2 Result Analysis and Discussion

From the analysis of the test results, the proposed method performs better in image denoising than other methods both in terms of visual inspection and use of performance measures. Both median and non-optimized thresholding performed poorly in comparison with other methods. Wavelet based Wiener filter performed relatively better and there was marked improvement when bilateral filter and Nowak filters are used. The combination algorithm developed in this thesis outperforms all these filters. It is able to recover much more detail from the noisy image. Analysis of MSE of the test result show that residual noise of the new filter is least at 15.23 compared to 19.95 for Wiener filter and 22.76 for Nowak filter when the initial noise level was set at 20%.

5.13 Method III: An LMMSE diffusion weighted MRI Image Denoising Wavelet based Algorithm with bilateral feature Enhancement

5.13.1 Proposed Method

In this method a linear minimum mean square error estimation of the original image is realized by a random variable related to the observed image. This estimate is further subjected to wavelet decomposition and thresholding in which case any signal bias is removed and coefficients under a certain threshold are removed as they contain more noise than signal components. A bilateral filter process is used to enhance the image edges so as to preserve details and remove any artifacts. This method consists of a new combinational filter consisting of bilateral filter and linear minimum mean square estimation. The algorithm for this method is shown in figure 5.13.

5.13.2 Selection of Parameters in LMMSE

In LMMSE, a closed form value of the estimator ensures that the method is computationally more efficient than the optimization solutions [3],

The estimator parameter is:

$$\hat{\vartheta}_r = E\{\theta\} + C_{OX} C_{XX}^{-1} C_X - E\{X\}) . \tag{5.28}$$

Where

C_{0x} is the cross covariance

C_{xx} is the covariance matrix

$$\hat{A_{ij}^2} = E\left\{A_{IJ}^2\right\} + CA_{IJ}^2 M_{IJ}^2 C^{-1} M_{IJ}^2 M_{IJ}^2 (M_{ih}^2 - E\left\{M_{ij}^2\right\}) \ . \tag{5.29}$$

Where M_{ij} is the amplitude of the pixel (*i,j*) in a 2D MR image

A^2 the even order moments in a Rician distribution are non-complex polynomials and therefore easier to determine. The covariance matrices are scalar values of the pixels.

In Rician distribution, the LMMSE estimator for 2D case is:

$$\hat{A_{ij}^2} = \frac{E\left\{A_{IJ}^2\right\}E\left\{A_{IJ}^4\right\} + 2E\left\{A_{IJ}^2\right\}\sigma_n^2 - E\left\{A_{IJ}^2\right\}E\left\{M_{IJ}^2\right\}x}{E\left\{M_{IJ}^4\right\} - E\left\{M_{ij}^2\right\}^2} \ x\left(M_{ij}^2 - E\left\{M_{ij}\right\}\right) \tag{5.30}$$

The noise variance normally computed from the image data

Two possibilities based on local statistics are used [3]

In case of background, then

$$\sigma_n^2 = \sqrt{\frac{2}{\pi}} \ \text{mode}\left\{\sigma_i^2, i, j, k\right\} \tag{5.31}$$

If not local variance is used, which is given by:

$$\sigma_n^2 = \text{mode}\left\{\sigma_i^2, i, j, k\right\} \tag{5.32}$$

The size of neighbourhood is parameter that also needs selection which uses the mode of the distribution hence its effects minimized typical value (5 x 5).

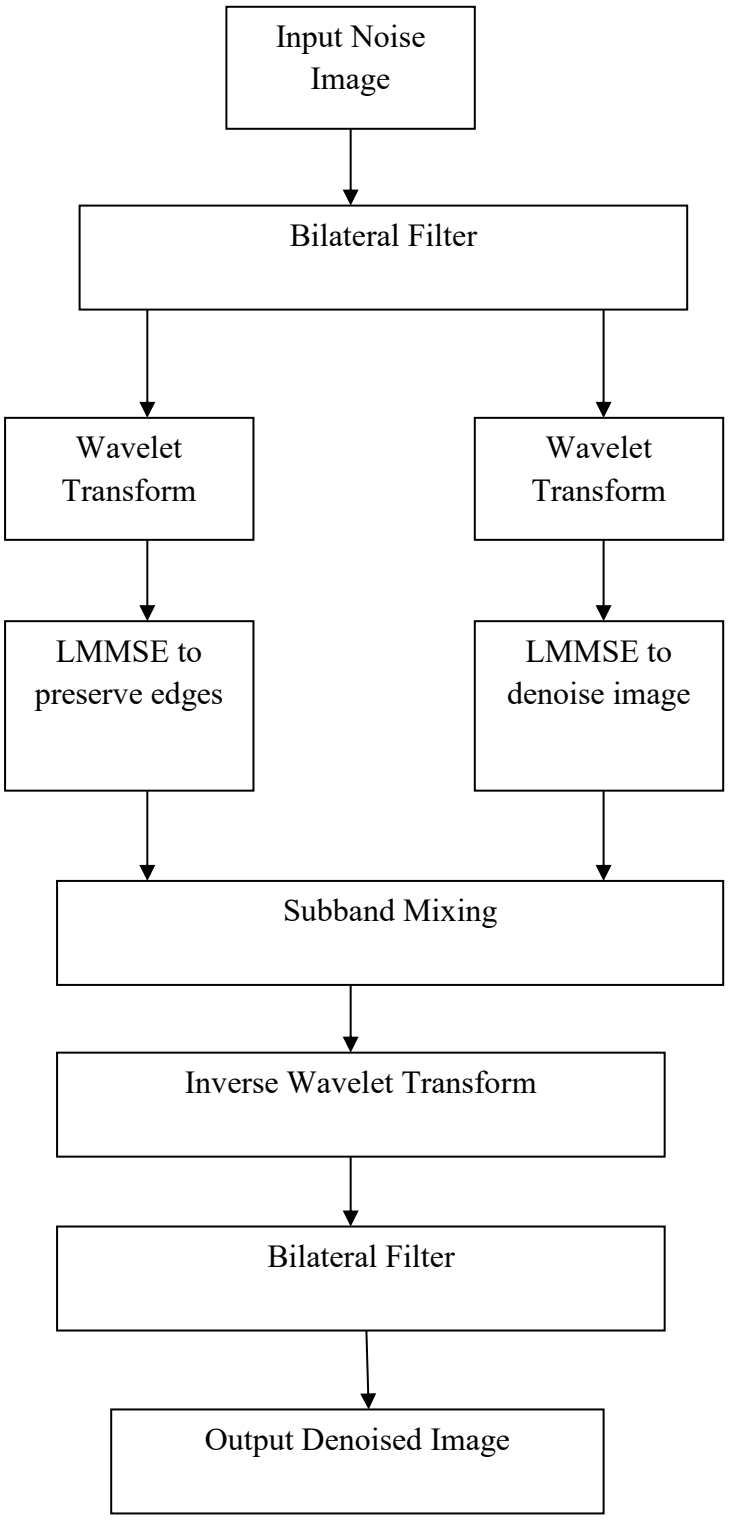


Figure 5.13: Algorithm for LMMSE Method

5.13.3 Denoising Algorithm

The input to the filter is either magnitude or square noisy images obtained by adding Rician noise at different levels to a relatively clean image as shown in figure 5.14. Such an image is shown in figure 5.15.

The LMMSE Estimator is with automatically estimated for size [5 5] neighbourhood both for filtering and noise estimation and then manually setting the standard deviation of noise. As the noise is dynamically estimated in each iteration, the filter should reach a steady state as the estimated noise diminishes. Therefore for appropriate noise estimation, the filter stops altering when the noise reduced to minimum possible.

To rate the denoising effectiveness of this method in comparison with others, the Structural Similarity (SSIM) index and the Quality Index based on Local Variance (QILV) measures are used and the mean square error is also determined. It is necessary that the measures are only applied on the part of the body being examined. Figure 5.16 shows the residue noise after denoising.

.

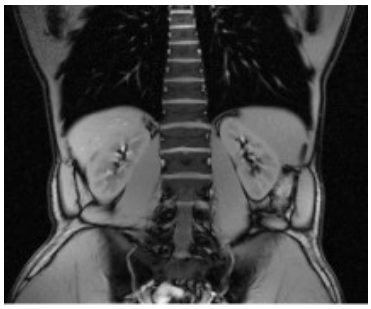
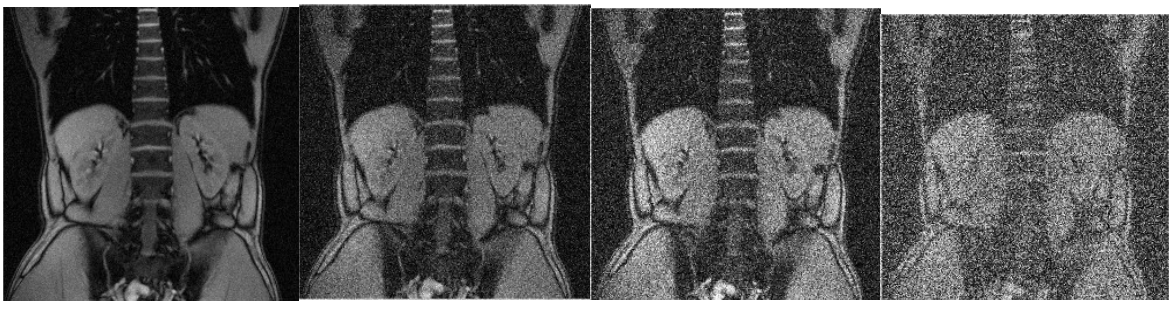


Figure 5.14: Relatively Clean Hip MRI Image



2% 5% 10% 20%

Figure 5.15: Noisy Hip Images at Various Levels

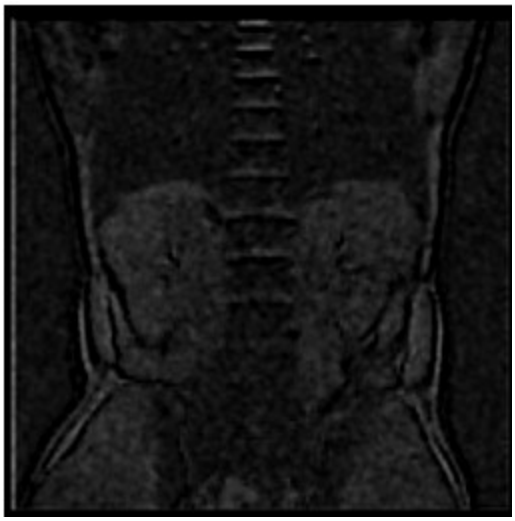
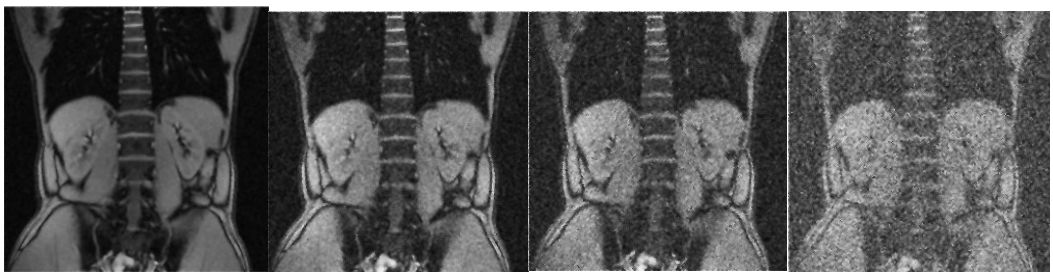


Figure 5.16: Residual Noise for Hip

5.13.4 Experimental Results

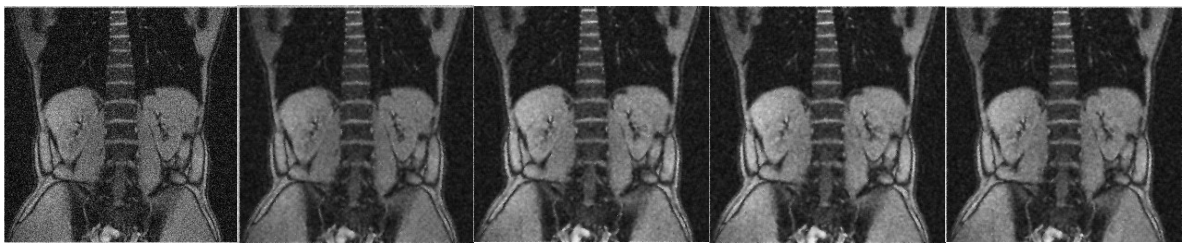
The proposed method developed was used to denoise a *Torso* structural image with Rician noise at levels 2%, 5%, 10% and 20%. The image was chosen because it exhibits many characteristics of interest including high frequency features, various features with different shapes and different edge gradients. The noisy images at these levels are shown in Figure 5.15

The denoising capabilities of the method are compared with Bilateral, Wiener, median thresholding, and other state of the art methods at the same noise levels. The images obtained are as shown; in figure 5.17 the denoised images of the combinational LMMSE filter at various noise levels are given. In figure 5.18 the resulting denoised images using the state of the art methods are shown.



2% **5%** **10%** **20%**

Figure 5.17: Denoised Hip images using Combinational LMMSE Algorithm



Wiener *Thresholding* *Bilateral* *Nowak* *Median*

Figure 5.18: Denoised Hip images using other methods

Detailed resulting image statistics are outlined in Tables 5.9, 5.10, 5.11 and 5.12.

Table 5.9: Quality Measures at 2% noise for combinational LMMSE algorithm

NoiseFilter type	RMSE	QILM	SSIM
Noisy image	16.23	0.96	0.86
Median filter	13.55	0.97	0.89
Wiener filter	11.12	0.97	0.97
Bilateral filter	8.654	0.99	0.98
Thresholding	22.43	0.95	0.93
Nowak filter	8.54	0.98	0.90
Combination filter	**7.25**	**0.98**	**0.96**

Table 5.10: Quality Measures at 5% noise for combinational LMMSE algorithm

NoiseFilter type	RMSE	QILM	SSIM
Noisy image	23.53	0.95	0.84
Median filter	18.45	0.98	0.89
Wiener filter	12.34.	0.99	0.95
Bilateral filter	8.654	0.98	0.95
Thresholding	34.12	0.87	0.84
Nowak filter	14.23	0.99	0.95
Combination filter	**13.56**	**0.99**	**0.95**

Table 5.11: Quality measures at 10% noise for combinational LMMSE algorithm

Filter type	RMSE	QILM	SSIM
Noisy image	35.45	0.80	0.74
Median filter	25.34	0.87	0.84
Wiener filter	19.95	0.97	0.87
Bilateral filter	16.78	0.98	0.89
Thresholding	45.67	0.95	0.84
Nowak filter	17.76	0.98	0.89
Combination filter	**16.43**	**0.99**	**0.89**

Table 5.12: Quality measures at 20% noise for combinational LMMSE algorithm

Filter type	RMSE	QILM	SSIM
Noisy image	46.72	0.85	0.65
Median filter	30.94	095	0.72
Wiener filter	25.43	0.96	0.73
Bilateral filter	22.69	0.96	0.80
Thresholding	57.31	0.95	0.73
Nowak filter	23.47	0.96	0.88
Combination filter	**20.73**	**0.97**	**0.88**

Result Analysis and discussion

Various measures of image quality used including RMSE, SSIM and QILM reveal better results in terms of effectiveness of denoising procedures. In LMMSE design it is taken that only one measurement realization of each pixel is available. So this component of the overall denoising procedure may require adjustments for multiple acquisitions. For diffusion weighted images a combinational filter with LMMSE is preferred to other combinational filters as the LMMSE estimators result in very effective denoising of such images.

5.14 Method IV: A Total Variational Wavelet based Structural MRI Denoising Method with Bilateral Feature Enhancement

In this method, a total variational minimization based analysis is used in which the objective is to minimize intensity and phase difference between pixels in the noisy image and the original image. This has the effect of denoising the image while preserving, and enhancing the image features. An automatic stopping criterion for the wavelet TV minimization methods is applied to the wavelet bases. The flowchart of the denoising algorithm is shown in Figure 5.19. In this method, the unique component is the total variational approximation procedure which involves tuning various parameters to optimum values depending on the noisy image characteristics and values of statistical quantities. The process of selecting the parameters is described in the following section.

A ground truth image of the cranium as shown in figure 5.20 is used to generate the noisy input images to the algorithm by adding various levels of Rician noise. These are shown in figure 5.21.

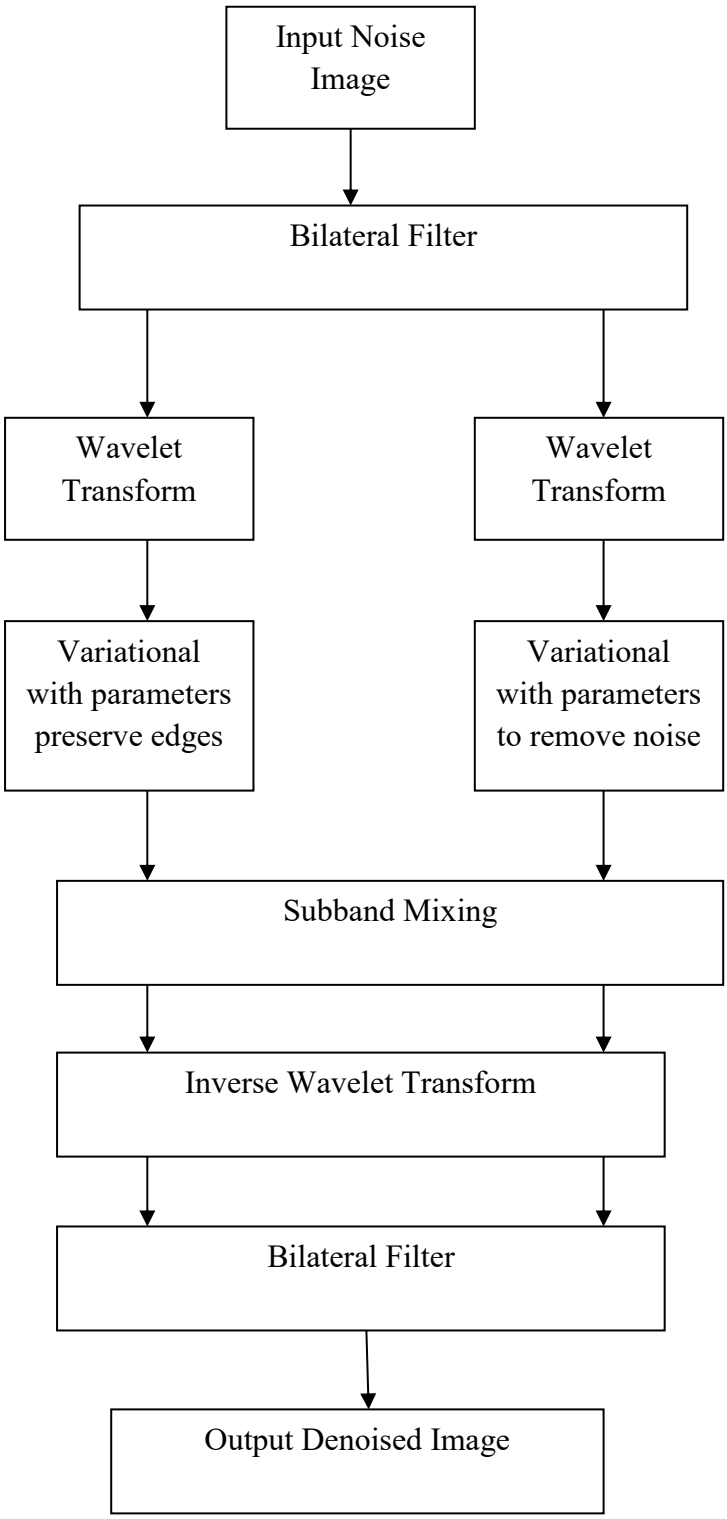


Figure 5.19: Algorithm for the Proposed method (Total variational combination method)

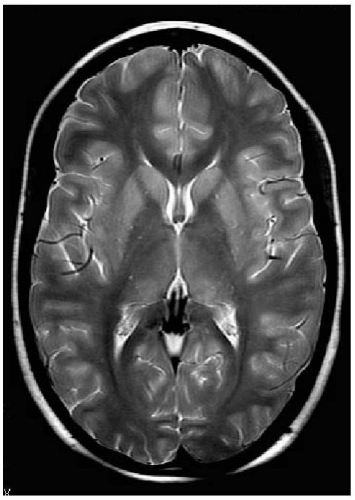
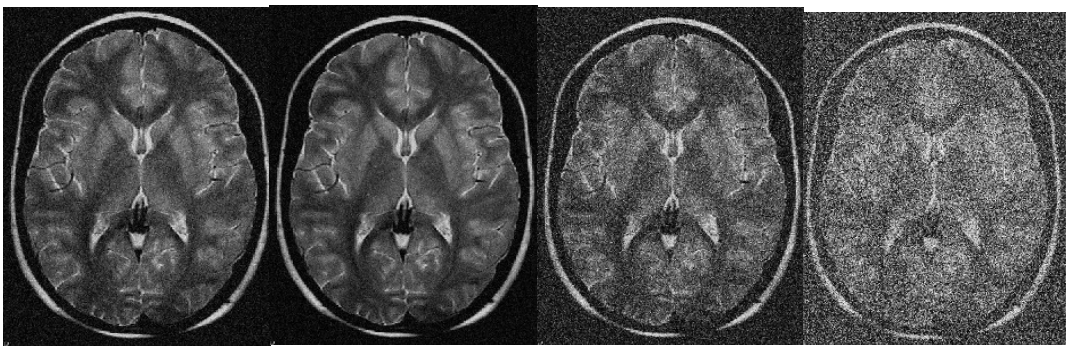


Figure 5.20: Relatively clean MRI image(cranium)



2% **5%** **10%** **20%**

Figure 5.21: Noisy Cranium Images at various levels

5.14.1 Selection of Parameters in Variation Denoising

For total variation with wavelet $\{\varphi_j\}$ is the basis for F=L^2 (Ω) the wavelet coefficients are chosen and enhanced to achieve desired denoising. For MRI application an automatic stopping criterion is needed together with multiscale fitting parameters which guide minimum number of interactions required to achieve acceptable denoising.

$$\text{Let } z(x) = \sum_{j \in I} \alpha_j \varphi_j(x) \tag{5.33}$$

And denote

$$u(x,\beta)=\sum_{j\in I}\beta_j\varphi_j(x) \tag{5.34}$$

Total variation function

$$F(u)=\int_{R^2}\left|\Delta_x u(x,\beta)dx+1/2\right|\sum_{j\in I}\lambda_j\left(\beta_j-\alpha_j\right)^2 \tag{5.35}$$

Where $u=u(x,\beta),\lambda_j>0$

$\left|\Delta_x u(x,\beta)\right|$ in practice may be replaced by $\left|\Delta_x u\right|E=\sqrt{\left|\Delta_x u\ \right|^2}+E$ with 0 <*E*<<1

E is introduced to prevent denominator vanishing in numerical implementations

To denoise *F*(u) the image need to be minimized

Take U^* = u (x, β*) so that

$$F(U^*)=\ \min\ \beta F(u) \tag{5.36}$$

In TV all λ_j are set to a single parameter λ .

To obtain minimum value

For $u = u\ (x, \beta\ )$

$$\frac{\delta F(u)}{\delta(\beta)}=\int_{R^2}\frac{\Delta xu}{\left|\Delta_x u\right|}.\Delta x\varphi_j dx+\lambda_j\left(\beta_j-\alpha_j\right) \tag{5.37}$$

$$=\int_{R^2}\Delta_x\left[\frac{\Delta xu}{\Delta_x u}\right].\varphi_j+dx+\lambda_j\left(\beta_j-\alpha_j\right) \tag{5.38}$$

Euler –Lagrange equation for the model is

$$\int_{R^2} \Delta_x \left[\frac{\Delta x u}{\Delta_x u} \right] . \varphi_j + dx + \lambda_j \left(\beta_j - \alpha_j \right) = 0 \tag{5.39}$$

At this point a time parameter is introduced and using gradient flow

Set $\beta = \beta(t) = (\beta_j(t)$ to solve the time evolution equation.

$$\frac{\delta \beta_j}{\gamma t} = \int_{R^2} \Delta_x \left[\frac{\Delta x u}{\Delta_x u} \right] . \varphi_j + (x) dx - \lambda_j \left(\beta_j - \alpha_j \right) \tag{5.40}$$

$$b_j(0) = \alpha_j$$

The minimized fuction is therefore the steady state of this equation.

The coefficient $u(x), \{\beta_j(0); j \in I\}$ will show how dirty the image is. In a completely clean image these coefficient will tend to be close to 0, but will be more in a noisy image.

Let $\mu(t) = \frac{1}{|jp|} \sum_{j \in I} |\beta_j(t)|$ be used to measure the noise in the image at time *t*

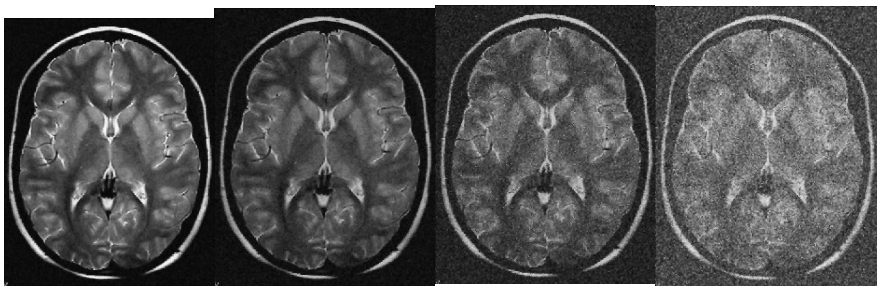
In a relative stopping criteria when $u(t)/u(0)$ is the threshold b. value of which ranges from (0.05 to 0.1) when noise has been reduced by 90%

b may be 0.03 for very noisy images

$$\text{Threshold } P = \frac{2}{|I_o|} \sum_{j \in I} \alpha_j \tag{5.41}$$

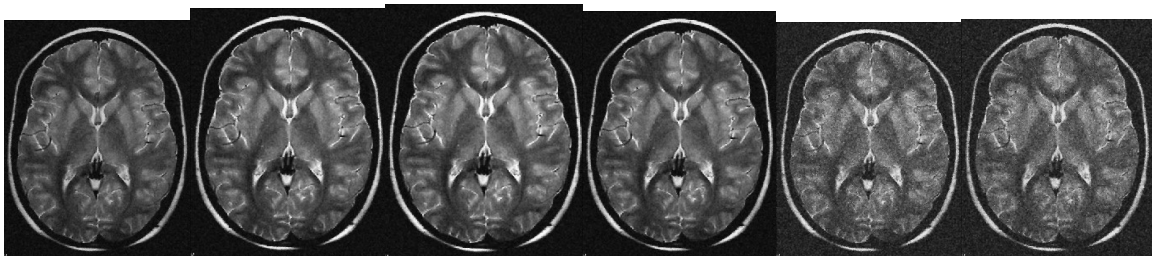
does not affect the automatic stopping criterion.

Figure 5.22: Residual noise for Cranium image



2% 5% 10% 20%

Figure 5.23: Denoised Cranium images new algorithm(Combinational total variational)



Wiener Thresholding Bilateral Nowak Median Pizzurica

Figure 5.24: Denoised Cranium images using other methods

5.14.2 Experimental Results

The method developed was used to denoise a cranium structural image with Rician noise at levels 2%, 5%, 10% and 20%. The image was chosen because it exhibits many characteristics of interest including high frequency content, various features with different shapes and different edge gradients. The noisy images at these levels are shown in Figure 5.21.

The denoising capabilities of the method are compared with the state of the art methods at the same noise levels. In figure 5.23 the denoised images of the combinational total variational filter at various noise levels are given. In figure 5.24 the resulting denoised images using the state of the art methods are shown.

Detailed statistics of the resulting images obtained using various method are given based on relevant measures of quality in tables 5.13, 5.14,5.15 and 5.16.

Table 5.13: Quality measures at 2% noise for combinational total variational algorithm

noise Filter type	SNR (dB)	PSNR (dB)	MSE
Noisy image	24.56	35.02	23.53
Median filter	36.54	37.64	18.45
Wiener filter	27.92	38.56	12.34.
Bilateral filter	28 .67	41.23	8.654
Thresholding	34.21	36.54	34.12
Nowak filter	26.87	39.54	14.23
Combination filter	38.25	37.82	10.24

Table 5.14:Quality measures at 5% noise for combinational total variational algorithm

noise Filter type	SNR (dB)	PSNR (dB)	MSE
Noisy image	24.56	35.02	23.53
Median filter	36.54	37.64	18.45
Wiener filter	27.92	38.56	12.34.
Bilateral filter	28 .67	41.23	8.654
Thresholding	34.21	36.54	34.12
Nowak filter	26.87	39.54	14.23
Combination filter	38.12	38.66	11.28

Table 5.15: Quality measures at 10% noise for combinational total variational algorithm

Filter type	SNR (dB)	PSNR (dB)	RMSE
Noisy image	21.45	29.5	35.45
Median filter	22.54	31.21	25.34
Wiener filter	24.28	33.25	19.95
Bilateral filter	25.69	34.53	16.78
Thresholding	20.32	28.51	45.67
Nowak filter	26.54	36.45	22.76
Combination filter	26.86	36.48	14.64

Table 5.16: Quality measures at 20% noise for combinational total variational algorithm

Filter type	SNR (dB)	PSNR (dB)	RMSE
Noisy image	21.45	29.5	35.45
Median filter	22.54	31.21	25.34
Wiener filter	24.28	33.25	19.95
Bilateral filter	25.69	34.53	16.78
Thresholding	20.32	28.51	45.67
Nowak filter	26.54	36.45	22.76
Combination filter	28.02	35.86	14.28

5.14.3 Result Analysis and Discussion

Measures of performance including PSNR, RMSE, and also visual inspection show that there is significant improvement that is obtained using stand alone methods such as Gaussian smoothing, wiener filter, NLM filter, bilateral filter and wavelet thresholding. For example, the combination of filter improves the PSNR from 29.5 to 35.86 dB.

When the four combination methods are compared for noise levels of five percent and ten percent it is found that the LMMSE is the best performance followed by combinational non-local means filter then y total variational denoising and then and least performing is the non-centric chi-square filter.

CHAPTER SIX

CONCLUSION AND RECOMMENDATION FOR FURTHER WORK

6.1 Conclusion

The MRI is one of the most effective medical imaging technologies. It produces an image that has relatively high resolution and does not have any harmful radiations to the human being during image acquisition. The Rician distributed noise present in the acquired image however should be minimized for image analysis, segmentation and other forms of processing to extract information on the condition and functioning of body tissues.

This thesis has developed four wavelet based MRI denoising methods with bilateral filter enhancement. In all the four cases multi-resolution thresholding and appropriate sub-band combination have been used. The thesis has further developed the concept of combinational image filters where each component is critical in a given aspect of image denoising including preserving fine features, enhancing boundaries, reducing artifacts, deblurring among others. The filters used for each component include non-local means filter, LMMSE filter, total variational filter, Nowak square image filter, Wiener, fourth order and bilateral filter.

These combination filters in general and specific(as explained in chapter 5 for each) have been shown to increase the effectiveness and efficiency of image denoising in terms of various measures of quality and visual inspection when compared to the stand alone filters.

6.2 Recommendations

There are many possibilities of extending the work in this thesis. They can categorized as follows

One approach is to subject other categories of MRI images to the filters. Closely related to this is to apply the combination filters on colour medical images and the investigation performance.

Other combination filters can be developed and more advanced signal and noise estimation methods using approaches based on linear estimation of thresholds for appropriate statistical distributions of the noise to be removed can be proposed.

Another is to concentrate on the MATLAB and 'C' language routines realized and developing application packages where the main task would be to create an application taking into consideration all the requirements of user friendly medical software.

The instrumentation system for MRI acquisition and initial processing can be assessed and new methods of improving the acquired image proposed in terms of resolution, noise freeness and clarity.

Use of Karhunen Louve Transform in combination with the methods used in this thesis including LMMSE, Total Variational and Chi-square estimation for functional MRI images. The Anisotropic diffusion method may also be used in place of total variation for functional MRI image denoising.

REFERENCES

1. R. Nowak, "Wavelet-based Rician noise removal for Magnetic Resonance Imaging," IEEE Transactions on Image Processing, vol. 8, No. 10, pp. 1408–1419, October 1999.
2. T. Dylan "MRI denoising via phase error estimation", medical imaging journal proc on image processing SPIE Vol.5, No. 747, 2005.
3. S. R. Fernandez, M. Niethammer, M. Kabicki, M. E. Shenton, C. F. Weston, "Restoration of DWI data using a Rician LMMSE Estimator", IEEE Transactions on medical imaging, Vol. 27,No. 10,pp. c1-c4, October 2008.
4. J. V. Manj´om, P. Coupé, L. Marti-bonmati, M. Robles, and D. L. Collins, "Adaptive non-local means denoising of MR images with spatially varying noise levels," Journal of Magnetic Resonance Imaging, Vol. 31, pp.192–203, 2010.
5. M. Lysaker, A. Lundervold, and X. C. Tai, "Noise removal using fourth-order partial differential equation with applications to medical magnetic resonance images in space and time," IEEE Transactions on Image Processing, vol. 12, No. 12, pp. 1579–1590, December 2003.
6. Z. A. Mustafa, Y M. Kadah, "Multiresolution Bilateral Filter for MR Image Denoising ," Biomedical Engineering Department, Cairo University, Egypt. Pp.176-179 IEEE 2011.
7. R. Sudipta "A new hybrid image denoising method", International Journal of Information Technology and Knowledge Management July-December 2010, Vol. 2,pp. 491-497.
8. V Loganayaagi, "An improved Denoising Algorithm Using Wavelet Transform for Magnetic Resonance Images", International journal of Communications and Engineering Vol. 07, No 7, March2012.
9. A. Pizurica, A. M. Wink, E. Vansteenkiste, W. Philips, and J. B. T. M. Roerdink, "A review of wavelet denoising in MRI and ultrasound brain imaging," Current Medical Imaging Reviews, Vol. 2, No. 2, pp. 247–260, May 2006.
10. S. Dolui, A. Kuurstra, C. Iv´an, S. Patarroyo and V. Oleg, Michailovich "A New Similarity Measure for Non-Local Means Filtering of MRI Images," Elsevier Journal of

Visual communications and image representation. Vol. 24, No. 7, pp. 1040-1054, October 28 2011.

11. T. Tasdizen "Principal Neighbourhood Dictionaries for Non-local Means Image Denoising," IEEE Transactions **o**n Image Processing, Vol. 18, No. 12, pp. 2649-2660, January 2009.
12. P. Coupé, P. Yger, S. Prima, P. Hellier, C. Kervrann, and C. Barillot, "An optimized blockwise nonlocal means denoising filter for 3-D magnetic resonance images," IEEE Transactions on Medical Imaging, Vol. 27, No. 4, pp. 425–441, March 2008.
13. C. Lakshmi Devasena "Noise Removal in Magnetic Resonance Images using Hybrid KSL Filtering Technique," International Journal of Computer Applications (0975 – 8887) Vol. 27, No.8, August 2011.
14. Palaniappan "Denoising of dynamic magnetic resonance images by combined application of wavelet filtering and Karhunen-Loeve Transform (KLT)", Journal of Cardiovascular Magnetic resonance, 2012 14(sup1) W71.
15. Jose V. Manjon "Multi-component MR image denoising", International Journal of Biomedical imaging volume 2009, article ID756897.
16. F. Luisier, T. Bhi, P. J. Wolfe. "A Cure for Noisy Magnetic Resonance Images: Chi-square Unbiased Risk Estimation", IEEE transactons on image processing, Vol. 21, No. 8, pp. 3454-3466, 2012.
17. F. Luisier "The SURE-LET Approach to Image Denoising" Doctoral thesis January 2010 Ecole Polytechnique Fédérale de Lausanne, Switzerland.
18. J. A.Wells "Arterial Spin Labelling Magnetic Resonance Imaging of the Brain: Techniques and Development" Ph.D Thesis University College London, United Kingdom.
19. M. T. G. Sebastián, Doctoral Thesis "Contributions to Brain MRI Processing and Analysis" September 2009. The University of the Basque Country Donostia - San Sebastian, Spain.

20. J. D. Clayden, Doctoral Thesis "Comparative Analysis of Connection and Disconnection in the Human Brain Using Diffusion MRI: New Methods and Applications" 2008 University of Edinburgh, UK.
21. A. Raj "Improvements in magnetic resonance imaging using information redundancy" Doctoral thesis May 2005 Cornell University, New York, USA .
22. J.Rajan, J. Veraart, J. Audekerke, M. Verhoye, J. Sijbers, Nonlocal maximum likelihood estimation method for denoising multiple-coil magnetic resonance images. Magnetic Resonance Imaging, Vol. 30, pp. 1512-1518, 2012.
23. J. Rajan, A. Dekker, J. Sijbers.A new non-local maximum likelihood estimation method for Rician noise reduction in magnetic resonance images using the Kolmogorov-Smirnov test. A journal of Signal Process. Vol. 103, pp.16–23. 2014.
24. A., Tristán-Vega, V. García-Pérez, S. Aja-Fernández. "Efficient and robust nonlocal means denoising of MR data based on salient features matching". Computer Method Programmes in Biomedicine, Vol. 105, pp.131–144. 2012.
25. J. Petr "Parallel Magnetic Resonance Imaging Reconstruction", Doctoral Thesis'petrj5@cmp.felk.cvut.cz,ctu–cmp–2007–09 Czech Technical university, Prague, Czechoslovakia. May 2007.
26. F. Bloch, W. W. Hanson, and M. E. Packard. Nuclear induction. Physical review, 69(2):127, 1946.
27. A. Kumar, D. Welti, and R. R. Ernst. "NMR Fourier zeugmatography". Journal of MagneticResonance, Vol. 18, pp.69–83, 1975.
28. P. C. Lauterbur. "Image formation by induced local interactions: Examples employing nuclear magnetic resonance". Nature (London), Vol. 242, pp.190–191, 1973.
29. M. T. Vlaardingerbroek and J. A. Boer. Magnetic Resonance Imaging: Theory and Practice. Springer-Verlag, 1996.
30. D. Kwiat and S. Einav. A decoupled coil detector array for fast image acquisition in magnetic resonance imaging. Medical Physics, Vol. 18, pp. 251–6, 1991.
31. P. B. Roemer, W. A. Edelstein, C. E. Hayes, S. P. Souza, and O.M.Mueller. The NMR phased array. Magnetic Resonance in Medicine, Vol. 16, pp.192–225, 1990.

32. D. K. Sodickson and C. A. McKenzie. A generalized approach to parallel magnetic resonance imaging. Medical Physics, Vol. 28, No. 8, pp. 1629–1643, August 2001.
33. P. Kellman and E. R. McVeigh. SENSE coefficient calculation using adaptive regularization. In Proceedings of ISMRM Workshop on Minimum MR Data Acquisition Methods, Marco Island, Florida, USA, October 2001. ISMRM.
34. F. A. Breuer, M. Blaimer, R.M. Heidemann, M. F. Mueller, M. A. Griswold, and P. M. Jakob. Controlled aliasing in parallel imaging results in higher acceleration (CAIPIRINHA) for multi-slice imaging. Magnetic Resonance in Medicine, Vol. 53, No.3, pp.684–691,March 2005
35. Harold Phelippeau. Shot Noise Adaptive Bilateral Filter. IEEE Signal Processing, ICSP 2008, 9th International Conference, Beijing, China, pp. 864 – 867,2008
36. M. Zhang and B. K. Gunturk, ”Multiresolution Bilateral Filtering for Image Denoising”, IEEE Trans Image Process, Vol. 17,pp. 2324–2333,2008.
37. Z.Liao, S. Hu,Z. Yu,D. Sun. Medical Image Blind Denoising Using Context Bilateral Filter. International Conference of Medical Image Analysis and Clinical Application, , Guangzhou, China, pp.12-17,June 2010.
38. C.Tomasi, R. Manduchi. Bilateral Filtering for Gray and Color images. Proceedings of IEEE international Conference on Computer Vision, Mumbai, India, 1998, pp. 839-846.
39. M.I Elad. On the Origin of the Bilateral Filter and Ways to Improve It. IEEE transaction on image processing, 2002, Vol. 11, No. pp.1141-1151.
40. P. Perona and J. Malik, “Scale-Space and Edge Detection Using Anisotropic Diffusion,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 12, no. 7, pp. 629–639, July 1990.
41. L. Zhang, J. Chen, Y. Zhu, J. Luo. Comparisons of Several New Denoising Methods for Medical Images. IEEE, “Proceeding on Medical Imaging”2009, pp.1-4.
42. S.K. Mohideen, S. A. Perumal, M.M. Sathik. Image De-noising using Discrete Wavelet transform. IJCSNS International Journal of Computer Science and Network Security, (IJCSNS). 2008, Vol. 8, No.1, pp.213-216.

43. B. Zhang. Adaptive Bilateral Filter for Sharpness Enhancement and Noise Removal. IEEE transaction on image processing, Annual symposium of the Pattern Recognition Association of South Africa. November 2003, Vol. 17, No. 5, pp. 664-678.
44. J.J. Francis, G. de Jager. "The Bilateral Median Filter" IEEE Transactions on Image Processing, 2008, pp. 421–432.
45. P. Adtormann, B. Gossens, W. Phillips "Removal of Correlated Rician Noise in Magnetic Resonance Imaging", 16th European Signal processing Conference (EUSIPCO 2008), Lausanne, Switzerland, August 2008, pp.25- 29.
46. A.K. Jain "Fundamentals of digital image processing", Prentice Hall Engle Wood Cliffs, New Jersey USA 1989).
47. www.brainweb.bic.min.mcgill.ca/brainweb, June 15 2014.
48. http://bigwww.epfl.ch/luisier/MRIdenoising.TestImages.zip, June 15 2014.
49. www/Mc Cluinie.com, June 15 2014.
50. Y.Q. Zhang, Y. Ding,J.Y. Liu, Z.M. Guo." Guided image filtering using signal subspace projection". IET Image Proc.2013, Vol. 7, pp.270–279.
51. Y.Q. Zhang, J.Y. Liu, M.D. Li, Z.M. Guo,.Joint image denoising using adaptive principal component analysis and self-similarity. Information Science journal.2014. Vol. 259, pp.128–141.
52. J. Salmon, R. Willett, E. Arias-Castro, 2012. A two-stage denoising filter: the preprocessed Yaroslavsky filter. Statistical Signal Processing Workshop (SSP), Michigan U.S.A, August 2012 IEEE, pp. 464–467.
53. S. Pyatykh, J. Hesser, L. Zheng. Image noise level estimation by principal component analysis, IEEE Transactions on image Processing. 2013, Vol. 22, pp.687–699.
54. J.V. Manjón, P. Coupé, L. Concha, A. Buades, D.L. Collins, M. Robles, Diffusion weighted image denoising using overcomplete local PCA. PLoS ONE. 2013;8:e730211518.
55. J.V. Manjón, P. Coupé, A. Buades, D.L. Collins, M. Robles, New methods for MRI denoising based on sparseness and self-similarity. Medical Image Annual Journal.2012, Vol. 16, pp. 8–27.

56. M. Maggioni, A. Foi. Nonlocal transform-domain denoising of volumetric data with groupwise adaptive variance estimation. In: Proc. SPIE Electronic Imaging (EI), January 2012, San Francisco, California, USA.
57. M. Maggioni, V. Katkovnik, K. Egiazarian, A. Foi.A nonlocal transform-domain filter for volumetric data denoising and reconstruction. IEEE Trans. Image Process.2013, Vol. 22, pp.119–133.
58. D.W. Kim, C. Kim, D.H. Kim, D.H. Lim, Rician nonlocal means denoising for MR images using nonparametric principal component analysis. EURASIP J. Image Video Process.2011;2011:15.
59. S. Fan Lam, D. Babacan, J.P. Haldar, M.W. Weiner, N. Schuff, Denoising diffusion-weighted magnitude MR images using rank and edge constraints. Magnetic Resonance Medical Journal, 2013, Vol. 69, pp.1–13.
60. J. Mohan, V. Krishnaveni, Y.Guo.A survey on the magnetic resonance image denoising methods. Biomedical. Signal Process and Control Journal. 2014, Vol. 9, pp.56–69.
61. J. Salmon, R.Willett, E. Arias-Castro. A two-stage denoising filter: the preprocessed Yaroslavsky filter. Statistical Signal Processing Workshop (SSP), 2012 IEEE, pp. 464–467, Ann Arbor, Michigan USA, August 2012
62. A. Tristán-Vega, V. García-Pérez, S. Aja-Fernández.Efficient and robust nonlocal means denoising of MR data based on salient features matching. Computer Methods Program in Biomedicine. Bio.2012, Vol. No.105, pp.131–144.
63. A. Pizuria,A, Philips, W. Lemahieu and M. Acheroy " A versatile wavelet domain noise filtration technique for medical imaging" ,IEEE Trans Med Imaging 2003, Vol. 22, No. 3, pp 232-331.
64. Hossein "Wavelet Domain medical image denoising using Bivariate Laplacian mixture model", IEEE Transactions on Biomedical Engineering vol 56 No 12 December 2009
65. A.Webb. Introduction to Biomedical Imaging. John Wiley & Sons Inc, January 2003.
66. C. R. Smith and R. C. Lange. Understanding magnetic resonance imaging. CRC Press LCC, 1998.

67. J. P. Hornak. The basics of MRI, http://www.cis.rit.edu/htbooks/mri. 1996-2004 June 15 2014
68. J. Mohan, V. Krishnaveni, Y. Guo. A survey on the magnetic resonance image denoising methods, Biomedical Signal Processing and Control.2012. Vol. 8, No. 6. 56-69.
69. J. Mohan, V. Krishnaveni, Y. Guo. MRI denoising using nonlocal neutrosophic set approach of Wiener filtering. Biomedical Signal Processing and Control. February 2012, Vol. 8, No. 6, pp. 779-791.
70. J. Mohan, Y. Guo, V. Krishnaveni, K. Jeganathan. MRI denoising based on neutrosophic Wiener filtering. February 2012 IEEE International Conference on Imaging Systems and Techniques, Manchester,UK
71. J Mohan, V Krishnaveni, Y Guo A new neutrosophic approach of Wiener filtering for MRI denoising. Measurement Science Review,Vol. 13, No. 4, pp.177-186
72. J. Mohan, V. Krishnaveni, Y. Guo A Neutrosophic approach of MRI denoising Image Information Processing (ICIIP), International Conference of Image, Nov. 2011, Shimla India pp.1-6.
73. J. Mohan, APTS Chandra, V. Krishnaveni, Y. Guo.Evaluation of Neutrosophic Set Approach Filtering Technique for Image Denoising. The International Journal of Multimedia & Its Applications (IJMA). Vol 4. August 2012.
74. J. Mohan, V. Krishnaveni, Y. Guo. Performance analysis of neutrosophic set approach of median filtering for MRI denoising. International Journal of Electronics and Communication Engineering, vol 3 no 2 pp148-163 September 2012.
75. J. Mohan, V. Krishnaveni, Y. Huo.Automated brain tumor segmentation on MR images based on neutrosophic set approach. 2nd International conference on Electronics and Communication Systems (ICECS), Cairo, Egypt. September 2015
76. J. Mohan, APTS Chandra, V. Krishnaveni, Y. Guo.Image Denoising Based on Neutrosophic Wiener Filtering. Proceedings of Second international conference on Advances in Computing and Information Technology, July 2012 Chennai India vol 1 No 1 861-869.
77. H. Gudbjartsson and S. Patz. "The Rician distribution of noisy MRI data", Magnetic Resonance in Medicine, 34(6), pp 910-914,1995.

APPENDICES

Appendix 1: Measures of quality

```
function [mssim, ssim_map] = ssim(img1, img2, K, window, L)

% ========================================================================
% SSIM Index with automatic downsampling, Version 1.0
% Copyright(c) 2009 Zhou Wang
% All Rights Reserved.
%
% ----------------------------------------------------------------------
% Permission to use, copy, or modify this software and its documentation
% for educational and research purposes only and without fee is hereby
% granted, provided that this copyright notice and the original authors'
% names appear on all copies and supporting documentation. This program
% shall not be used, rewritten, or adapted as the basis of a commercial
% software or hardware product without first obtaining permission of the
% authors. The authors make no representations about the suitability of
% this software for any purpose. It is provided "as is" without express
% or implied warranty.
%----------------------------------------------------------------------
%
% This is an implementation of the algorithm for calculating the
% Structural SIMilarity (SSIM) index between two images
%
% Please refer to the following paper and the website with suggested usage
%
% Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image
% quality assessment: From error visibility to structural similarity,"
% IEEE Transactios on Image Processing, vol. 13, no. 4, pp. 600-612,
% Apr. 2004.
%
% http://www.ece.uwaterloo.ca/~z70wang/research/ssim/
%
% Note: This program is different from ssim_index.m, where no automatic
% downsampling is performed. (downsampling was done in the above paper
% and was described as suggested usage in the above website.)
%
% Kindly report any suggestions or corrections to zhouwang@ieee.org
%
%----------------------------------------------------------------------
%
%Input : (1) img1: the first image being compared
%        (2) img2: the second image being compared
%        (3) K: constants in the SSIM index formula (see the above
%            reference). defualt value: K = [0.01 0.03]
%        (4) window: local window for statistics (see the above
%            reference). default widnow is Gaussian given by
%            window = fspecial('gaussian', 11, 1.5);
%        (5) L: dynamic range of the images. default: L = 255
%
%Output: (1) mssim: the mean SSIM index value between 2 images.
%            If one of the images being compared is regarded as
%            perfect quality, then mssim can be considered as the
%            quality measure of the other image.
```

```
%            If img1 = img2, then mssim = 1.
%        (2) ssim_map: the SSIM index map of the test image. The map
%            has a smaller size than the input images. The actual size
%            depends on the window size and the downsampling factor.
%
%Basic Usage:
%   Given 2 test images img1 and img2, whose dynamic range is 0-255
%
%   [mssim, ssim_map] = ssim(img1, img2);
%
%Advanced Usage:
%   User defined parameters. For example
%
%   K = [0.05 0.05];
%   window = ones(8);
%   L = 100;
%   [mssim, ssim_map] = ssim(img1, img2, K, window, L);
%
%Visualize the results:
%
%   mssim                        %Gives the mssim value
%   imshow(max(0, ssim_map).^4)  %Shows the SSIM index map
%=========================================================================


if (nargin < 2 || nargin > 5)
   mssim = -Inf;
   ssim_map = -Inf;
   return;
end

if (size(img1) ~= size(img2))
   mssim = -Inf;
   ssim_map = -Inf;
   return;
end

[M N] = size(img1);

if (nargin == 2)
   if ((M < 11) || (N < 11))
           mssim = -Inf;
           ssim_map = -Inf;
      return
   end
   window = fspecial('gaussian', 11, 1.5);    %
   K(1) = 0.01;                               % default settings
   K(2) = 0.03;                               %
   L = 255;                                     %
end

if (nargin == 3)
   if ((M < 11) || (N < 11))
           mssim = -Inf;
           ssim_map = -Inf;
      return
   end
```

```
   window = fspecial('gaussian', 11, 1.5);
   L = 255;
   if (length(K) == 2)
      if (K(1) < 0 || K(2) < 0)
                  mssim = -Inf;
               ssim_map = -Inf;
               return;
      end
   else
           mssim = -Inf;
        ssim_map = -Inf;
           return;
   end
end

if (nargin == 4)
   [H W] = size(window);
   if ((H*W) < 4 || (H > M) || (W > N))
           mssim = -Inf;
           ssim_map = -Inf;
      return
   end
   L = 255;
   if (length(K) == 2)
      if (K(1) < 0 || K(2) < 0)
                  mssim = -Inf;
               ssim_map = -Inf;
               return;
      end
   else
           mssim = -Inf;
        ssim_map = -Inf;
           return;
   end
end

if (nargin == 5)
   [H W] = size(window);
   if ((H*W) < 4 || (H > M) || (W > N))
           mssim = -Inf;
           ssim_map = -Inf;
      return
   end
   if (length(K) == 2)
      if (K(1) < 0 || K(2) < 0)
                  mssim = -Inf;
               ssim_map = -Inf;
               return;
      end
   else
           mssim = -Inf;
        ssim_map = -Inf;
           return;
   end
end
```

```
img1 = double(img1);
img2 = double(img2);

% automatic downsampling
f = max(1,round(min(M,N)/256));
%downsampling by f
%use a simple low-pass filter
if(f>1)
    lpf = ones(f,f);
    lpf = lpf/sum(lpf(:));
    img1 = imfilter(img1,lpf,'symmetric','same');
    img2 = imfilter(img2,lpf,'symmetric','same');

    img1 = img1(1:f:end,1:f:end);
    img2 = img2(1:f:end,1:f:end);
end

C1 = (K(1)*L)^2;
C2 = (K(2)*L)^2;
window = window/sum(sum(window));

mu1   = filter2(window, img1, 'valid');
mu2   = filter2(window, img2, 'valid');
mu1_sq = mu1.*mu1;
mu2_sq = mu2.*mu2;
mu1_mu2 = mu1.*mu2;
sigma1_sq = filter2(window, img1.*img1, 'valid') - mu1_sq;
sigma2_sq = filter2(window, img2.*img2, 'valid') - mu2_sq;
sigma12 = filter2(window, img1.*img2, 'valid') - mu1_mu2;

if (C1 > 0 && C2 > 0)
   ssim_map = ((2*mu1_mu2 + C1).*(2*sigma12 + C2))./((mu1_sq + mu2_sq +
C1).*(sigma1_sq + sigma2_sq + C2));
else
   numerator1 = 2*mu1_mu2 + C1;
   numerator2 = 2*sigma12 + C2;
        denominator1 = mu1_sq + mu2_sq + C1;
   denominator2 = sigma1_sq + sigma2_sq + C2;
   ssim_map = ones(size(mu1));
   index = (denominator1.*denominator2 > 0);
   ssim_map(index) =
(numerator1(index).*numerator2(index))./(denominator1(index).*denominator2
(index));
   index = (denominator1 ~= 0) & (denominator2 == 0);
   ssim_map(index) = numerator1(index)./denominator1(index);
end

mssim = mean2(ssim_map);

return
function [quality, quality_map] = img_qi(img1, img2, block_size)

%=========================================================================
%
%Copyright (c) 2001 The University of Texas at Austin
%All Rights Reserved.
%
```

```
%This program is free software; you can redistribute it and/or modify
%it under the terms of the GNU General Public License as published by
%the Free Software Foundation; either version 2 of the License, or
%(at your option) any later version.
%
%This program is distributed in the hope that it will be useful,
%but WITHOUT ANY WARRANTY; without even the implied warranty of
%MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
%GNU General Public License for more details.
%
%The GNU Public License is available in the file LICENSE, or you
%can write to the Free Software Foundation, Inc., 59 Temple Place -
%Suite 330, Boston, MA 02111-1307, USA, or you can find it on the
%World Wide Web at http://www.fsf.org.
%
%Author  : Zhou Wang
%Version : 1.0
%
%The authors are with the Laboratory for Image and Video Engineering
%(LIVE), Department of Electrical and Computer Engineering, The
%University of Texas at Austin, Austin, TX.
%
%Kindly report any suggestions or corrections to zwang@ece.utexas.edu
%
%Acknowledgement:
%The author would like to thank Mr. Umesh Rajashekar, the Matlab master
%in our lab, for spending his precious time and giving his kind help
%on writing this program. Without his help, this program would not
%achieve its current efficiency.
%
%========================================================================
%
%This is an efficient implementation of the algorithm for calculating
%the universal image quality index proposed by Zhou Wang and Alan C.
%Bovik. Please refer to the paper "A Universal Image Quality Index"
%by Zhou Wang and Alan C. Bovik, published in IEEE Signal Processing
%Letters, 2001. In order to run this function, you must have Matlab's
%Image Processing Toobox.
%
%Input : an original image and a test image of the same size
%Output: (1) an overall quality index of the test image, with a value
%            range of [-1, 1].
%        (2) a quality map of the test image. The map has a smaller
%            size than the input images. The actual size is
%            img_size - BLOCK_SIZE + 1.
%
%Usage:
%
%1. Load the original and the test images into two matrices
%   (say img1 and img2)
%
%2. Run this function in one of the two ways:
%
%   % Choice 1 (suggested):
%   [qi qi_map] = img_qi(img1, img2);
%
%   % Choice 2:
```

```
%   [qi qi_map] = img_qi(img1, img2, BLOCK_SIZE);
%
%   The default BLOCK_SIZE is 8 (Choice 1). Otherwise, you can specify
%   it by yourself (Choice 2).
%
%3. See the results:
%
%   qi                      %Gives the over quality index.
%   imshow((qi_map+1)/2)  %Shows the quality map as an image.
%
%=========================================================================

if (nargin == 1 | nargin > 3)
   quality = -Inf;
   quality_map = -1*ones(size(img1));
   return;
end

if (size(img1) ~= size(img2))
   quality = -Inf;
   quality_map = -1*ones(size(img1));
   return;
end

if (nargin == 2)
   block_size = 8;
end

N = block_size.^2;
sum2_filter = ones(block_size);

img1_sq   = img1.*img1;
img2_sq   = img2.*img2;
img12 = img1.*img2;

img1_sum   = filter2(sum2_filter, img1, 'valid');
img2_sum   = filter2(sum2_filter, img2, 'valid');
img1_sq_sum = filter2(sum2_filter, img1_sq, 'valid');
img2_sq_sum = filter2(sum2_filter, img2_sq, 'valid');
img12_sum = filter2(sum2_filter, img12, 'valid');

img12_sum_mul = img1_sum.*img2_sum;
img12_sq_sum_mul = img1_sum.*img1_sum + img2_sum.*img2_sum;
numerator = 4*(N*img12_sum - img12_sum_mul).*img12_sum_mul;
denominator1 = N*(img1_sq_sum + img2_sq_sum) - img12_sq_sum_mul;
denominator = denominator1.*img12_sq_sum_mul;

quality_map = ones(size(denominator));
index = (denominator1 == 0) & (img12_sq_sum_mul ~= 0);
quality_map(index) = 2*img12_sum_mul(index)./img12_sq_sum_mul(index);
index = (denominator ~= 0);
quality_map(index) = numerator(index)./denominator(index);

quality = mean2(quality_map);
```

Appendix 2: STAND ALONE FILTERS

Rician generation using two orthogonal Gaussian generators

```
=========================================================================
% Rician noise generation using two orthogal Gaussian  generators
% copyright © 2016 Kenneth Kagoiya

% Permission to use, copy or modify this software and its document.
% This is a method of generating  Rician noise from two orthogal Gaussian
% genertors  and also exact image characteristic  including pixel
% intensity mean and standard devation for use in setting parameters of
% various parts of the combinational filters developed.
-------------------------------------------------------------------------

A= imread('C:/torso.bmp');
I = rgb2gray(A);
for phi= 0:0.375:1.5
II=I*cos(phi)
J = imnoise(II,'gaussian',0.0,0.0005);
J = imresize(J,[ 480 480])
figure,imshow(J)
K=I*sin(phi)
K = imnoise(K,'gaussian',0.0,0.0005);
K = imresize(K,[ 480 480])
figure,imshow(K)
L=(J+K)*0.75
figure,imshow(L)
end
val1 = std2(I)
val2 = mean2(I)
val3 = std2(II)
val4 = mean2(II)
val5 = std2(K)
val6 = mean2(K)
val7 = std2(K)
val8 = mean2(K)
```

FOURTH ORDER DENOISING

```
I=L
T=4

% Automatic noise removal using You method
% fourth order PDE
% Method : du/dt = - del^2[c(del^2(u))del^u]
% where u is the noisy input image.
function [frth]=fpdepyou(I,T)
% I = Noisy Image
% T - Threshold , Based on the threshold you will get scale sapce images.
At a particular
% value of T you will gwt the converged image.
[x y z]=size(I);
I=double(I);
dt=0.9; % Time step
I1=I;
I2=I;
t=1;
```

```
k=0.5;
for  t=1:T
    [Ix,Iy]=gradient(I1);
    [Ixx,Iyt]=gradient(Ix);
    [Ixt,Iyy]=gradient(Iy);
    c=1./(1.+sqrt(Ixx.^2+Iyy.^2)+0.0000001);
    [div1,divt1]=gradient(c.*Ixx);
    [divt2,div2]=gradient(c.*Iyy);
    [div11,divt3]=gradient(div1);
    [divt4,div22]=gradient(div2);
    div=div11+div22;
    I2=I1-(dt.*div);
    I1=I2;
end;
frth=uint8(I1); % Converting to 8 bit image
```

WIENER WITH THRESHOLDING

```
==========================================================================
% Rician noise generation using two orthogal Gaussian  generators
% copyright © 2016 Kenneth Kagoiya
% Permission to use, copy or modify this software and its document.

% this is a Wiener filter application in denoising MRI with two dimensions
  thresholding
--------------------------------------------------------------------------


I = imread('C:/torso.bmp');
I = rgb2gray(I);
for phi= 0:0.375:1.5
II=I*cos(phi)
J = imnoise(II,'gaussian',0.0,0.0005);
J = imresize(J,[ 480 480])
figure,imshow(J)
K=I*sin(phi)
K = imnoise(K,'gaussian',0.0,0.0005);
K = imresize(K,[ 480 480])
figure,imshow(K)
L=(J+K)*0.75
figure,imshow(L)
end
%Val1=mean2(ID)
%Val2=std2(ID)
h=4
DenoiseDI = ThreshWave2(L,h,1,0.001,4,3,4)
figure ,imshow(DenoiseDI)
Val1=mean2(DenoiseDI)
Val2=std2(DenoiseDI)
[PD,noise] = wiener2(P,[3 3])
figure ,imshow(PD)
Val1=mean2(PD)
Val2=std2(PD)
DenoiseDP = ThreshWave2(P,H,1,0.001,4,3,4)
figure ,imshow(DenoiseDP)
Val1=mean2(DenoiseDP)
Val2=std2(DenoiseDP)
[JD,noise] = wiener2(J,[3 3])
```

```
figure ,imshow(JD)
Val1=mean2(JD)
Val2=std2(JD)
DenoiseDJ = ThreshWave2(J,H,1,0.001,4,3,4)
figure ,imshow(DenoiseDJ)
Val1=mean2(DenoiseDJ)
Val2=std2(DenoiseDJ)
[KD,noise] = wiener2(K,[3 3])
figure ,imshow(KD)
Val1=mean2(KD)
Val2=std2(KD)
DenoiseDK = ThreshWave2(K,H,1,0.001,4,3,4)
figure ,imshow(DenoiseDK)
Val1=mean2(DenoiseDK)
Val2=std2(DenoiseDK)
[LD,noise] = wiener2(L,[3 3])
figure ,imshow(LD)
Val1=mean2(LD)
Val2=std2(LD)
DenoiseDL = ThreshWave2(L,H,1,0.001,4,3,4)
figure ,imshow(DenoiseDL)
Val1=mean2(DenoiseDL)
Val2=std2(DenoiseDL)
```

C LANGUAGE WIENER DETAILS

wienerC(g,varargin)

```
function [f,noise,localVar] = wienerC(g,varargin)
%WIENERC Perform 2D adaptive noise-removal filtering.
%
%  Modified MATLAB Wiener filtering with alternative
%  noise estimation techniques (for additive Gaussian noise)
%
%
%                'Moda':    Estimation with mode
%                'Med':     Estimation with median
%           'Min':     Estimation with minimum
%                'Mad':      Median absolute deviation
%
% if 'inses' is added, the estimation of the variance is done
% over N-1 points
%
% Default estimation window=[3,3]. If 'big', [Wx Wy] is added
% estimation is done over a different window.
.

[nhood, block, noise, msg,estim,inses] = ParseInputs(varargin{:});

if (~isempty(msg))
  error(msg);
end

classin = class(g);
classChanged = 0;
if ~isa(g, 'double')
  classChanged = 1;
```

```
  g = im2double(g);
end

% Estimate the local mean of f.
localMean = filter2(ones(nhood), g) / prod(nhood);
% Estimate of the local variance of f.
localVar = filter2(ones(nhood), g.^2) / prod(nhood) - localMean.^2;

if inses
          localVar=(((nhood(1)*nhood(2))./(nhood(1)*nhood(2))-1).*localVar;
end

% Estimate the noise power if necessary.
if (isempty(noise))
  if estim==2
                    noise = moda(localVar,1000);
                    %if noise<=1.7
                    %         noise=0;
                    %end
  elseif estim==3
                    noise=median(localVar(:));
  elseif estim==4
                    noise=min(localVar(:));
  elseif estim==5
                [CA,CH,CV,CD] =dwt2(g,'haar');
             noise=(1.4826.*median(abs(CH(:)-median(CH(:))))).^2;
  else
                    noise = mean2(localVar);
  end
noise=max(noise,0.0001);
end

% Compute result
% f = localMean + (max(0, localVar - noise) ./ ...
%          max(localVar, noise)) .* (g - localMean);
%
% Computation is split up to minimize use of memory
% for temp arrays.
f = g - localMean;
g = localVar - noise;
g = max(g, 0);
localVar = max(localVar, noise);
f = f ./ localVar;
f = f .* g;
f = f + localMean;

if classChanged,
  f = changeclass(classin, f);
end


%%%-------------------------------------------------------------
%%% Subfunction ParseInputs
%%%

function [nhood, block, noise, msg,estim,inses] = ParseInputs(varargin)
```

```
g = [];
nhood = [3 3];
block = [];
noise = [];
msg = '';
estim=1;
inses=0;
dfsteppos = -1;

for i=1:length(varargin)

        flag =0;
        if i == dfsteppos
        flag = 1;
        end
        if(isstr(varargin{i}))
                if strcmp(varargin{i},'Moda')
                estim=2;
                flag = 1;
                elseif strcmp(varargin{i},'Med')
                        estim=3;
                flag = 1;
                elseif strcmp(varargin{i},'Min')
                        estim=4;
                flag = 1;
                elseif strcmp(varargin{i},'Mad')
                        estim=5;
                flag = 1;
                elseif strcmp(varargin{i},'inses')
                        inses=1;
                flag = 1;
                elseif strcmp(varargin{i},'big')
                        big=1;
                flag = 1;
                        nhood= varargin{i+1};
                dfsteppos = i+1;
                        flag = 1;
                        end
        else
                        nhood = varargin{i};
                        flag = 1;

        end
        if flag == 0
    error('Too many parameters !')
    return
  end
end



% checking if input image is a truecolor image-not supported by WIENER2
%if (ndims(g) == 3)
%    msg = 'WIENER2 does not support 3D truecolor images as an input.';
%    return;
```

```
%end;

%if (isempty(block))
%    block = bestblk(size(g));
%end

function m=moda(u,N)
% MODA   Mode of a distribution
%
%    m=MODE(u,N) calculates the mode of the set of data "u" using the histogram.
%    To avoid outliers, for the calculation are only taken into account those
%    values in [mean-2sigma, mean+2sigma];
%
%    INPUT:
%
%        - u (set of data)
%      - N: Number of points for the histogram. If N=0 then 5000 points are
%           considered
%
%
u=double(u);
if N==0
          N=5000;
end
M1=mean(u(:));
V1=std(u(:));
C2=u( (u(:)>=(M1-2*V1)) & (u(:)<=(M1+2*V1)) ) ;
%C2=u;
[h,x]=hist(C2,N);
[M,M2]=max(h);
m=x(M2);
%
```

FAST BILATERAL FILTER

```
 I = imread('C:/torso.bmp');
for phi= 0:0.375:1.5
II=I*cos(phi)
J = imnoise(II,'gaussian',0.0,0.0005);
J = imresize(J,[ 480 480])
figure,imshow(J)
K=I*sin(phi)
K = imnoise(K,'gaussian',0.0,0.0005);
K = imresize(K,[ 480 480])
figure,imshow(K)
L=(J+K)*0.75
figure,imshow(L)
end
val1 = std2(I)
val2 = mean2(I)
val3 = std2(II)
val4 = mean2(II)
val5 = std2(K)
val6 = mean2(K)
val7 = std2(K)
val8 = mean2(K)
sigmaSxy=val7,
sigmaSz=val7,
```

```
sigmaR=val7
samS=4,
samR=4,

Ibf=bilateral3(I, sigmaSxy,sigmaSz,sigmaR,samS,samR,verbose)
%   sigmaSxy and sigmaSz - spatial smoothing parameters (standard
%   deviation of the Gaussian kernel)
%   sigmaR - the smoothing parameter in the "range" dimension
%   samS - the amount of downsampling performed by the fast
%   approximation in the spatial dimensions (x,y). In z direction, it is
%   derived from spatial sigmas and samS.
%   samR - the amount of downsampling in the "range" dimension.
%   verbose - optional, for debugging

%Notes:
% - the internal function, BILATERAL3I, allows you to vary all parameters,
%   if you want to.
% - spatial sigmas are assumed to be in the units of "pixel".
% - range sigma is assumed to be in the units of range values whose
% minimum is 0 and maximum is 255 (i.e. 8-bit).

function Ibf=bilateral3(I, sigmaSxy,sigmaSz,sigmaR,samS,samR,verbose)
if ~exist('verbose','var'), verbose=0; end
I=range1(double(I),255);
%simplification #1
%smoothing and subsampling in x and y spatial directions is equal
sigmaSx=sigmaSxy;
sigmaSy=sigmaSxy;
samSx=samS;
samSy=samS;

%simplification #2
% obtain samSz by assuming samSz/samS=sigmaSz/sigmaSxy
c=sigmaSz/sigmaSxy;
samSz=ceil(c*samS);

%optional simpilification #3:
% obtain samR by assuming samS/sigmaS = samR/sigmaR
if ~exist('samR','var')
    samR = sigmaR*samS/sigmaSxy;
end

%re-scale sigmaR, samR to normalize them ( [0, 255] --> [0, 1] )
% samR has to be such that 1/samR is an integer
%e.g. if supplied samR is 12 (divide range [0,255] into bins of size 12):
%
%normalize: samR=12/256=0.0469;
%1/samR=21.3333 ~=22 bins (round up).
%the new samR = 1/22 = 0.0455;
%later on, when samR is used, 1/samR will yield the number of range bins,
22.
sigmaR  = sigmaR/256;
samR    = 1/ceil(256/samR);

%for debugging purposes:
```

```
if verbose
    displ('sigmaSx',sigmaSx);
    displ('sigmaSy',sigmaSy);
    displ('sigmaSz',sigmaSz);
    displ('sigmaR',sigmaR);
    displ('samSx',samSx);
    displ('samSy',samSy);
    displ('samSz',samSz);
    displ('samR',samR);
    disp(['{' n2s(size(I,1)) ',' n2s(size(I,2)) ',' n2s(size(I,3)) ...
        ',256} --> {' n2s(ceil(size(I,1)/samSx)) ','
n2s(ceil(size(I,2)/samSy)) ...
        ',' n2s(ceil(size(I,3)/samSz)) ',' n2s(1/samR) '}']);
end

%run the filter
Ibf=bilateral3i(I,sigmaSx,sigmaSy,sigmaSz,sigmaR,samSx,samSy,samSz,samR);

function
Ibf=bilateral3i(I,sigmaSx,sigmaSy,sigmaSz,sigmaR,samSx,samSy,samSz,samR)
I=double(I);
[N M P]=size(I);

%find the number of bins in each spatial dimension
Xbins=ceil(N/samSx);
Ybins=ceil(M/samSy);
Zbins=ceil(P/samSz);

Np=Xbins*samSx;
Mp=Ybins*samSy;
Pp=Zbins*samSz;

Xrem=ceil((Np-N)/2);
Yrem=ceil((Mp-M)/2);
Zrem = ceil((Pp-P)/2);

%zero-pad the image as symmetrically as possible to
%attain a size divisible by the subsample rate
Ip = padarray(I,[Xrem Yrem Zrem],0);
I = Ip(1:Np,1:Mp,1:Pp);

clear Ip

%%%%%
% 1. subsampling in spatial domain
%
% quantize intensity values
% a) map the image I onto a range [0,1].
% b) map the image I onto a range [0,1/sR].
% c) round the values to obtain an image Iq in which each pixel value
% Iq(x0) represents the range bin to which x0 belongs.
Iq=round((I-min(I(:)))/range(I(:))/samR);

%what bin does each pixel belong to, based on its x,y,z coordinate?
[X Y Z]=ndgrid(1:Np,1:Mp,1:Pp);
```

```
binX=floor((X-1)/samSx)+1;
binY=floor((Y-1)/samSy)+1;
binZ=floor((Z-1)/samSz)+1;
clear X Y Z
%find which range bin each location in the x-y-z-range space belongs to
binW=repmat(reshape(1:samSx*samSy*samSz,[samSx samSy samSz]),[Xbins,Ybins
Zbins]);

D=zeros(Xbins,Ybins,Zbins,samSx*samSy*samSz);
W=D;
D(sub2ind(size(D),binX(:),binY(:),binZ(:),binW(:)))=I(:);
W(sub2ind(size(D),binX(:),binY(:),binZ(:),binW(:)))=Iq(:);

clear Iq binX binY binZ binW

%- W represents which 4D bin each data point belongs to
%- D represents what the actual value of each data point is*
%both contain values oriented according to coordinate system:
%(subsampled-x,subsampled-y,subsampled-z,location-in-bin)
%- the ordering of values along the 4th dimension no longer matters after
%this point.

%* (D is simply the input image I suitably rearranged. numel(D)=numel(I).)
%if I was a 2D image of size 12 by 12, and bin size in X and Y was 3
pixels
%each,constructing D would amount to breaking the image into 3-by-3
chunks,
%then taking the 9 pixels in each chunk and arranging them in a vector,
%placing these vectors next to each other to create a 4-by-4-by-9
%rearrangement of I.

GData       =zeros(Xbins,Ybins,Zbins,1/samR);
GWeights    =zeros(Xbins,Ybins,Zbins,1/samR);

%for each range bin:
for k=1:(1/samR)
    %find the indices of pixels belonging to this range bin
    tmp  = (W==k);
    %make a copy of D but zero out all values except those belonging to
    %k-th bin
    tmp2 = zeros(size(D));
    tmp2(tmp)=D(tmp);
    %GWeights will contain the number of pixels in k-th range bin for each
    %x-y-z bin
    GWeights(:,:,:,k) = sum(tmp,4);
    %And GData will contain the sum of their values
    GData(:,:,:,k)    = sum(tmp2,4);
end

%2. Smoothing with gaussians
hSx=gkernel(sigmaSx/samSx);
hSy=gkernel(sigmaSy/samSy);
hSz=gkernel(sigmaSz/samSz);
hR=gkernel(sigmaR/samR);
```

```
GWeights=convnsep({hSx,hSy,hSz,hR},GWeights,'same');
GData=convnsep({hSx,hSy,hSz,hR},GData,'same');

%calculate coordinates in the non-subsampled X-Y-Z-Range plain
%to which filtered bin values belong
[n m p k]=size(GData);
Xloc=repmat(linspace(1,n,Np)',[1 Mp Pp]);
Yloc=repmat(linspace(1,m,Mp), [Np 1 Pp]);
Zloc=repmat(shiftdim(linspace(1,p,Pp),-1),   [Np Mp 1]);
Rloc=(I-min(I(:)))/range(I(:))*(1/samR-1)+1;

%compute the pixel values that will go to locations computed above
Indx=GWeights~=0;
Data=zeros(size(GData));
Data(Indx)=GData(Indx)./GWeights(Indx);

clear GData GWeights Indx

%At this point, values at evenly spaced locations Xloc,Yloc,Zloc are
%known (stored in Data), but for the rest of the image they are unknown.
%Interpolation will find them.

Ibf=interpn(Data,Xloc,Yloc,Zloc,Rloc);

clear Data Xloc Yloc Zloc Rloc

%remove zero-padding
Ibf=Ibf(Xrem+1:N+Xrem,Yrem+1:M+Yrem,Zrem+1:P+Zrem);

%Auxiliary functions:

%displaying variable value
function displ(name,val)
disp([name '=' n2s(val) ';']);

%shorthand for num2str
function S = n2s(N,f)
if N~=fix(N)
    if ~exist('f','var'),S=num2str(N,'%4.2f');
    elseif f==0
        S=num2str(N);
    else
        S=num2str(N,f);
    end
else
    S=num2str(N);
end

%make a gaussian kernel
function [h,L] = gkernel(sd)
L=ceil(3.5*sd);
h=exp(-0.5*((-L:L)'/sd).^2);
h=h/sum(h);
```

```
function Y = range1(x,rg)
if ~exist('rg','var'),rg=1;end
Y=1/range(x(:))*(x-min(x(:)))*rg;

function y = range(x)
y=max(x)-min(x);
```

PHASE DENOISE

```
A= imread('C:/torso.bmp');
w=5
sigma_d=2
sigma_r=2
[X,Y] = meshgrid(-w:w,-w:w);
G = 2.71^(-(X.^2+Y.^2)/(2*sigma_d^2));

I = rgb2gray(A);
for phi= 0:0.375:1.5
II=I*cos(phi)
J = imnoise(II,'gaussian',0.0,0.0005);
J = imresize(J,[ 480 480])
figure,imshow(J)
K=I*sin(phi)
K = imnoise(K,'gaussian',0.0,0.0005);
K = imresize(K,[ 480 480])
figure,imshow(K)
L=(J+K)*0.75
figure,imshow(L)
end
val1 = std2(I)
val2 = mean2(I)
val3 = std2(II)
val4 = mean2(II)
val5 = std2(K)
val6 = mean2(K)
val7 = std2(K)
val8 = mean2(K)

function [x, mask0] = mri_phase_denoise(yi, varargin)
%|function [x, mask0] = mri_phase_denoise(yi, [options]) <- recommended
usage
%|function [x] = mri_phase_denoise(yi, l2b, niter, chat, wthresh) <- old
way
%|
%| in
%|  yi  [(N)]   noisy complex image: y = mag .* exp(1i * x)
%|          can be any size: 1D, 2D, 3D, ...
%|
%| options
%|  l2b     log_2(beta), regularization parameter
%|  order       regularization order (default: 1, for historical
%|              reasons, but 2 is probably preferable)
%|  niter       # of iterations
%|  chat        1 to show pictures
%|  wthresh     fraction of magnitude maximum to include in fitting
%|  init        initial image for iterations
```

```
%|  isave       which iterations to save.  (default: last)
%|  'pl'    1|0 1 for new PL method (recommend), 0 for old way (default)
%| out
%|  x   [(N)]   cleaned up phase estimate
%|  mask0   [(N)]   logical: 1 for high-magnitude pixels

if nargin < 1, help(mfilename), error(mfilename), end
if ischar(yi)
    [x mask0] = mri_phase_denoise_test(yi, varargin{:});
    if ~nargout, clear x mask0, end
return
end

% defaults
arg.chat = 0;
arg.l2b = -5;
arg.order = 1;
arg.niter = 150;
arg.isave = [];
arg.wthresh = 0.4;
arg.fmax = 0.05; % fraction of max threshold for trimmed median for wi_ml
arg.init = [];
arg.clim = []; % limits for phase display
arg.pl = false; % PL
arg.wi_ml = false; % use wi based on ML instead of threshold

% backword compatible for old argument list: l2b, niter, chat, wthresh
if length(varargin) && isnumeric(varargin{1})
    arg.l2b = varargin{1};
    if length(varargin) >= 2, arg.niter = varargin{2}; end
    if length(varargin) >= 3, arg.chat = varargin{3}; end
    if length(varargin) >= 4, arg.wthresh = varargin{4}; end
else
    arg = vararg_pair(arg, varargin);
end
if isempty(arg.isave), arg.isave = arg.niter; end

mag = abs(yi);
yi = angle(yi);
if arg.chat
    im clf, im pl 2 3
    im(1, mag, 'magnitude'), cbar
    im(2, yi, 'raw phase map', arg.clim), cbar
end

dim_yi = size(yi);

%
% specify weights: this needs more work to be automatic!
%
mask0 = mag > arg.wthresh * max(mag(:)); % ignore pixels with "too small"
magnitude
if arg.chat
    im(3, mask0, 'weights'), cbar
    im(4, mask0 .* yi, 'masked phase', arg.clim), cbar
end
```

```
%mean(mag(:))
%median(mag(:))
%clf, hist(mag(:), 100), pause
%median(mag(mag(:) > 0.05 * max(mag(:))))

if arg.wi_ml
    wi = mri_phase_wi_ml(mag(:), arg.fmax);
else
    wi = mask0(:);
end
W = diag_sp(wi);

%
% initial phase image
%
if isempty(arg.init)
    arg.init = yi;
    arg.init(mask0 == 0) = mean(yi(mask0 == 0));
end
if arg.chat
    im(5, arg.init, 'Initial phase', arg.clim), cbar
end

G = diag_sp(ones(prod(dim_yi),1));

%
% regularizer
%
if arg.order ~= 2, warn('order=2 recommended'), end
mask1 = true(size(yi)); % estimate / extrapolate to *all* pixels
R = Reg1(mask1, 'beta', 2^arg.l2b, 'order', arg.order);
%R = Robject(mask1, 'beta', 2^arg.l2b, 'order', arg.order);
%   'type_denom', 'matlab', ...

if 0 % old way
    [C wjk] = C2sparse('tight', mask1, 8); % todo: cut?
    C = spdiag(sqrt(wjk), 'nowarn') * C; % caution: missing prior to 2005-
11-28
    C = sqrt(2^arg.l2b) * C;
end

% report expected blur (at image center)
if 1
    qpwls_psf(G, R, 1, mask1, W);
end

%
% run qpwls algorithm for regularized fitting
%
xinit = arg.init(mask1);
if arg.pl
    med = median(mag(mag(:) > 0.05 * max(mag(:))));
    data = {yi(:), (mag(:)/med).^2}; handle = @phase_dercurv; % PL
%profile on % todo!
```

```
    x = pl_pcg_qs_ls(xinit, G, data, handle, R, ...
        'niter', arg.niter, 'isave', arg.isave);
%profile report
    x = embed(x, mask1);
else
    warn 'recommend using "pl" option.  use wls for historical only'
%   data = {yi(:), wi(:)}; handle = @wls_dercurv; % qpwls
    x = qpwls_pcg1(xinit, G, W, yi(:), R.C, ...
        'niter', arg.niter, 'isave', arg.isave);
    x = embed(x, mask1);
end

if 0 % old way
    x = qpwls_pcg(x, G, W, yi(:), 0, R.C, 1, arg.niter);
    x = reshape(x, [dim_yi arg.niter]);
    x = x(:,:,end);
end

if arg.chat
    if ndims(yi) == 2
        im(6, x(:,:,end), 'QPWLS-CG phase', arg.clim), cbar
    else % 3d
        im(6, x(:,:,:,end), 'QPWLS-CG phase', arg.clim), cbar
    end
end



%
% mri_phase_wi_ml()
% wi based on ML estimation
% trick: normalize by median of non-background so that beta is "universal"
%
function wi = mri_phase_wi_ml(mag, fmax)
med = median(mag(mag(:) > fmax * max(mag(:))));
wi = (mag(:) / med).^2;



%
% phase_dercurv()
% wi * (1 - cos(yi - li))
%
function [deriv, curv] = phase_dercurv(data, li, varargin)
yi = data{1};
wi = data{2};
deriv = wi .* sin(li - yi);
curv = wi;



%
% built-in test/example
%
function [xq, mask0] = mri_phase_denoise_test(type, varargin)

% read data
f.dir = path_find_dir('mri');
```

```
f.dir = [f.dir '/phase-data/'];
f.dat = [f.dir 'phfit.mat'];
if ~exist(f.dat, 'file')
    fail('edit the path in %s!', mfilename)
end
yi = mat_read(f.dat);
clim = [-0.5 1.5];

%if nargout
%   order = 1;
%else
%   order = 2;
%end
[xq mask0] = mri_phase_denoise(yi, ...
    'clim', clim, 'init', [], 'chat', 1, varargin{:});
%   'init', 5*randn(size(yi)));
%   'init', 5*ones(size(yi)));
xq = xq(:,:,end);
if im
    title 'QPWLS-CG phase (simple wi)'
end

cpu etic
xpl = mri_phase_denoise(yi, 'pl', 1, varargin{:});
cpu etoc 'PL time'
im(4, xpl, 'PL-CG phase', clim), cbar

cpu etic
xml_qpwls = mri_phase_denoise(yi, 'wi_ml', 1, varargin{:});
cpu etoc 'PWLS time'
im(5, xml_qpwls, 'QPWLS-CG (ML wj)', clim), cbar

max_percent_diff(xpl, xml_qpwls)
%max_percent_diff(xpl, xq)
nrms(xml_qpwls, xpl)
nrms(xq, xpl)
%im(4, xpl-xq), cbar

%savefig fig_mr_phase_pl
```

FAST LMMSE filter

```
Im=L
Ws=[7,7]
function I_est=MRI_lmmse(Im,Ws,varargin)

%MRI_LMMSE Linear minimum Mean Square error Estimation of MRI data
%
% Filter assumes a Rician distribution with a Rayleigh Background
% Noise estimation is automatically performed using the background.
%
% Usage
%
%   I_est=MRI_lmmse(Im,[7,7],[Noise method]);
%
```

```
%
% DEFAULT:
%
%   I_est=MRI_lmmse(Im,[7,7]);
%        Noise estimationmethod: mode2N
%
%   INPUTS:
%        - Im: input image
%        - Ws: Size [M,N] of the square window used for local estimation
%              Odd number recomended
%
%  Noise Estimation Method:
%       y= MRI_lmmse(...,'sigma',sigma) Standard deviation of noise given
%                  - sigma: Standar deviation of noise
%       y= MRI_lmmse(...,'bckN2',threshold) Estimate the noise from the
%              background of the image based on order 2 moment
%                  - threshold: Threshold value for the mask
%       y= MRI_lmmse(...,'bckNm',threshold) Estimate the noise from the
%              background of the image based on the mean
%                  - threshold: Threshold value for the mask
%       y= MRI_lmmse(...,'momentN') Estimate the noise using the methods of
%              moments
%
%   The following methods may use a background mask
%
%       y= MRI_lmmse(...,'mask',threshold) A background mask is used.
%                  - threshold: Threshold value for the mask
%       y= MRI_lmmse(...,'histoN') Estimate the noise using the method
%              based on the mode of the histogram [Sijbers06]
%       y= MRI_lmmse(...,'mode2N') Estimate the noise using the mode of the
%              local order 2 moment [Aja06]
%       y= MRI_lmmse(...,'modeMN') Estimate the noise using the mode of the
%              local mean [Aja06]
%       y= MRI_lmmse(...,'modeVN') Estimate the noise using the mode of the
%              local variance [Aja06]
%       y= MRI_lmmse(...,'modeVN_NI') Estimate the noise using the mode of the
%  local mean [Aja06]


[mask,thresM,noise,sigma] = parse_inputs(varargin{:});

%Noise Estimation----------------------------------------
%LOCAL STATISTICS
%Order 2 moment
En=filter2(ones(Ws), Im.^2) / (prod(Ws));
%Mean
Mn=filter2(ones(Ws), Im) / (prod(Ws));
%Variance
Vn=(prod(Ws)/(prod(Ws)-1)).*(En-Mn.^2);


if noise==0
%Sigma given
   sigma2=sigma.^2;
elseif noise==1
   mask =im2bw(1-double(imfill(Im>thresM,'holes')));
```

```
    sigma2=0.5.*(sum((Im(mask)).^2))./sum(mask(:));
    sigma=sqrt(sigma2);
elseif noise==2
    mask =im2bw(1-double(imfill(Im>thresM,'holes')));
    sigma=sqrt(2/pi).*(sum(Im(mask))./sum(mask(:));
    sigma2=sigma^2;
elseif noise==3
    M2=mean(Im(:).^2);
    M4=mean(Im(:).^4);
    sigma2=0.5.*(M2-sqrt(sqrt(abs(2*M2^2-M4))));
    sigma=sqrt(sigma2);
elseif noise==4
    if mask==0
        I2=round(Im);
        Tp=max(I2(:));
        Tpm=min(I2(:));
        [h,x]=hist(I2(:),Tp);
        sigma=x(argmax(h));
    else
       mask =im2bw(1-double(imfill(Im>thresM,'holes')));
       Tp=max(I2(mask));
       [h,x]=hist(I2(mask),Tp);
       sigma=x(argmax(h));
    end
    sigma2=sigma^2;
elseif noise==5
    if mask==0
        sigma2=(prod(Ws)/(prod(Ws)-1)).*(moda(En,1000)./2);
    else
       mask =im2bw(1-double(imfill(Im>thresM,'holes')));
       sigma2=(prod(Ws)/(prod(Ws)-1)).*(moda(En(mask),1000)./2);
    end
    sigma=sqrt(sigma2);
elseif noise==6
    if mask==0
        sigma=sqrt(2/pi).*moda(Mn,1000);
    else
        mask =im2bw(1-double(imfill(Im>thresM,'holes')));
        sigma=sqrt(2/pi).*moda(Mn(mask),1000);
    end
    sigma2=sigma^2;
elseif noise==7
    if mask==0
        sigma2=((2/(4-pi)).*moda(Vn,1000));
    else
        mask =im2bw(1-double(imfill(Im>thresM,'holes')));
        sigma2=(2/(4-pi)).*moda(Vn(mask),1000);
    end
    sigma=sqrt(sigma2);
elseif noise==8
    if mask==0
        sigma2=((prod(Ws)-1)/(prod(Ws)-3)).*moda(Vn,1000);
    else
        mask =im2bw(double(imfill(Im>thresM,'holes')));
        sigma2=((prod(Ws)-1)/(prod(Ws)-3)).*moda(Vn,1000);
    end
```

```
   sigma=sqrt(sigma2);
end

%End Noise estimation--------------------------------------------


%FILTERING--------------------------------------------

Qua=filter2(ones(Ws),Im.^4)./prod(Ws);
%Squ=filter2(ones(Ws),Im.^2)./prod(Ws);
Squ=En;
%Squ=Squ.*(Squ>2.*sigma2)+(Squ<=2.*sigma2).*2.*sigma2;
%Qua=Qua.*(Qua>8.*sigma2^2)+(Qua<=8.*sigma2^2).*8.*sigma2^2;


K1=1+(4.*sigma2^2-4.*sigma2.*Squ)./(Qua-Squ.^2);
K1=max(K1,0);
I_est=sqrt(Squ-2.*sigma2+K1.*(Im.^2-Squ));
%I_est= sqrt(Im.^2-2.*sigma2+0.5.*(1-K1).*(Squ-Ac.^2));
I_est=abs(I_est);


%---------------------------------------------------------------------
function [mask,thresM,noise,sigma] = parse_inputs(varargin)
dfsteppos = -1;
mask=0;
thresM=0;
noise=5;
sigma=0;



for i = 1 : length(varargin)
  flag = 0;
  if i == dfsteppos
    flag = 1;
  end
  if strcmp(varargin{i},'mask')
     mask=1;
    thresM = varargin{i+1};
    flag = 1;
    dfsteppos = i+1;
   elseif strcmp(varargin{i},'sigma')
    noise=0;
    sigma = varargin{i+1};
    flag = 1;
    dfsteppos = i+1;
  elseif strcmp(varargin{i},'bckN2')
    noise=1;
    thresM = varargin{i+1};
    flag = 1;
    dfsteppos = i+1;
  elseif strcmp(varargin{i},'bckNm')
    noise=2;
    thresM = varargin{i+1};
    flag = 1;
```

```
    dfsteppos = i+1;
  elseif strcmp(varargin{i},'momentN')
    noise=3;
    flag = 1;
   elseif strcmp(varargin{i},'histoN')
    noise=4;
    flag = 1;
  elseif strcmp(varargin{i},'mode2N')
    noise=5;
    flag = 1;
  elseif strcmp(varargin{i},'modeMN')
    noise=6;
    flag = 1;
  elseif strcmp(varargin{i},'modeVN')
    noise=7;
    flag = 1;
   elseif strcmp(varargin{i},'modeVN_NI')
    noise=8;
    flag = 1;
   end
  if flag == 0
    error('Too many parameters !')
    return
  end
end

%---------------------------------------------------------------
function m=moda(u,N)
% MODA   Mode of a distribution
%
%    m=MODE(u,N) calculates the mode of the set of data "u" using the histogram.
%    To avoid outliers, for the calculation are only taken into account those
%    values in [mean-2sigma, mean+2sigma];
%
%    INPUT:
%
%         - u (set of data)
%       - N: Number of points for the histogram. If N=0 then 5000 points are
%            considered
%
u=double(u);
if N==0
           N=5000;
end
M1=mean(u(:));
V1=std(u(:));
C2=u( (u(:)>=(M1-2*V1)) & (u(:)<=(M1+2*V1)) ) ;
%C2=u;
[h,x]=hist(C2,N);
[M,M2]=max(h);
m=x(M2);
%
```

NOWAK2

```
========================================================================
% Rician noise generation using two orthogal Gaussian  generators
% copyright © 2016 Kenneth Kagoiya
% Permission to use, copy or modify this software and its document.

% This program implement Nowak2 filter denoising and applies both soft and
hard denoising
------------------------------------------------------------------------
h=4
Type=0
option=[0 3.6 0 1 0 0]T
%option = [0 3.6 0 1 0 0]

function [xd,xn,option] = denoise(L,h,type,option)

if(nargin < 2)
  error('You need to provide at least 2 inputs: x and h');
end;
if(nargin < 3),
  type = 0;
  option = [];
elseif(nargin < 4)
  option = [];
end;
if(isempty(type)),
  type = 0;
end;
if(type == 0),
  default_opt = [0 3.0 0 0 0 0];
elseif(type == 1),
  default_opt = [0 3.6 0 1 0 0];
else,
  error(['Unknown denoising method',10,...
      'If it is any good we need to have a serious talk :-)']);
end;
option = setopt(option,default_opt);
[mx,nx] = size(x);
dim = min(mx,nx);
if(dim == 1),
  n = max(mx,nx);
else,
  n = dim;
end;
if(option(5) == 0),
  L = floor(log2(n));
else
  L = option(5);
end;
if(type == 0),          % Denoising by DWT
  xd = mdwt(x,h,L);
  if (option(6) == 0),
    tmp = xd(floor(mx/2)+1:mx,floor(nx/2)+1:nx);
    if(option(3) == 0),
      thld = option(2)*median(abs(tmp(:)))/.67;
    elseif(option(3) == 1),
```

```
      thld = option(2)*std(tmp(:));
    else
      error('Unknown threshold estimator, Use either MAD or STD');
    end;
  else,
    thld = option(6);
  end;
  if(dim == 1)
    ix = 1:n/(2^L);
    ykeep = xd(ix);
  else
    ix = 1:mx/(2^L);
    jx = 1:nx/(2^L);
    ykeep = xd(ix,jx);
  end;
  if(option(4) == 0),
    xd = SoftTh(xd,thld);
  elseif(option(4) == 1),
    xd = HardTh(xd,thld);
  else,
    error('Unknown threshold rule. Use either Soft (0) or Hard (1)');
  end;
  if (option(1) == 0),
    if(dim == 1),
      xd(ix) = ykeep;
    else,
      xd(ix,jx) = ykeep;
    end;
  end;
  xd = midwt(xd,h,L);
elseif(type == 1),          % Denoising by UDWT
  [xl,xh] = mrdwt(x,h,L);
  if(dim == 1),
    c_offset = 1;
  else,
    c_offset = 2*nx + 1;
  end;
  if (option(6) == 0),
    tmp = xh(:,c_offset:c_offset+nx-1);
    if(option(3) == 0),
      thld = option(2)*median(abs(tmp(:)))/.67;
    elseif(option(3) == 1),
      thld = option(2)*std(tmp(:));
    else
      error('Unknown threshold estimator, Use either MAD or STD');
    end;
  else,
    thld = option(6);
  end;
  if(option(4) == 0),
    xh = SoftTh(xh,thld);
    if(option(1) == 1),
      xl = SoftTh(xl,thld);
    end;
  elseif(option(4) == 1),
    xh = HardTh(xh,thld);
    if(option(1) == 1),
```

```
      xl = HardTh(xl,thld);
    end;
  else,
    error('Unknown threshold rule. Use either Soft (0) or Hard (1)');
  end;
  xd = mirdwt(xl,xh,h,L);
else,                    % Denoising by unknown method
  error(['Unknown denoising method',10,...
         'If it is any good we need to have a serious talk :-)']);
end;
option(6) = thld;
option(7) = type;
xn = x - xd;
```

Dynamic non-local means (DNLM)

```
function fimg = dnlm(img, options)

% FIMG = DNLM(IMG, OPTIONS)
%
% Filters a 4D image using dynamic non-local means
%
%INPUT
% img is a 4D image
% options is a structure containing:
% options.k is a vector, holding the radius of the comparison window for each dimension
%         if a single value is used an isotropic window will be used
% options.sig is the estimated noise standard deviation. If not given it
%           will be automatically estimated using the method from "Noise Reduction for Magnetic
Resonance Images via
%           Adaptive Multiscale Products Thresholding", Paul Bao, Lei Zhang, 2003.
% options.beta is the denoising factor. If not given, it will be set to be 1
%        note that h^2 = 2 * beta * sig^2 * n_of_neighbours (according to the similarity window size)
% options.dstsig is the std, in pixels, to use for distance weighting (default is max(k)/2)
% options.win is a vector , holding the size of search radius in each dimension
%         if a single value is used an isotropic search area will be used
% options.ancor is a scalar, 1 for classic NL-Means, 2 (default) for DNLM
%            ancor=0 will use DNLM without noise thresholding (i.e. will denoise even differences
smaller than sig)
%
%OUTPUT
% A denoised image
%
size_x = size(img,1);
size_y = size(img,2);
size_z = size(img,3);
size_w = size(img,4);

fimg = zeros(size_x, size_y, size_z, size_w);

if length(options.k) == 1
   k = [options.k options.k options.k options.k];
else
   k = options.k;
end
```

```
if isfield(options, 'dstsig')
   dst_sig2 = options.dstsig * options.dstsig;
else
   dst_sig2 = max(k)*max(k)/4;
end

if isfield(options, 'ancor')
   ancor = options.ancor; % 0=Old DNLM, 1=ENLM, 2=New DNLM
else
   ancor = 2; % new DNLM
end

if isfield(options, 'win')
   if length(options.win) == 1
      win = [options.win options.win options.win options.win];
   else
      win = options.win;
   end
else
   win = [0 0 0 size(img,4)];
end

if isfield(options, 'sig')
   sig = options.sig;
else
   est_noise = 0;
   est_count = 0;
   for w=1:size_w
      for z=1:size_z
         [~, ~, est_MAV] = estimate_noise_dwt(img(:,:,z,w));
         est_noise = est_noise + est_MAV;
         est_count = est_count + 1;
      end
   end
   sig = est_noise / est_count;
end

if isfield(options, 'beta')
   sig2 = 2 * options.beta * sig * sig * (prod(2*k+1)-1); % = 2 * beta * noise^2 * n_of_neighbours
else
   sig2 = 2 * sig * sig * (prod(2*k+1)-1); % = 2 * noise^2 * n_of_neighbours
end

% Zero padding
size_px = size_x + 2 * k(1);
size_py = size_y + 2 * k(2);
size_pz = size_z + 2 * k(3);
size_pw = size_w + 2 * k(4);
pimg = zeros(size_px, size_py, size_pz, size_pw);
pimg((k(1)+1):(k(1)+size_x),(k(2)+1):(k(2)+size_y),(k(3)+1):(k(3)+size_z),(k(4)+1):(k(4)+size_w))=img;

% Building a distance Gaussian filter
dist_flt = zeros(k(1)*2+1, k(2)*2+1, k(3)*2+1, k(4)*2+1);
for wx=1:(k(1)*2+1)
   for wy=1:(k(2)*2+1)
      for wz=1:(k(3)*2+1)
```

```
          for ww=1:(k(4)*2+1)
             dst = norm([wx-k(1)-1 wy-k(2)-1 wz-k(3)-1 ww-k(4)-1]);
             dist_flt(wx,wy,wz,ww)=exp(-dst*dst/dst_sig2);
          end
       end
    end
end
dist_flt(k(1)+1,k(2)+1,k(3)+1,k(4)+1)=dist_flt(k(1),k(2)+1,k(3)+1,k(4)+1); % Reduce central weight to
avoid over-weighting
dist_flt = dist_flt / sum(dist_flt(:));
%dist_flt = dist_flt / prod(2*k+1); % Normalise by number of pixels

% pixel external loops
for w=(k(4)+1):(k(4)+size_w)
   for z=(k(3)+1):(k(3)+size_z)
      for y=(k(2)+1):(k(2)+size_y)
         for x=(k(1)+1):(k(1)+size_x)
            pix_win = pimg((x-k(1)):(x+k(1)), (y-k(2)):(y+k(2)), (z-k(3)):(z+k(3)), (w-k(4)):(w+k(4)));
            mean_pix_win = mean(pix_win(:));
            win_len = length(pix_win(:));
            Zi = 0;
            Wval_sum = 0;

            % internal search loops
            for ww=max(w-win(4),k(4)+1):min(w+win(4),k(4)+size_w)
               for zz=max(z-win(3),k(3)+1):min(z+win(3),k(3)+size_z)
                  for yy=max(y-win(2),k(2)+1):min(y+win(2),k(2)+size_y)
                     for xx=max(x-win(1),k(1)+1):min(x+win(1),k(1)+size_x)
                        comp_win = pimg((xx-k(1)):(xx+k(1)), (yy-k(2)):(yy+k(2)), (zz-k(3)):(zz+k(3)), (ww-
k(4)):(ww+k(4)));
                        mean_comp_win = mean(comp_win(:));
                        if (mean_comp_win > 0)
                           if ww ~= w && ancor ~= 1
                           %if ww ~= w && ancor ~= 1 && 2*mean_pix_win/sig >= 3 &&
2*mean_comp_win/sig >= 3
                              diff_win = comp_win-pix_win;
                              diff_mean = sum(abs(diff_win(:)))/win_len;
                              if diff_mean > sig || ancor == 0 % old DNLM
                                 comp_win_factor = mean_pix_win / mean_comp_win; % original factor
                                 comp_win = comp_win * comp_win_factor;
                              else
                                 comp_win_factor = 1;
                              end
                           else
                              comp_win_factor = 1;
                           end

                           diff_win = comp_win-pix_win;
                           diff_nrm2 = sum(dist_flt(:).*diff_win(:).*diff_win(:)); % norm2 convolved with a
Gaussian
                           nb_Wij = exp(-diff_nrm2/sig2);

                           Zi = Zi + nb_Wij;
                           Wval_sum = Wval_sum + (nb_Wij*pimg(xx,yy,zz,ww)*comp_win_factor);
                        end
                     end % closing internal search loops
```

```
            end % closing internal search loops
          end % closing internal search loops
        end % closing internal search loops

        if (Zi > 0)
          fimg(x-k(1),y-k(2),z-k(3),w-k(4)) = Wval_sum / Zi; % normalising
        else
          fimg(x-k(1),y-k(2),z-k(3),w-k(4)) = 0; % normalising
        end

      end % closing pixel external loops
    end % closing pixel external loops
  end % closing pixel external loops
end
CoherenceFilter(u,Options)
```

ANISOTROPIC FILTER

```
function u = CoherenceFilter(u,Options)
% This function COHERENCEFILTER will perform Anisotropic Diffusion of a
% 2D gray/color image or 3D image volume, Which will reduce the noise in
% an image while preserving the region edges, and will smooth along
% the image edges removing gaps due to noise.
%
% Don't forget to compile the c-code by executing compile_c_files.m
%
% Iout = CoherenceFilter(Iin, Options)
%
% inputs,
%   Iin : 2D gray/color image or 3D image volume. Use double datatype in
2D
%            and single data type in 3D.  Range of image data must
%            be approximately [0 1]
%   Options : Struct with filtering options
%
% outputs,
%   Iout : The anisotropic diffusion filtered image
%
% Options,
%   Options.Scheme :  The numerical diffusion scheme used
%                     'R', Rotation Invariant, Standard Discretization
%                          (implicit) 5x5 kernel (Default)
%                     'O', Optimized Derivative Kernels
%                     'I', Implicit Discretization (only works in 2D)
%                     'S', Standard Discretization
%                     'N', Non-negativity Discretization
%   Options.T  :      The total diffusion time (default 5)
%   Options.dt :      Diffusion time stepsize, in case of scheme H,R or I
%                     defaults to 1, in case of scheme S or N defaults to
%                     0.15.
%   Options.sigma :   Sigma of gaussian smoothing before calculation of
the
%                     image Hessian, default 1.
%   Options.rho :     Rho gives the sigma of the Gaussian smoothing of the
%                     Hessian, default 1.
%   Options.verbose : Show information about the filtering, values :
%                     'none', 'iter' (default) , 'full'
```

```
%   Options.eigenmode : There are many different equations to make an
diffusion tensor,
%                       this value (only 3D) selects one.
%                       0 (default) : Weickerts equation, line like kernel
%                       1 : Weickerts equation, plane like kernel
%                       2 : Edge enhancing diffusion (EED)
%                       3 : Coherence-enhancing diffusion (CED)
%                       4 : Hybrid Diffusion With Continuous Switch (HDCS)
%
% Constants which determine the amplitude of the diffusion smoothing in
% Weickert equation
%   Options.C :     Default 1e-10
%   Options.m :     Default 1
%   Options.alpha : Default 0.001
% Constants which are needed with CED, EED and HDCS eigenmode
%   Options.lambda_e : Default 0.02, planar structure contrast
%   Options.lambda_c : Default 0.02, tube like structure contrast
%   Options.lambda_h : Default 0.5 , treshold between structure and noise
%
%
% The basis of the method used is the one introduced by Weickert:
%   1, Calculate Hessian from every pixel of the gaussian smoothed input
image
%   2, Gaussian Smooth the Hessian, and calculate its eigenvectors and
values
%      (Image edges give large eigenvalues, and the eigenvectors
corresponding
%         to those large eigenvalues describe the direction of the edge)
%   3, The eigenvectors are used as diffusion tensor directions. The
%      amplitude of the diffusion in those 3 directions is determined
%      by equations below.
%   4, An Finite Difference scheme is used to do the diffusion
%   5, Back to step 1, till a certain diffusion time is reached.
%
% Weickert equation 2D:
%    lambda1 = alpha + (1 - alpha)*exp(-C/(mu1-mu2).^(2*m));
%    lambda2 = alpha;
%
% 0 : 3D, Weickerts equation, plane line like kernel
%    lambda1 = alpha + (1 - alpha)*exp(-C/(mu1-mu3).^(2*m));
%    lambda2 = alpha;
%    lambda3 = alpha;
%   (with mu1 the largest eigenvalue and mu3 the smallest)
% 1 : 3D, Weickerts equation, plane line like kernel
%    lambda1 = alpha + (1 - alpha)*exp(-C/(mu1-mu3).^(2*m));
%    lambda2 = alpha + (1 - alpha)*exp(-C/(mu2-mu3).^(2*m));
%    lambda3 = alpha;
%   (with mu1 the largest eigenvalue and mu3 the smallest)
% 2 : 3D, Edge Enhancing diffusion
%    lambda3e = 1;
%    lambda2e = 1;
%    lambda1e = 1 - exp(-3.31488 / (Gradient_Magnitude_Squared /
lambda_e^2)^4);
% 3 : 3D, Coherence Enhancing diffusion
%    lambda1c = alpha + (1 - alpha)*exp(-
ln(2)*lambda_c^2/(mu2/(alpha+mu3))^4));
%    lambda2c = alpha;
```

```
%    lambda3c = alpha;
% 4 :  Hybrid Diffusion With Continuous Switch
%   Xi : = (mu1 / (alpha+mu2)) - (mu2 / (alpha+mu3))
%   epsilon = exp ( mu2*(lambda_h^2(Xi-abs(Xi)-2*mu3) )/ ( 2 * lambda_h^4)
)
%   lambda1 = (1 -epsilon) * lambda1c + epsilon *lambda1e;
%   lambda2 = (1 -epsilon) * lambda2c + epsilon *lambda2e;
%   lambda3 = (1 -epsilon) * lambda3c + epsilon *lambda3e;
%
% Notes:
% - The standard and non-negative discretization only allow small time
%   steps before they become unstable. The Implicit discretization
%   was introduced to allow larger diffusion time steps.
%   Previous schemes were not rotational invariant, under certain angles
%   edges blur away. Thus Weickert introduced a rotational invariant
scheme.
%   His scheme sufferes from checkerboard artifacts, due to the central
%   differences used. This code contains our own improved version of his
%   scheme in which  the data is upsampled before calculating the image
%   derivatives for the diffusion flux, to prevent those checkerboard
artifacts.
%
% - If the time step is choosing to large the scheme becomes unstable,
this
%   can be seen by setting verbose to 'full'. The image variance has to
%   decrease every itteration if the scheme is stable.
%
% Literature used
%  - Weickert : "A Scheme for Coherence-Enhancing Diffusion Filtering
%                   with Optimized Rotation Invariance"
%  - Mendrik et al, "Noise Reduction in Computed Tomography Scans Using
%                   3-D Anisotropic Hybrid Diffusion With Continuous
%                   Switch", October 2009
%  - Weickert : "Anisotropic Diffusion in Image Processing", Thesis 1996
%  - Laura Fritz : "Diffusion-Based Applications for Interactive Medical
%                   Image Segmentation"
%  - Siham Tabik, et al. : "Multiprocessing of Anisotropic Nonlinear
%                           Diffusion for filtering 3D image"
%
% example 2d,
%   I = im2double(imread('images/sync_noise.png'));
%   JS = CoherenceFilter(I,struct('T',15,'rho',10,'Scheme','S'));
%   JN = CoherenceFilter(I,struct('T',15,'rho',10,'Scheme','N'));
%   JR = CoherenceFilter(I,struct('T',15,'rho',10,'Scheme','R'));
%   JI = CoherenceFilter(I,struct('T',15,'rho',10,'Scheme','I'));
%   JO = CoherenceFilter(I,struct('T',15,'rho',10,'Scheme','O'));
%   figure,
%   subplot(2,3,1), imshow(I), title('Before Filtering');
%   subplot(2,3,2), imshow(JI), title('Standard Scheme');
%   subplot(2,3,3), imshow(JN), title('Non Negative Scheme');
%   subplot(2,3,4), imshow(JI), title('Implicit Scheme');
%   subplot(2,3,5), imshow(JR), title('Rotation Invariant Scheme');
%   subplot(2,3,6), imshow(JO), title('Optimized Scheme');
%
% example 2d, color HDCS 2D not in literature
%   I = im2double(imread('images/lena.jpg'));
%   I = I+(rand(size(I))-0.5)*0.3;
```

```
%   JO =
CoherenceFilter(I,struct('T',1,'dt',0.1,'rho',4,'Scheme','O','eigenmode',0
));
%   JO_EED =
CoherenceFilter(I,struct('T',1,'dt',0.1,'rho',4,'Scheme','O','eigenmode',2
));
%   JO_HDCS =
CoherenceFilter(I,struct('T',1,'dt',0.1,'rho',4,'Scheme','O','eigenmode',4
));
%   JS_HDCS =
CoherenceFilter(I,struct('T',1,'dt',0.1,'rho',4,'Scheme','S','eigenmode',4
));
%   JR_HDCS =
CoherenceFilter(I,struct('T',1,'dt',0.1,'rho',4,'Scheme','R','eigenmode',4
));
%
%   figure,
%   subplot(2,3,1), imshow(I), title('Before Filtering');
%   subplot(2,3,2), imshow(JO), title('Optimized Scheme');
%   subplot(2,3,3), imshow(JO_EED), title('Edge Enhancing Optimized
Scheme');
%   subplot(2,3,4), imshow(JO_HDCS), title('HDCS Optimized Scheme');
%   subplot(2,3,5), imshow(JS_HDCS), title('HDCS Standard Scheme');
%   subplot(2,3,6), imshow(JR_HDCS), title('Rotation invariant Scheme');
%
% example 3d,
%   % First compile the c-code by executing compile_c_files.m
%   load('images/sphere');
%   showcs3(V);
%   JR = CoherenceFilter(V,struct('T',50,'dt',2,'Scheme','R'));
%   showcs3(JR);
%
% example 3d, Mendrik
%   load('images/sphere');
%   showcs3(V);
%   JS =
CoherenceFilter(V,struct('T',5,'dt',0.15,'Scheme','S','eigenmode',4));
%   JO =
CoherenceFilter(V,struct('T',5,'dt',0.50,'Scheme','O','eigenmode',4));
%   showcs3(JS);
%   showcs3(JO);
%

% add all needed function paths
try
    functionname='CoherenceFilter.m';
    functiondir=which(functionname);
    functiondir=functiondir(1:end-length(functionname));
    addpath([functiondir '/functions2D'])
    addpath([functiondir '/functions3D'])
    addpath([functiondir '/functions'])
catch me
    disp(me.message);
end

% Default parameters
```

```
defaultoptions=struct('T',2,'dt',[],'sigma', 1, 'rho', 1, 'TensorType', 1,
'eigenmode',0,'C', 1e-10,
'm',1,'alpha',0.001,'lambda_e',0.02,'lambda_c',0.02,'lambda_h',0.5,'RealDe
rivatives',false,'Scheme','R','verbose','iter');

if(~exist('Options','var')),
    Options=defaultoptions;
else
    tags = fieldnames(defaultoptions);
    for i=1:length(tags)
        if(~isfield(Options,tags{i})),
Options.(tags{i})=defaultoptions.(tags{i}); end
    end
    if(length(tags)~=length(fieldnames(Options))),
        warning('CoherenceFilter:unknownoption','unknown options found');
    end
end

if(isempty(Options.dt))
    switch lower(Options.Scheme)
      case 'r', Options.dt=0.15;
      case 'o', Options.dt=0.15;
      case 'i', Options.dt=0.15;
      case 's', Options.dt=0.15;
      case 'n', Options.dt=0.15;
      otherwise
        error('CoherenceFilter:unknownoption','unknown scheme');
    end
end



% Initialization
dt_max = Options.dt; t = 0;

% In case of 3D use single precision to save memory
if(size(u,3)<4), u=double(u); else u=single(u); end

% Process time
process_time=tic;

% Show information
switch lower(Options.verbose(1))
case 'i'
    disp('Diffusion time   Sec. Elapsed');
case 'f'
    disp('Diffusion time   Sec. Elapsed   Image mean    Image variance');
end



% Anisotropic diffusion main loop
while (t < (Options.T-0.001))
    % Update time, adjust last time step to exactly finish at the wanted
    % diffusion time
    Options.dt = min(dt_max,Options.T-t); t = t + Options.dt;
    tn=toc(process_time);
```

```
    switch lower(Options.verbose(1))
    case 'n'
    case 'i'
        s=sprintf('    %5.0f        %5.0f    ',t,round(tn)); disp(s);
    case 'f'
        s=sprintf('    %5.0f        %5.0f      %13.6g    %13.6g
',t,round(tn), mean(u(:)), var(u(:))); disp(s);

    end

    if(size(u,3)<4) % Check if 2D or 3D
        % Do a diffusion step

if(strcmpi(Options.Scheme,'R')&&(Options.eigenmode==0)&&(exist('CoherenceF
ilterStep2D')==3))
            u=CoherenceFilterStep2D(u,Options);
        else
            u=Anisotropic_step2D(u,Options);
        end
    else
        % Do a diffusion step
        if(strcmpi(Options.Scheme,'R'))
            u=CoherenceFilterStep3D(u,Options);
        else
            u=Anisotropic_step3D(u,Options);
        end
    end
end

function u=Anisotropic_step2D(u,Options)
% Perform tensor-driven diffusion filtering update

% Gaussian smooth the image, for better gradients
usigma=imgaussian(u,Options.sigma,4*Options.sigma);


% Calculate the gradients
switch lower(Options.Scheme)
  case {'r','o','i'}
    ux=derivatives(usigma,'x'); uy=derivatives(usigma,'y');
  case {'s','n'}
    [uy,ux]=gradient(usigma);
  otherwise
    error('CoherenceFilter:unknownoption','unknown scheme');
end


% Compute the 2D structure tensors J of the image
[Jxx, Jxy, Jyy] = StructureTensor2D(ux,uy,Options.rho);

% Compute the eigenvectors and values of the strucure tensors, v1 and v2,
mu1 and mu2
[mu1,mu2,v1x,v1y,v2x,v2y]=EigenVectors2D(Jxx,Jxy,Jyy);

% Gradient magnitude squared
```

```
gradA=ux.^2+uy.^2;

% Construct the edge preserving diffusion tensors D = [Dxx,Dxy;Dxy,Dyy]
[Dxx,Dxy,Dyy]=ConstructDiffusionTensor2D(mu1,mu2,v1x,v1y,v2x,v2y,gradA,Opt
ions);

% Do the image diffusion
switch lower(Options.Scheme)
  case 'o'
      u=diffusion_scheme_2D_novel(u,Dxx,Dxy,Dyy,Options.dt);
      %u=diffusion_scheme_2D_high_rotation(u,Dxx,Dxy,Dyy,Options.dt,b);
  case 'r'
      u=diffusion_scheme_2D_rotation_invariant(u,Dxx,Dxy,Dyy,Options.dt);
  case 'i'
      u=diffusion_scheme_2D_implicit(u,Dxx,Dxy,Dyy,Options.dt);
  case 's'
      u=diffusion_scheme_2D_standard(u,Dxx,Dxy,Dyy,Options.dt);
  case 'n'
      u=diffusion_scheme_2D_non_negativity(u,Dxx,Dxy,Dyy,Options.dt);
  otherwise
    error('CoherenceFilter:unknownoption','unknown scheme');
end

function u=Anisotropic_step3D(u,Options)
% Perform tensor-driven diffusion filtering update

% Gaussian smooth the image, for better gradients
usigma=imgaussian(u,Options.sigma,4*Options.sigma);

% Calculate the gradients
ux=derivatives(usigma,'x');
uy=derivatives(usigma,'y');
uz=derivatives(usigma,'z');

% Compute the 3D structure tensors J of the image
[Jxx, Jxy, Jxz, Jyy, Jyz, Jzz] = StructureTensor3D(ux,uy,uz, Options.rho);

% Gradient magnitude squared
gradA=ux.^2+uy.^2+uz.^2;

% Free memory
clear ux; clear uy; clear uz;

% Compute the eigenvectors and eigenvalues of the hessian and directly
% use the equation of Weickert to convert them to diffusion tensors
[Dxx,Dxy,Dxz,Dyy,Dyz,Dzz]=StructureTensor2DiffusionTensor3D(Jxx,Jxy,Jxz,Jy
y,Jyz,Jzz,gradA,Options);

% Free memory
clear J*;

% Do the image diffusion
switch lower(Options.Scheme)
  case 'o'
      u=diffusion_scheme_3D_novel(u,Dxx,Dxy,Dxz,Dyy,Dyz,Dzz,Options.dt);
```

```
%u=diffusion_scheme_3D_high_rotation(u,Dxx,Dxy,Dxz,Dyy,Dyz,Dzz,Options.dt)
;
  case 'r'

u=diffusion_scheme_3D_rotation_invariant(u,Dxx,Dxy,Dxz,Dyy,Dyz,Dzz,Options
.dt);
  case 'i'

u=diffusion_scheme_3D_implicit(u,Dxx,Dxy,Dxz,Dyy,Dyz,Dzz,Options.dt);
  case 's'

u=diffusion_scheme_3D_standard(u,Dxx,Dxy,Dxz,Dyy,Dyz,Dzz,Options.dt);
  case 'n'

u=diffusion_scheme_3D_non_negativity(u,Dxx,Dxy,Dxz,Dyy,Dyz,Dzz,Options.dt)
;
  otherwise
    error('CoherenceFilter:unknownoption','unknown scheme');
end
```

Appendix 3: CHI SQUARE COMBINATION

```
==========================================================================
% Rician noise generation using two orthogal Gaussian  generators
% copyright © 2016 Kenneth Kagoiya
% Permission to use, copy or modify this software and its document.

% This program implements method two developed i.e wavelet hybrid MRI
%denoising scheme using chi square and unbiased risk estimate with
% bilateral filtering
--------------------------------------------------------------------------
 I= imread('C:/torso.bmp');
w=5
sigma_d=2
sigma_r=2
[X,Y] = meshgrid(-w:w,-w:w);
G = 2.71^(-(X.^2+Y.^2)/(2*sigma_d^2));

I = rgb2gray(A);
for phi= 0:0.375:1.5
II=I*cos(phi)
J = imnoise(II,'gaussian',0.0,0.0005);
J = imresize(J,[ 480 480])
figure,imshow(J)
K=I*sin(phi)
K = imnoise(K,'gaussian',0.0,0.0005);
K = imresize(K,[ 480 480])
figure,imshow(K)
L=(J+K)*0.75
figure,imshow(L)
end
val1 = std2(I)
val2 = mean2(I)
val3 = std2(II)
val4 = mean2(II)
val5 = std2(K)
val6 = mean2(K)
val7 = std2(K)
val8 = mean2(K)
% Apply bilateral filter for boundary enhancement.
dim = size(J);
B = zeros(dim);
for i = 1:dim(1)
   for j = 1:dim(2)

         % Extract local region.
         iMin = max(i-w,1);
         iMax = min(i+w,dim(1));
         jMin = max(j-w,1);
         jMax = min(j+w,dim(2));
         I = A(iMin:iMax,jMin:jMax);
      figure, imshow(I)
         % Compute Gaussian intensity weights.
         H = 2.71^(-(I-A(i,j)).^2/(2*sigma_r^2));

         % Calculate bilateral filter response.
         F = H.*G((iMin:iMax)-i+w+1,(jMin:jMax)-j+w+1);
```

```
         B(i,j) = sum(F(:).*I(:))/sum(F(:));
      figure, imshow(B);
      figure, imshow(A);
   end

end

level 4 decomposition
I = rgb2gray(R);
noisy_image=I
[cA,cH,cV,cD] = dwt2(I,'Haar')
%[wc] = fwt_image(noisy_image,6);  % 6-level DWT: lowest band is 16x16
[CA1, CH1, CV1, CD1] = dwt2(cA,'Haar');
[CA2, CH2, CV2, CD2] = dwt2(CA1,'Haar');
[CA3, CH3, CV3, CD3] = dwt2(CA2,'Haar');
[CA4, CH4, CV4, CD4] = dwt2(CA3,'Haar');
[CA5, CH5, CV5, CD5] = dwt2(CA4,'Haar');

% Reconstruction algorithm
%   ca5 .... ch3_w cv3_w cd3_w ch2_w
%   cv2_w cd2_w ch1_w cv1_w cd1_w
% --------------------------------

ch5_w = wiener2(CH5,[5 5], sigman^2);
cv5_w = wiener2(CV5,[5 5], sigman^2);
cd5_w = wiener2(CD5,[5 5], sigman^2);

ch4_w = wiener2(CH4,[5 5], sigman^2);
cv4_w = wiener2(CV4,[5 5], sigman^2);
cd4_w = wiener2(CD4,[5 5], sigman^2);

ch3_w = wiener2(CH3,[5 5], sigman^2);
cv3_w = wiener2(CV3,[5 5], sigman^2);
cd3_w = wiener2(CD3,[5 5], sigman^2);

ch2_w = wiener2(CH2,[5 5], sigman^2);
cv2_w = wiener2(CV2,[5 5], sigman^2);
cd2_w = wiener2(CD2,[5 5], sigman^2);

ch1_w = wiener2(CH1,[5 5], sigman^2);
cv1_w = wiener2(CV1,[5 5], sigman^2);
cd1_w = wiener2(CD1,[5 5], sigman^2);

%  reconstructed image
% ---------------------------

ca5 = CA5;

wc = [ca5 ch5_w
      cv5_w   cd5_w  ];
wc = [wc ch4_w
      cv4_w   cd4_w  ];
wc = [wc ch3_w
      cv3_w   cd3_w  ];
wc = [wc ch2_w
```

```
      cv2_w   cd2_w  ];
wc = [wc ch1_w
      cv1_w   cd1_w  ];
  X = idwt2(cA,cH,cV,cD,'Haar')

%out = iwt_image(wc,6);
psnr = feval('psnr',clean_image,out);
% Apply bilateral filter for feature enhancement.
dim = size(J);
B = zeros(dim);
for i = 1:dim(1)
   for j = 1:dim(2)

         % Extract local region.
         iMin = max(i-w,1);
         iMax = min(i+w,dim(1));
         jMin = max(j-w,1);
         jMax = min(j+w,dim(2));
         I = A(iMin:iMax,jMin:jMax);
      figure, imshow(I)
         % Compute Gaussian intensity weights.
         H = 2.71^(-(I-A(i,j)).^2/(2*sigma_r^2));

         % Calculate bilateral filter response.
         F = H.*G((iMin:iMax)-i+w+1,(jMin:jMax)-j+w+1);
         B(i,j) = sum(F(:).*I(:))/sum(F(:));
     figure, imshow(B);
     figure, imshow(A);
   end

end
Level   4 decomposition
I = rgb2gray(R);
noisy_image=I
[cA,cH,cV,cD] = dwt2(I,'Haar')
%[wc] = fwt_image(noisy_image,6);  % 6-level DWT: lowest band is 16x16
[CA1, CH1, CV1, CD1] = dwt2(cA,'Haar');
[CA2, CH2, CV2, CD2] = dwt2(CA1,'Haar');
[CA3, CH3, CV3, CD3] = dwt2(CA2,'Haar');
[CA4, CH4, CV4, CD4] = dwt2(CA3,'Haar');
[CA5, CH5, CV5, CD5] = dwt2(CA4,'Haar');

% Reconstruction algorithm
%   ca5 .... ch3_w cv3_w cd3_w ch2_w
%   cv2_w cd2_w ch1_w cv1_w cd1_w
% ------------------------------

ch5_w = wiener2(CH5,[5 5], sigman^2);
cv5_w = wiener2(CV5,[5 5], sigman^2);
cd5_w = wiener2(CD5,[5 5], sigman^2);

ch4_w = wiener2(CH4,[5 5], sigman^2);
cv4_w = wiener2(CV4,[5 5], sigman^2);
cd4_w = wiener2(CD4,[5 5], sigman^2);
```

```
ch3_w = wiener2(CH3,[5 5], sigman^2);
cv3_w = wiener2(CV3,[5 5], sigman^2);
cd3_w = wiener2(CD3,[5 5], sigman^2);

ch2_w = wiener2(CH2,[5 5], sigman^2);
cv2_w = wiener2(CV2,[5 5], sigman^2);
cd2_w = wiener2(CD2,[5 5], sigman^2);

ch1_w = wiener2(CH1,[5 5], sigman^2);
cv1_w = wiener2(CV1,[5 5], sigman^2);
cd1_w = wiener2(CD1,[5 5], sigman^2);

%  reconstructed image
% ---------------------------

ca5 = CA5;

wc = [ca5 ch5_w
      cv5_w   cd5_w  ];
wc = [wc ch4_w
      cv4_w   cd4_w  ];
wc = [wc ch3_w
      cv3_w   cd3_w  ];
wc = [wc ch2_w
      cv2_w   cd2_w  ];
wc = [wc ch1_w
      cv1_w   cd1_w  ];
  X = idwt2(cA,cH,cV,cD,'Haar')

%out = iwt_image(wc,6);
psnr = feval('psnr',clean_image,out);

function [fima]=mixingsubband(fimau,fimao)

s = size(fimau);

p(1) = 2^(ceil(log2(s(1))));
p(2) = 2^(ceil(log2(s(2))));
p(3) = 2^(ceil(log2(s(3))));

pad1 = zeros(p(1),p(2),p(3));
pad2 = pad1;
pad1(1:s(1),1:s(2),1:s(3)) = fimau(:,:,:);
pad2(1:s(1),1:s(2),1:s(3)) = fimao(:,:,:);

[af, sf] = farras;
w1 = dwt3D(pad1,1,af);
w2 = dwt3D(pad2,1,af);

w1{1}{3} = w2{1}{3};
w1{1}{5} = w2{1}{5};
w1{1}{6} = w2{1}{6};
w1{1}{7} = w2{1}{7};
```

```
fima = idwt3D(w1,1,sf);
fima = fima(1:s(1),1:s(2),1:s(3));

% NAN checking
ind=find(isnan(fima(:)));
fima(ind)=fimau(ind);

% negative checking (only for rician noise mixing)
ind=find(fima<0);
fima(ind)=0;

 % s = size(fimau);
%
% p(1) = 2^(ceil(log2(s(1))));
% p(2) = 2^(ceil(log2(s(2))));
% p(3) = 2^(ceil(log2(s(3))));
%
% pad1 = zeros(p(1),p(2),p(3));
% pad2=pad1;
% pad1(1:s(1),1:s(2),1:s(3)) = fimau(:,:,:);
% pad2(1:s(1),1:s(2),1:s(3)) = fimao(:,:,:);
%
% [af, sf] = farras;
% w1 = dwt3D(pad1,1,af);
% w2 = dwt3D(pad2,1,af);
%
%   w1{1}{1} = (w1{1}{1} + w2{1}{1})/2;
%   w1{1}{2} = (w1{1}{2} + w2{1}{2})/2;
%   w1{1}{3} = w2{1}{3};
%   w1{1}{4} = (w1{1}{4} + w2{1}{4})/2;
%   w1{1}{5} = w2{1}{5};
%   w1{1}{6} = w2{1}{6};
%   w1{1}{7} = w2{1}{7};
%
% fima = idwt3D(w1,1,sf);
% fima = fima(1:s(1),1:s(2),1:s(3));

% Apply bilateral filter for final enhancement.
dim = size(J);
B = zeros(dim);
for i = 1:dim(1)
   for j = 1:dim(2)

         % Extract local region.
         iMin = max(i-w,1);
         iMax = min(i+w,dim(1));
         jMin = max(j-w,1);
         jMax = min(j+w,dim(2));
         I = A(iMin:iMax,jMin:jMax);
      figure, imshow(I)
         % Compute Gaussian intensity weights.
         H = 2.71^(-(I-A(i,j)).^2/(2*sigma_r^2));

         % Calculate bilateral filter response.
         F = H.*G((iMin:iMax)-i+w+1,(jMin:jMax)-j+w+1);
         B(i,j) = sum(F(:).*I(:))/sum(F(:));
```

```
     figure, imshow(B);
     figure, imshow(A);
   end

end


% Close waitbar.
%close(h);
% Pre-compute Gaussian distance weights.
```

PHASE DENOISE

```
A= imread('C:/torso.bmp');
w=5
sigma_d=2
sigma_r=2
[X,Y] = meshgrid(-w:w,-w:w);
G = 2.71^(-(X.^2+Y.^2)/(2*sigma_d^2));

I = rgb2gray(A);
for phi= 0:0.375:1.5
II=I*cos(phi)
J = imnoise(II,'gaussian',0.0,0.0005);
J = imresize(J,[ 480 480])
figure,imshow(J)
K=I*sin(phi)
K = imnoise(K,'gaussian',0.0,0.0005);
K = imresize(K,[ 480 480])
figure,imshow(K)
L=(J+K)*0.75
figure,imshow(L)
end
val1 = std2(I)
val2 = mean2(I)
val3 = std2(II)
val4 = mean2(II)
val5 = std2(K)
val6 = mean2(K)
val7 = std2(K)
val8 = mean2(K)
% Apply bilateral filter.
dim = size(A);
B = zeros(dim);
for i = 1:dim(1)
   for j = 1:dim(2)

         % Extract local region.
         iMin = max(i-w,1);
         iMax = min(i+w,dim(1));
         jMin = max(j-w,1);
         jMax = min(j+w,dim(2));
         I = A(iMin:iMax,jMin:jMax);
      figure, imshow(I)
         % Compute Gaussian intensity weights.
```

```
        H = 2.71^(-(I-A(i,j)).^2/(2*sigma_r^2));

        % Calculate bilateral filter response.
        F = H.*G((iMin:iMax)-i+w+1,(jMin:jMax)-j+w+1);
        B(i,j) = sum(F(:).*I(:))/sum(F(:));
    figure, imshow(B);
    figure, imshow(A);
  end

end

% Close waitbar.
%close(h);
 function [x, mask0] = mri_phase_denoise(yi, varargin)
%|function [x, mask0] = mri_phase_denoise(yi, [options]) <- recommended
usage
%|function [x] = mri_phase_denoise(yi, l2b, niter, chat, wthresh) <- old
way
%|
%| in
%|  yi  [(N)]   noisy complex image: y = mag .* exp(1i * x)
%|          can be any size: 1D, 2D, 3D, ...
%|
%| options
%|  l2b     log_2(beta), regularization parameter
%|  order       regularization order (default: 1, for historical
%|              reasons, but 2 is probably preferable)
%|  niter       # of iterations
%|  chat        1 to show pictures
%|  wthresh     fraction of magnitude maximum to include in fitting
%|  init        initial image for iterations
%|  isave       which iterations to save.  (default: last)
%|  'pl'    1|0 1 for new PL method (recommend), 0 for old way (default)
%| out
%|  x   [(N)]   cleaned up phase estimate
%|  mask0   [(N)]   logical: 1 for high-magnitude pixels
%|
%| Example of weighted "denoising" of MRI phase images.
%| This is a "simple" way to estimate good field inhomogeneity maps
%| from the usual approach of two readouts with a short delay.
%| It also smoothly interpolates over regions with signal voids.
%|
%| Copyright 1999, Jeff Fessler, University of Michigan

if nargin < 1, help(mfilename), error(mfilename), end
if ischar(yi)
    [x mask0] = mri_phase_denoise_test(yi, varargin{:});
    if ~nargout, clear x mask0, end
return
end

% defaults
arg.chat = 0;
arg.l2b = -5;
arg.order = 1;
arg.niter = 150;
```

```
arg.isave = [];
arg.wthresh = 0.4;
arg.fmax = 0.05; % fraction of max threshold for trimmed median for wi_ml
arg.init = [];
arg.clim = []; % limits for phase display
arg.pl = false; % PL
arg.wi_ml = false; % use wi based on ML instead of threshold

% backword compatible for old argument list: l2b, niter, chat, wthresh
if length(varargin) && isnumeric(varargin{1})
    arg.l2b = varargin{1};
    if length(varargin) >= 2, arg.niter = varargin{2}; end
    if length(varargin) >= 3, arg.chat = varargin{3}; end
    if length(varargin) >= 4, arg.wthresh = varargin{4}; end
else
    arg = vararg_pair(arg, varargin);
end
if isempty(arg.isave), arg.isave = arg.niter; end

mag = abs(yi);
yi = angle(yi);
if arg.chat
    im clf, im pl 2 3
    im(1, mag, 'magnitude'), cbar
    im(2, yi, 'raw phase map', arg.clim), cbar
end

dim_yi = size(yi);

%
% specify weights: this needs more work to be automatic!
%
mask0 = mag > arg.wthresh * max(mag(:)); % ignore pixels with "too small"
magnitude
if arg.chat
    im(3, mask0, 'weights'), cbar
    im(4, mask0 .* yi, 'masked phase', arg.clim), cbar
end

%mean(mag(:))
%median(mag(:))
%clf, hist(mag(:), 100), pause
%median(mag(mag(:) > 0.05 * max(mag(:))))

if arg.wi_ml
    wi = mri_phase_wi_ml(mag(:), arg.fmax);
else
    wi = mask0(:);
end
W = diag_sp(wi);

%
% initial phase image
%
if isempty(arg.init)
```

```
    arg.init = yi;
    arg.init(mask0 == 0) = mean(yi(mask0 == 0));
end
if arg.chat
    im(5, arg.init, 'Initial phase', arg.clim), cbar
end

G = diag_sp(ones(prod(dim_yi),1));
%
% regularizer
%
if arg.order ~= 2, warn('order=2 recommended'), end
mask1 = true(size(yi)); % estimate / extrapolate to *all* pixels
R = Reg1(mask1, 'beta', 2^arg.l2b, 'order', arg.order);
%R = Robject(mask1, 'beta', 2^arg.l2b, 'order', arg.order);
%   'type_denom', 'matlab', ...

if 0 % old way
    [C wjk] = C2sparse('tight', mask1, 8); % todo: cut?
    C = spdiag(sqrt(wjk), 'nowarn') * C; % caution: missing prior to 2005-
11-28
    C = sqrt(2^arg.l2b) * C;
end

% report expected blur (at image center)
if 1
    qpwls_psf(G, R, 1, mask1, W);
end

%
% run qpwls algorithm for regularized fitting
%
xinit = arg.init(mask1);
if arg.pl
    med = median(mag(mag(:) > 0.05 * max(mag(:))));
    data = {yi(:), (mag(:)/med).^2}; handle = @phase_dercurv; % PL
%profile on % todo!
    x = pl_pcg_qs_ls(xinit, G, data, handle, R, ...
        'niter', arg.niter, 'isave', arg.isave);
%profile report
    x = embed(x, mask1);
else
    warn 'recommend using "pl" option.  use wls for historical only'
%   data = {yi(:), wi(:)}; handle = @wls_dercurv; % qpwls
    x = qpwls_pcg1(xinit, G, W, yi(:), R.C, ...
        'niter', arg.niter, 'isave', arg.isave);
    x = embed(x, mask1);
end

if 0 % old way
    x = qpwls_pcg(x, G, W, yi(:), 0, R.C, 1, arg.niter);
    x = reshape(x, [dim_yi arg.niter]);
    x = x(:,:,end);
end

if arg.chat
```

```
    if ndims(yi) == 2
        im(6, x(:,:,end), 'QPWLS-CG phase', arg.clim), cbar
    else % 3d
        im(6, x(:,:,:,end), 'QPWLS-CG phase', arg.clim), cbar
    end
end


%
% mri_phase_wi_ml()
% wi based on ML estimation
% trick: normalize by median of non-background so that beta is "universal"
%
function wi = mri_phase_wi_ml(mag, fmax)
med = median(mag(mag(:) > fmax * max(mag(:))));
wi = (mag(:) / med).^2;


%
% phase_dercurv()
% wi * (1 - cos(yi - li))
%
function [deriv, curv] = phase_dercurv(data, li, varargin)
yi = data{1};
wi = data{2};
deriv = wi .* sin(li - yi);
curv = wi;


%
% built-in test/example
%
function [xq, mask0] = mri_phase_denoise_test(type, varargin)

% read data
f.dir = path_find_dir('mri');
f.dir = [f.dir '/phase-data/'];
f.dat = [f.dir 'phfit.mat'];
if ~exist(f.dat, 'file')
    fail('edit the path in %s!', mfilename)
end
yi = mat_read(f.dat);
clim = [-0.5 1.5];

%if nargout
%   order = 1;
%else
%   order = 2;
%end
[xq mask0] = mri_phase_denoise(yi, ...
    'clim', clim, 'init', [], 'chat', 1, varargin{:});
%   'init', 5*randn(size(yi)));
%   'init', 5*ones(size(yi)));
xq = xq(:,:,end);
if im
```

```
    title 'QPWLS-CG phase (simple wi)'
end

cpu etic
xpl = mri_phase_denoise(yi, 'pl', 1, varargin{:});
cpu etoc 'PL time'
im(4, xpl, 'PL-CG phase', clim), cbar

cpu etic
xml_qpwls = mri_phase_denoise(yi, 'wi_ml', 1, varargin{:});
cpu etoc 'PWLS time'
im(5, xml_qpwls, 'QPWLS-CG (ML wj)', clim), cbar

max_percent_diff(xpl, xml_qpwls)
%max_percent_diff(xpl, xq)
nrms(xml_qpwls, xpl)
nrms(xq, xpl)
%im(4, xpl-xq), cbar

%savefig fig_mr_phase_pl
```

Appendix 4: COMBINATION NON-LOCAL MEANS FILTER

```
=========================================================================
% Rician noise generation using two orthogal Gaussian  generators
% copyright © 2016 Kenneth Kagoiya
% Permission to use, copy or modify this software and its document.

% This program implements method one a hybrid and adaptive MRI denoising
% method involving a bilateral filer enhancement and Non-local means
% wavelet based method
-------------------------------------------------------------------------

I = imread('C:/torso.bmp');
for phi= 0:0.375:1.5
II=I*cos(phi)
J = imnoise(II,'gaussian',0.0,0.0005);
J = imresize(J,[ 480 480])
figure,imshow(J)
K=I*sin(phi)
K = imnoise(K,'gaussian',0.0,0.0005);
K = imresize(K,[ 480 480])
figure,imshow(K)
L=(J+K)*0.707
figure,imshow(L)
end
val1 = std2(I)
val2 = mean2(I)
val3 = std2(II)
val4 = mean2(II)
val5 = std2(K)
val6 = mean2(K)
val7 = std2(K)
val8 = mean2(K)

% BFILTER2 Two dimensional bilateral filtering.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Pre-process input and select appropriate filter.
function B = bfilter2(A,w,sigma)

% Verify that the input image exists and is valid.
if ~exist('A','var') || isempty(A)
   error('Input image A is undefined or invalid.');
end
if ~isfloat(A) || ~sum([1,3] == size(A,3)) || ...
      min(A(:)) < 0 || max(A(:)) > 1
   error(['Input image A must be a double precision ',...
          'matrix of size NxMx1 or NxMx3 on the closed ',...
          'interval [0,1].']);
end

% Verify bilateral filter window size.
if ~exist('w','var') || isempty(w) || ...
      numel(w) ~= 1 || w < 1
   w = 5;
end
```

```
w = ceil(w);

% Verify bilateral filter standard deviations.
if ~exist('sigma','var') || isempty(sigma) || ...
      numel(sigma) ~= 2 || sigma(1) <= 0 || sigma(2) <= 0
   sigma = [3 0.1];
end

% Apply either grayscale or color bilateral filtering.
if size(A,3) == 1
   B = bfltGray(A,w,sigma(1),sigma(2));
else
   B = bfltColor(A,w,sigma(1),sigma(2));
end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Implements bilateral filtering for grayscale images.
function B = bfltGray(A,w,sigma_d,sigma_r)

% Pre-compute Gaussian distance weights.
[X,Y] = meshgrid(-w:w,-w:w);
G = exp(-(X.^2+Y.^2)/(2*sigma_d^2));

% Create waitbar.
h = waitbar(0,'Applying bilateral filter...');
set(h,'Name','Bilateral Filter Progress');

% Apply bilateral filter.
dim = size(A);
B = zeros(dim);
for i = 1:dim(1)
   for j = 1:dim(2)

         % Extract local region.
         iMin = max(i-w,1);
         iMax = min(i+w,dim(1));
         jMin = max(j-w,1);
         jMax = min(j+w,dim(2));
         I = A(iMin:iMax,jMin:jMax);

         % Compute Gaussian intensity weights.
         H = exp(-(I-A(i,j)).^2/(2*sigma_r^2));

         % Calculate bilateral filter response.
         F = H.*G((iMin:iMax)-i+w+1,(jMin:jMax)-j+w+1);
         B(i,j) = sum(F(:).*I(:))/sum(F(:));

   end
   waitbar(i/dim(1));
end

% Close waitbar.
close(h);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Implements bilateral filter for color images.
function B = bfltColor(A,w,sigma_d,sigma_r)

% Convert input sRGB image to CIELab color space.
if exist('applycform','file')
   A = applycform(A,makecform('srgb2lab'));
else
   A = colorspace('Lab<-RGB',A);
end

% Pre-compute Gaussian domain weights.
[X,Y] = meshgrid(-w:w,-w:w);
G = exp(-(X.^2+Y.^2)/(2*sigma_d^2));

% Rescale range variance (using maximum luminance).
sigma_r = 100*sigma_r;

% Create waitbar.
h = waitbar(0,'Applying bilateral filter...');
set(h,'Name','Bilateral Filter Progress');

% Apply bilateral filter.
dim = size(A);
B = zeros(dim);
for i = 1:dim(1)
   for j = 1:dim(2)

         % Extract local region.
         iMin = max(i-w,1);
         iMax = min(i+w,dim(1));
         jMin = max(j-w,1);
         jMax = min(j+w,dim(2));
         I = A(iMin:iMax,jMin:jMax,:);

         % Compute Gaussian range weights.
         dL = I(:,:,1)-A(i,j,1);
         da = I(:,:,2)-A(i,j,2);
         db = I(:,:,3)-A(i,j,3);
         H = exp(-(dL.^2+da.^2+db.^2)/(2*sigma_r^2));

         % Calculate bilateral filter response.
         F = H.*G((iMin:iMax)-i+w+1,(jMin:jMax)-j+w+1);
         norm_F = sum(F(:));
         B(i,j,1) = sum(sum(F.*I(:,:,1)))/norm_F;
         B(i,j,2) = sum(sum(F.*I(:,:,2)))/norm_F;
         B(i,j,3) = sum(sum(F.*I(:,:,3)))/norm_F;

   end
   waitbar(i/dim(1));
end
```

```
% Convert filtered image back to sRGB color space.
if exist('applycform','file')
   B = applycform(B,makecform('lab2srgb'));
else
   B = colorspace('RGB<-Lab',B);
end

% Close waitbar.
close(h);
% BFILTER2 Two dimensional bilateral filtering for feature enhancement.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Pre-process input and select appropriate filter.
function B = bfilter2(A,w,sigma)

% Verify that the input image exists and is valid.
if ~exist('A','var') || isempty(A)
   error('Input image A is undefined or invalid.');
end
if ~isfloat(A) || ~sum([1,3] == size(A,3)) || ...
      min(A(:)) < 0 || max(A(:)) > 1
   error(['Input image A must be a double precision ',...
          'matrix of size NxMx1 or NxMx3 on the closed ',...
          'interval [0,1].']);
end

% Verify bilateral filter window size.
if ~exist('w','var') || isempty(w) || ...
      numel(w) ~= 1 || w < 1
   w = 5;
end
w = ceil(w);

% Verify bilateral filter standard deviations.
if ~exist('sigma','var') || isempty(sigma) || ...
      numel(sigma) ~= 2 || sigma(1) <= 0 || sigma(2) <= 0
   sigma = [3 0.1];
end

% Apply either grayscale or color bilateral filtering.
if size(A,3) == 1
   B = bfltGray(A,w,sigma(1),sigma(2));
else
   B = bfltColor(A,w,sigma(1),sigma(2));
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Implements bilateral filtering for grayscale images.
function B = bfltGray(A,w,sigma_d,sigma_r)

% Pre-compute Gaussian distance weights.
[X,Y] = meshgrid(-w:w,-w:w);
G = exp(-(X.^2+Y.^2)/(2*sigma_d^2));
```

```
% Create waitbar.
h = waitbar(0,'Applying bilateral filter...');
set(h,'Name','Bilateral Filter Progress');

% Apply bilateral filter.
dim = size(A);
B = zeros(dim);
for i = 1:dim(1)
   for j = 1:dim(2)


         % Extract local region.
         iMin = max(i-w,1);
         iMax = min(i+w,dim(1));
         jMin = max(j-w,1);
         jMax = min(j+w,dim(2));
         I = A(iMin:iMax,jMin:jMax);

         % Compute Gaussian intensity weights.
         H = exp(-(I-A(i,j)).^2/(2*sigma_r^2));

         % Calculate bilateral filter response.
         F = H.*G((iMin:iMax)-i+w+1,(jMin:jMax)-j+w+1);
         B(i,j) = sum(F(:).*I(:))/sum(F(:));

   end
   waitbar(i/dim(1));
end

% Close waitbar.
close(h);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Implements bilateral filter for color images.
function B = bfltColor(A,w,sigma_d,sigma_r)

% Convert input sRGB image to CIELab color space.
if exist('applycform','file')
   A = applycform(A,makecform('srgb2lab'));
else
   A = colorspace('Lab<-RGB',A);
end

% Pre-compute Gaussian domain weights.
[X,Y] = meshgrid(-w:w,-w:w);
G = exp(-(X.^2+Y.^2)/(2*sigma_d^2));

% Rescale range variance (using maximum luminance).
sigma_r = 100*sigma_r;

% Create waitbar.
h = waitbar(0,'Applying bilateral filter...');
set(h,'Name','Bilateral Filter Progress');
```

```
% Apply bilateral filter.
dim = size(A);
B = zeros(dim);
for i = 1:dim(1)
   for j = 1:dim(2)

         % Extract local region.
         iMin = max(i-w,1);
         iMax = min(i+w,dim(1));
         jMin = max(j-w,1);
         jMax = min(j+w,dim(2));
         I = A(iMin:iMax,jMin:jMax,:);

         % Compute Gaussian range weights.
         dL = I(:,:,1)-A(i,j,1);
         da = I(:,:,2)-A(i,j,2);
         db = I(:,:,3)-A(i,j,3);
         H = exp(-(dL.^2+da.^2+db.^2)/(2*sigma_r^2));

         % Calculate bilateral filter response.
         F = H.*G((iMin:iMax)-i+w+1,(jMin:jMax)-j+w+1);
         norm_F = sum(F(:));
         B(i,j,1) = sum(sum(F.*I(:,:,1)))/norm_F;
         B(i,j,2) = sum(sum(F.*I(:,:,2)))/norm_F;
         B(i,j,3) = sum(sum(F.*I(:,:,3)))/norm_F;

   end
   waitbar(i/dim(1));
end

% Convert filtered image back to sRGB color space.
if exist('applycform','file')
   B = applycform(B,makecform('lab2srgb'));
else
   B = colorspace('RGB<-Lab',B);
end

% Close waitbar.
close(h);
 J=NLMF(I,Options)

function J=NLMF(I,Options)
% This function NLMF performs Non-Local Means noise filtering of
% 2D grey/color or 3D image data. The function is partly c-coded for
% cpu efficient filtering.
%
% Principle NL-Mean filter:
% A local pixel region (patch) around a pixel is compared to patches
% of pixels in the neighbourhood. The centerpixels of the patches are
% averaged depending on the quadratic pixel distance between the patches.
%
% Function:
%
%   J = NLMF( I, Options);
```

```
%
% inputs,
%   I : 2D grey/color or 3D image data, of type Single or Double
%           in range [0..1]
%   Options : Struct with options
%
% outputs,
%   J : The NL-means filtered image or image volume
%
% options,
%   Options.kernelratio : Radius of local Patch (default 3)
%   Options.windowratio : Radius of neighbourhood search window (default
3)
%   Options.filterstrength : Strength of the NLMF filtering (default 0.05)
%   Options.blocksize : The image is split in sub-blocks for efficienter
%                       memory usage,  (default 2D: 150, default 3D: 32);
%   Options.nThreads : Number of CPU Threads used default (2);
%   Options.verbose : When set to true display information (default false)
%
% Beta Options:
%   Options.enablepca : Do PCA on the patches to reduce amount of
%                          calculations (default false)
%   Options.pcaskip : To reduce amount of PCA calculations the data for
PCA
%                       is first reduced with V(:,1:pcaskip:end)  (default
10)
%   Options.pcane : Number of eigenvectors used (default 25)
%
% Literature:
% - Non-local filter proposed for A. Buades, B. Coll  and J.M. Morel
%   "A non-local algorithm for image denoising"
% - Basic Matlab implementation of Jose Vicente Manjon-Herrera
%
%
% First Compile c-code!!!!, with :
%   mex vectors_nlmeans_single.c -v
%   mex image2vectors_single.c -v
%   mex vectors_nlmeans_double.c -v
%   mex image2vectors_double.c -v
%
% Example 2D greyscale,
%  I=im2double(imread('moon.tif'));
%  Options.kernelratio=4;
%  Options.windowratio=4;
%  Options.verbose=true;
%  J=NLMF(I,Options);
%  figure,
%  subplot(1,2,1),imshow(I); title('Noisy image')
%  subplot(1,2,2),imshow(J); title('NL-means image');
%
% Example 2D color,
%  I=im2double(imread('lena.jpg'));
%  I=imnoise(I,'gaussian',0.01);
%  Options.kernelratio=4;
%  Options.windowratio=4;
%  Options.verbose=true;
%  Options.filterstrength=0.1;
```

```
%  J=NLMF(I,Options);
%  figure,
%  subplot(1,2,1),imshow(I); title('Noisy image')
%  subplot(1,2,2),imshow(J); title('NL-means image');
%
% Example 3D,
%  load('mri');
%  D=squeeze(D); D=single(D); D=D./max(D(:));
%  Options.verbose=true;
%  Options.blocksize=45;
%  V=NLMF(D,Options);
%  figure,
%  subplot(1,2,1),imshow(imresize(D(:,:,3),5),[]); title('Noisy slice')
%  subplot(1,2,2),imshow(imresize(V(:,:,3),5),[]); title('NL-means slice')
%
% See also NLMF2Dtree.
%
% Function is written by D.Kroon University of Twente (April 2010)

if((min(I(:))<0)||(max(I(:))>1)),
    warning('NLMF:inputs','Preferable data range [0..1]');
end
if((~isa(I,'double'))&&(~isa(I,'single')))
    error('NLMF:inputs','Input data must be single or double');
end



is2D=size(I,3)<4;

% Process inputs
defaultoptions=struct('kernelratio',3,'windowratio',3,'filterstrength',0.0
5,'blocksize',150,'nThreads',2,'verbose',false,'enablepca',false,'pcaskip'
,10,'pcane',25);
if(is2D), defaultoptions.blocksize=150; else defaultoptions.blocksize=32;
end
if(~exist('Options','var')), Options=defaultoptions;
else
    tags = fieldnames(defaultoptions);
    for i=1:length(tags), if(~isfield(Options,tags{i})),
Options.(tags{i})=defaultoptions.(tags{i}); end, end
    if(length(tags)~=length(fieldnames(Options))),
        warning('NLMF:unknownoption','unknown options found');
    end
end

kernelratio=round(Options.kernelratio);
windowratio=round(Options.windowratio);
filterstrength=Options.filterstrength;
blocksize=round(Options.blocksize);
nThreads=round(Options.nThreads);
verbose=Options.verbose;
enablepca=Options.enablepca;
pcaskip=round(Options.pcaskip);
pcane=round(Options.pcane);

if(is2D)
```

```
    Ipad = padarray(I,[kernelratio+windowratio
kernelratio+windowratio],'symmetric'); %,
else
    Ipad = padarray(I,[kernelratio+windowratio kernelratio+windowratio
kernelratio+windowratio],'symmetric'); %,
end

% Separate the image into smaller blocks, for less memory usage
% and efficient cpu-cache usage.
block=makeBlocks(kernelratio,windowratio, blocksize, I, Ipad, is2D);

tic; erms='***';
J=zeros(size(I),class(Ipad));
for i=1:length(block);
    if(verbose)
        disp(['Processing Block ' num2str(i) ' of ' num2str(length(block))
' estimated time remaining ' erms]);
    end
    if(is2D)
        Iblock=Ipad(block(i).x1:block(i).x2,block(i).y1:block(i).y2,:);
    else

Iblock=Ipad(block(i).x1:block(i).x2,block(i).y1:block(i).y2,block(i).z1:bl
ock(i).z2);
    end

        if(isa(Ipad,'double'))
            % Get the local patches of every pixel-coordinate in the block

V=image2vectors_double(double(Iblock),double(kernelratio),double(nThreads)
);
        else

V=image2vectors_single(single(Iblock),single(kernelratio),single(nThreads)
);
        end

        if(enablepca)
            % Do PCA on the block
            [Evalues, Evectors, x_mean]=PCA(V(:,1:pcaskip:end),pcane);
            % Project the block to the reduced PCA feature space
            V = Evectors'*(V-repmat(x_mean,1,size(V,2)));
        end

        % Do NL-means on the vectors in the block
        if(isa(Ipad,'double'))

Iblock_filtered=vectors_nlmeans_double(double(Iblock),double(V),double(ker
nelratio),double(windowratio),double(filterstrength),double(nThreads));
        else

Iblock_filtered=vectors_nlmeans_single(single(Iblock),single(V),single(ker
nelratio),single(windowratio),single(filterstrength),single(nThreads));
        end

    if(is2D)
```

```
J(block(i).x3:block(i).x4,block(i).y3:block(i).y4,:)=Iblock_filtered;
    else

J(block(i).x3:block(i).x4,block(i).y3:block(i).y4,block(i).z3:block(i).z4)
=Iblock_filtered;
    end

    if(verbose)
        t=toc; erm=(t/i)*(length(block)-i); erms=num2str(erm);
    end
end
toc;


function block=makeBlocks(kernelratio,windowratio,blocksize, I,Ipad, is2D)
block=struct;
i=0;
blocksize_real=blocksize-(kernelratio+windowratio)*2;
if(is2D)
    for y1=1:blocksize_real:size(Ipad,2)
        for x1=1:blocksize_real:size(Ipad,1)
            x2=x1+blocksize-1; y2=y1+blocksize-1;
            x2=max(min(x2,size(Ipad,1)),1);
            y2=max(min(y2,size(Ipad,2)),1);
            x3=x1; y3=y1;
            x4=min(x1+blocksize_real-1,size(I,1));
            y4=min(y1+blocksize_real-1,size(I,2));
            if((x4>=x3)&&(y4>=y3))
                i=i+1;
                block(i).x1=x1; block(i).y1=y1; block(i).x2=x2;
block(i).y2=y2;
                block(i).x3=x3; block(i).y3=y3; block(i).x4=x4;
block(i).y4=y4;
            end
        end
    end
else
    for z1=1:blocksize_real:size(Ipad,3)
        for y1=1:blocksize_real:size(Ipad,2)
            for x1=1:blocksize_real:size(Ipad,1)
                x2=x1+blocksize-1; y2=y1+blocksize-1;  z2=z1+blocksize-1;
                x2=max(min(x2,size(Ipad,1)),1);
                y2=max(min(y2,size(Ipad,2)),1);
                z2=max(min(z2,size(Ipad,3)),1);
                x3=x1; y3=y1; z3=z1;
                x4=min(x1+blocksize_real-1,size(I,1));
                y4=min(y1+blocksize_real-1,size(I,2));
                z4=min(z1+blocksize_real-1,size(I,3));
                if((x4>=x3)&&(y4>=y3)&&(z4>=z3))
                    i=i+1;
                    block(i).x1=x1; block(i).y1=y1;  block(i).z1=z1;
                    block(i).x2=x2; block(i).y2=y2;  block(i).z2=z2;
                    block(i).x3=x3; block(i).y3=y3;  block(i).z3=z3;
                    block(i).x4=x4; block(i).y4=y4;  block(i).z4=z4;
                end
```

```
            end
        end
    end
end

function [Evalues, Evectors, x_mean]=PCA(x,ne)
% PCA using Single Value Decomposition
% Obtaining mean vector, eigenvectors and eigenvalues
%
% [Evalues, Evectors, x_mean]=PCA(x,ne);
%
% inputs,
%   X : M x N matrix with M the trainingvector length and N the number
%               of training data sets
%   ne : Max number of eigenvalues
% outputs,
%   Evalues : The eigen values of the data
%   Evector : The eigen vectors of the data
%   x_mean : The mean training vector
%
s=size(x,2);

% Calculate the mean
x_mean=sum(x,2)/s;

% Substract the mean
x2=(x-repmat(x_mean,1,s))/ sqrt(s-1);

% Do the SVD
[U2,S2] = svds(x2,ne,'L',struct('tol',1e-4));
Evalues=diag(S2).^2;
Evectors=U2;

function [fima]=mixingsubband(fimau,fimao)
s = size(fimau);

p(1) = 2^(ceil(log2(s(1))));
p(2) = 2^(ceil(log2(s(2))));
p(3) = 2^(ceil(log2(s(3))));

pad1 = zeros(p(1),p(2),p(3));
pad2 = pad1;
pad1(1:s(1),1:s(2),1:s(3)) = fimau(:,:,:);
pad2(1:s(1),1:s(2),1:s(3)) = fimao(:,:,:);

[af, sf] = farras;
w1 = dwt3D(pad1,1,af);
w2 = dwt3D(pad2,1,af);

w1{1}{3} = w2{1}{3};
w1{1}{5} = w2{1}{5};
w1{1}{6} = w2{1}{6};
w1{1}{7} = w2{1}{7};

fima = idwt3D(w1,1,sf);
```

```
fima = fima(1:s(1),1:s(2),1:s(3));

% NAN checking
ind=find(isnan(fima(:)));
fima(ind)=fimau(ind);

% negative checking (only for rician noise mixing)
ind=find(fima<0);
fima(ind)=0;


% s = size(fimau);
%
% p(1) = 2^(ceil(log2(s(1))));
% p(2) = 2^(ceil(log2(s(2))));
% p(3) = 2^(ceil(log2(s(3))));
%
% pad1 = zeros(p(1),p(2),p(3));
% pad2=pad1;
% pad1(1:s(1),1:s(2),1:s(3)) = fimau(:,:,:);
% pad2(1:s(1),1:s(2),1:s(3)) = fimao(:,:,:);
%
% [af, sf] = farras;
% w1 = dwt3D(pad1,1,af);
% w2 = dwt3D(pad2,1,af);
%
%   w1{1}{1} = (w1{1}{1} + w2{1}{1})/2;
%   w1{1}{2} = (w1{1}{2} + w2{1}{2})/2;
%   w1{1}{3} = w2{1}{3};
%   w1{1}{4} = (w1{1}{4} + w2{1}{4})/2;
%   w1{1}{5} = w2{1}{5};
%   w1{1}{6} = w2{1}{6};
%   w1{1}{7} = w2{1}{7};
%
% fima = idwt3D(w1,1,sf);
% fima = fima(1:s(1),1:s(2),1:s(3));

% BFILTER2 Two dimensional bilateral filtering   final stage.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Pre-process input and select appropriate filter.
function B = bfilter2(A,w,sigma)

% Verify that the input image exists and is valid.
if ~exist('A','var') || isempty(A)
   error('Input image A is undefined or invalid.');
end
if ~isfloat(A) || ~sum([1,3] == size(A,3)) || ...
      min(A(:)) < 0 || max(A(:)) > 1
   error(['Input image A must be a double precision ',...
          'matrix of size NxMx1 or NxMx3 on the closed ',...
          'interval [0,1].']);
end

% Verify bilateral filter window size.
```

```
if ~exist('w','var') || isempty(w) || ...
      numel(w) ~= 1 || w < 1
   w = 5;
end
w = ceil(w);

% Verify bilateral filter standard deviations.
if ~exist('sigma','var') || isempty(sigma) || ...
      numel(sigma) ~= 2 || sigma(1) <= 0 || sigma(2) <= 0
   sigma = [3 0.1];
end

% Apply either grayscale or color bilateral filtering.
if size(A,3) == 1
   B = bfltGray(A,w,sigma(1),sigma(2));
else
   B = bfltColor(A,w,sigma(1),sigma(2));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Implements bilateral filtering for grayscale images.
function B = bfltGray(A,w,sigma_d,sigma_r)

% Pre-compute Gaussian distance weights.
[X,Y] = meshgrid(-w:w,-w:w);
G = exp(-(X.^2+Y.^2)/(2*sigma_d^2));

% Create waitbar.
h = waitbar(0,'Applying bilateral filter...');
set(h,'Name','Bilateral Filter Progress');

% Apply bilateral filter.
dim = size(A);
B = zeros(dim);
for i = 1:dim(1)
   for j = 1:dim(2)

         % Extract local region.
         iMin = max(i-w,1);
         iMax = min(i+w,dim(1));
         jMin = max(j-w,1);
         jMax = min(j+w,dim(2));
         I = A(iMin:iMax,jMin:jMax);

         % Compute Gaussian intensity weights.
         H = exp(-(I-A(i,j)).^2/(2*sigma_r^2));

         % Calculate bilateral filter response.
         F = H.*G((iMin:iMax)-i+w+1,(jMin:jMax)-j+w+1);
         B(i,j) = sum(F(:).*I(:))/sum(F(:));

   end
   waitbar(i/dim(1));
end
```

```
% Close waitbar.
close(h);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Implements bilateral filter for color images.
function B = bfltColor(A,w,sigma_d,sigma_r)

% Convert input sRGB image to CIELab color space.
if exist('applycform','file')
   A = applycform(A,makecform('srgb2lab'));
else
   A = colorspace('Lab<-RGB',A);
end

% Pre-compute Gaussian domain weights.
[X,Y] = meshgrid(-w:w,-w:w);
G = exp(-(X.^2+Y.^2)/(2*sigma_d^2));

% Rescale range variance (using maximum luminance).
sigma_r = 100*sigma_r;

% Create waitbar.
h = waitbar(0,'Applying bilateral filter...');
set(h,'Name','Bilateral Filter Progress');

% Apply bilateral filter.
dim = size(A);
B = zeros(dim);
for i = 1:dim(1)
   for j = 1:dim(2)

         % Extract local region.
         iMin = max(i-w,1);
         iMax = min(i+w,dim(1));
         jMin = max(j-w,1);
         jMax = min(j+w,dim(2));
         I = A(iMin:iMax,jMin:jMax,:);

         % Compute Gaussian range weights.
         dL = I(:,:,1)-A(i,j,1);
         da = I(:,:,2)-A(i,j,2);
         db = I(:,:,3)-A(i,j,3);
         H = exp(-(dL.^2+da.^2+db.^2)/(2*sigma_r^2));

         % Calculate bilateral filter response.
         F = H.*G((iMin:iMax)-i+w+1,(jMin:jMax)-j+w+1);
         norm_F = sum(F(:));
         B(i,j,1) = sum(sum(F.*I(:,:,1)))/norm_F;
         B(i,j,2) = sum(sum(F.*I(:,:,2)))/norm_F;
         B(i,j,3) = sum(sum(F.*I(:,:,3)))/norm_F;

   end
   waitbar(i/dim(1));
end
```

```
% Convert filtered image back to sRGB color space.
if exist('applycform','file')
   B = applycform(B,makecform('lab2srgb'));
else
   B = colorspace('RGB<-Lab',B);
end

% Close waitbar.
close(h);
```

```
/*              Details on ONLM filter                */

#include "math.h"
#include "mex.h"
#include <stdlib.h>
#include "matrix.h"


/* Function which compute the weighted average for one block */
void Average_block(double *ima,int x,int y,int z,int neighbourhood size,double *average, double
weight, int* vol_size)
{
int x_pos,y_pos,z_pos;
bool is_outside;

int a,b,c;

int count = 0;

        for (c = 0; c<(2*neighbourhood size+1);c++)
        {
                for (b = 0; b<(2*neighbourhood size+1);b++)
                {
                        for (a = 0; a<(2*neighbourhood size+1);a++)
                        {



                                is_outside = false;

                                x_pos = x+a-neighbourhood size;

                                y_pos = y+b-neighbourhood size;

                                z_pos = z+c-neighbourhood size;

                                if ((z_pos < 0) || (z_pos > vol_size[2]-1)) is_outside = true;

                                if ((y_pos < 0) || (y_pos > vol_size[0]-1)) is_outside = true;

                                if ((x_pos < 0) || (x_pos > vol_size[1]-1)) is_outside = true;
```

```
                                        if (is_outside)

                                                average[count] = average[count] +
ima[z*(vol_size[0]*vol_size[1])+(x*vol_size[0])+y]*weight;

                                        else

                                                average[count] = average[count] +
ima[z_pos*(vol_size[0]*vol_size[1])+(x_pos*vol_size[0])+y_pos]*weight;

                                                count++;

                                }
                        }
                }
}


/* Function which computes the value assigned to each voxel */

void Value_block(double *Estimate, double *Label,int x,int y,int z,int neighbourhood size,double
*average, double global_sum, int* vol_size)

{

int x_pos,y_pos,z_pos;

int ret;

bool is_outside;

double value = 0.0;

double label = 0.0;

int count=0 ;

int a,b,c;


        for (c = 0; c<(2*neighbourhood size+1);c++)
        {
                for (b = 0; b<(2*neighbourhood size+1);b++)
                {
                        for (a = 0; a<(2*neighbourhood size+1);a++)
```

```
                {
                        is_outside = false;
                        x_pos = x+a-neighbourhood size;
                        y_pos = y+b-neighbourhood size;
                        z_pos = z+c-neighbourhood size;

                        if ((z_pos < 0) || (z_pos > vol_size[2]-1)) is_outside = true;
                        if ((y_pos < 0) || (y_pos > vol_size[0]-1)) is_outside = true;
                        if ((x_pos < 0) || (x_pos > vol_size[1]-1)) is_outside = true;
                        if (!is_outside)
                        {
                        value =
Estimate[z_pos*(vol_size[0]*vol_size[1])+(x_pos*vol_size[0])+y_pos];
                                value = value + (average[count]/global_sum);

                                label = Label[(y_pos + x_pos*vol_size[0] + z_pos
*vol_size[0] * vol_size[1])];

        Estimate[z_pos*(vol_size[0]*vol_size[1])+(x_pos*vol_size[0])+y_pos] = value;
                                Label[(y_pos + x_pos*vol_size[0] + z_pos *vol_size[0] *
vol_size[1])] = label +1;

                        }
                        count++;
                }
            }
        }
}
```

```
double distance(double* ima,int x,int y,int z,int nx,int ny,int nz,int f,int sx,int sy,int sz)

{

double d,acu,distancetotal,inc;
int i,j,k,ni1,nj1,ni2,nj2,nk1,nk2,kk;

acu=0;
distancetotal=0;

for(k=-f;k<=f;k++)
{
	for(i=-f;i<=f;i++)
	{
		for(j=-f;j<=f;j++)
		{
			ni1=x+i;
			nj1=y+j;
			nk1=z+k;
			ni2=nx+i;
			nj2=ny+j;
			nk2=nz+k;

			if(ni1<0) ni1=-ni1;
			if(nj1<0) nj1=-nj1;
			if(ni2<0) ni2=-ni2;
			if(nj2<0) nj2=-nj2;
			if(nk1<0) nk1=-nk1;
			if(nk2<0) nk2=-nk2;

			if(ni1>=sx) ni1=2*sx-ni1-1;
			if(nj1>=sy) nj1=2*sy-nj1-1;
			if(nk1>=sz) nk1=2*sz-nk1-1;
			if(ni2>=sx) ni2=2*sx-ni2-1;
			if(nj2>=sy) nj2=2*sy-nj2-1;
			if(nk2>=sz) nk2=2*sz-nk2-1;

			distancetotal = distancetotal + ((ima[nk1*(sx*sy)+(ni1*sy)+nj1]-
ima[nk2*(sx*sy)+(ni2*sy)+nj2])*(ima[nk1*(sx*sy)+(ni1*sy)+nj1]-ima[nk2*(sx*sy)+(ni2*sy)+nj2]));
			acu=acu + 1;
		}
	}
}

d=distancetotal/acu;

return d;
```

```
}


void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{

/*Declarations*/
mxArray *xData;
double *ima, *fima,*average;
mxArray *Mxmeans, *Mxvariances, *MxEstimate, *MxLabel;
double *means, *variances, *Estimate, *Label;
mxArray *pv;
double label,estimate,epsilon,mu1,var1,h,w,totalweight,wmax,d,mean,var,t1,t2,hh;
int init,Ndims,i,j,k,ii,jj,kk,ni,nj,nk,v,f,ndim,indice;
const int  *dims;

/*Copy input pointer x*/
xData = prhs[0];

/*Get matrix x*/
ima = mxGetPr(xData);

ndim = mxGetNumberOfDimensions(prhs[0]);
dims= mxGetDimensions(prhs[0]);

/*Copy input parameters*/
pv = prhs[1];
/*Get the Integer*/
v = (int)(mxGetScalar(pv));

pv = prhs[2];
f = (int)(mxGetScalar(pv));

pv = prhs[3];
h = (double)(mxGetScalar(pv));

hh=2*h*h;
Ndims = pow((2*f+1),ndim);

/*Allocate memory and assign output pointer*/

plhs[0] = mxCreateNumericArray(ndim,dims,mxDOUBLE_CLASS, mxREAL);
Mxmeans = mxCreateNumericArray(ndim,dims,mxDOUBLE_CLASS, mxREAL);
Mxvariances = mxCreateNumericArray(ndim,dims,mxDOUBLE_CLASS, mxREAL);
MxEstimate = mxCreateNumericArray(ndim,dims,mxDOUBLE_CLASS, mxREAL);
MxLabel = mxCreateNumericArray(ndim,dims,mxDOUBLE_CLASS, mxREAL);

average=(double*) malloc(Ndims*sizeof(double));
```

```
/*Get a pointer to the data space in our newly allocated memory*/
fima = mxGetPr(plhs[0]);
means = mxGetPr(Mxmeans);
variances = mxGetPr(Mxvariances);
Estimate = mxGetPr(MxEstimate);
Label = mxGetPr(MxLabel);

for (i = 0; i < dims[2] *dims[1] * dims[0];i++)
{
        Estimate[i] = 0.0;
        Label[i] = 0.0;
        fima[i] = 0.0;
}


for(k=0;k<dims[2];k++)
{
        for(i=0;i<dims[1];i++)
        {
                for(j=0;j<dims[0];j++)
                {
                        mean=0;
                        indice=0;
                        for(ii=-1;ii<=1;ii++)
                        {
                                for(jj=-1;jj<=1;jj++)
                                {
                                        for(kk=-1;kk<=1;kk++)
                                        {
                                                ni=i+ii;
                                                        nj=j+jj;
                                                nk=k+kk;

                                                if(ni<0) ni=-ni;
                                                if(nj<0) nj=-nj;
                                                if(nk<0) nk=-nk;
                                                if(ni>=dims[1]) ni=2*dims[1]-ni-1;
                                                if(nj>=dims[0]) nj=2*dims[0]-nj-1;
                                                if(nk>=dims[2]) nk=2*dims[2]-nk-1;


                                                mean = mean +
ima[nk*(dims[0]*dims[1])+(ni*dims[0])+nj];
                                                indice=indice+1;

                                        }
                                }
```

```
                        }
                        mean=mean/indice;
                        means[k*(dims[0]*dims[1])+(i*dims[0])+j]=mean;
                }
        }
}

for(k=0;k<dims[2];k++)
{
        for(i=0;i<dims[1];i++)
        {
                for(j=0;j<dims[0];j++)
                {
                        var=0;
                        indice=0;
                        for(ii=-1;ii<=1;ii++)
                        {
                                for(jj=-1;jj<=1;jj++)
                                {
                                        for(kk=-1;kk<=1;kk++)
                                        {
                                                ni=i+ii;
                                                nj=j+jj;
                                                nk=k+kk;
                                                        if(ni>=0 && nj>=0 && nk>0 && ni<dims[1]
&& nj<dims[0] && nk<dims[2])
                                                        {
                                                        var = var +
(ima[nk*(dims[0]*dims[1])+(ni*dims[0])+nj]-
means[k*(dims[0]*dims[1])+(i*dims[0])+j])*(ima[nk*(dims[0]*dims[1])+(ni*dims[0])+nj]-
means[k*(dims[0]*dims[1])+(i*dims[0])+j]);
                                                        indice=indice+1;
                                                        }
                                        }
                                }
                        }
                        var=var/(indice-1);
                        variances[k*(dims[0]*dims[1])+(i*dims[0])+j]=var;
                }
        }
}

/*filter*/

epsilon = 0.00001;
mu1 = 0.95;
var1 = 0.5;
init = 0;
```

```
for(k=0;k<dims[2];k+=2)
{
	for(i=0;i<dims[1];i+=2)
	{
		for(j=0;j<dims[0];j+=2)
		{

			for (init=0 ; init < Ndims; init++)
			{
				average[init]=0.0;
			}
			/*average=0;*/

			totalweight=0.0;


			if ((means[k*(dims[0]*dims[1])+(i*dims[0])+j])>epsilon &&
(variances[k*(dims[0]*dims[1])+(i*dims[0])+j]>epsilon))
			{
				wmax=0.0;

				for(kk=-v;kk<=v;kk++)
				{
					for(ii=-v;ii<=v;ii++)
					{
						for(jj=-v;jj<=v;jj++)
						{
							ni=i+ii;
							nj=j+jj;
							nk=k+kk;

							if(ii==0 && jj==0 && kk==0) continue;

							if(ni>=0 && nj>=0 && nk>=0 && ni<dims[1]
&& nj<dims[0] && nk<dims[2])
							{

							if
((means[nk*(dims[0]*dims[1])+(ni*dims[0])+nj])> epsilon &&
(variances[nk*(dims[0]*dims[1])+(ni*dims[0])+nj]>epsilon))
{

t1 =
(means[k*(dims[0]*dims[1])+(i*dims[0])+j])/(means[nk*(dims[0]*dims[1])+(ni*dims[0])+nj]);
t2 =
(variances[k*(dims[0]*dims[1])+(i*dims[0])+j])/(variances[nk*(dims[0]*dims[1])+(ni*dims[0])+nj]);
```

```
                                                    if(t1>mu1 && t1<(1/mu1)
&& t2>var1 && t2<(1/var1))
                                                    {



                                d=distance(ima,i,j,k,ni,nj,nk,f,dims[1],dims[0],dims[2]);

                                                            w = exp(-d/(h*h));

                                                            if(w>wmax) wmax =
w;

        Average_block(ima,ni,nj,nk,f,average,w,dims);

                                                            totalweight =
totalweight + w;
                                                    }
                                                }

                                            }
                                        }
                                    }

                            }

                            if(wmax==0.0) wmax=1.0;


                            Average_block(ima,i,j,k,f,average,wmax,dims);

                            totalweight = totalweight + wmax;



                            if(totalweight != 0.0)
                            Value_block(Estimate,Label,i,j,k,f,average,totalweight,dims);


                    }


                    Average_block(ima,i,j,k,f,average,wmax,dims);
```

```
                }
        }
}

label = 0.0;
estimate = 0.0;


/* Aggregation of the estimators (i.e. means computation) */
for (k = 0; k < dims[2]; k++ )
{
        for (i = 0; i < dims[1]; i++ )
        {
                for (j = 0; j < dims[0]; j++ )
                {
                        label = Label[k*(dims[0]*dims[1])+(i*dims[0])+j];
                        if (label == 0.0)
                        {
                                fima[k*(dims[0]*dims[1])+(i*dims[1])+j] =
ima[k*(dims[0]*dims[1])+(i*dims[0])+j];

                        }
                        else
                        {
                                estimate = Estimate[k*(dims[0]*dims[1])+(i*dims[0])+j];
                                estimate = (estimate/label);
                                fima[k*(dims[0]*dims[1])+(i*dims[0])+j]=estimate;

                        }
                }
        }
}



return;

}
```

APPENDIX 5: LMMSE COMBINATION

```
=========================================================================
% Rician noise generation using two orthogal Gaussian  generators
% copyright © 2016 Kenneth Kagoiya
% Permission to use, copy or modify this software and its document.

% This program implements method three and LMMSE diffusion weighted MRI %
wavelet based  algorithm with bilateral feature enhancement
-------------------------------------------------------------------------
I = imread('C:/torso.bmp');
for phi= 0:0.375:1.5
II=I*cos(phi)
J = imnoise(II,'gaussian',0.0,0.0005);
J = imresize(J,[ 480 480])
figure,imshow(J)
K=I*sin(phi)
K = imnoise(K,'gaussian',0.0,0.0005);
K = imresize(K,[ 480 480])
figure,imshow(K)
L=(J+K)*0.707
figure,imshow(L)
end
val1 = std2(I)
val2 = mean2(I)
val3 = std2(II)
val4 = mean2(II)
val5 = std2(K)
val6 = mean2(K)
val7 = std2(K)
val8 = mean2(K)

% BFILTER2 Two dimensional bilateral filtering.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Pre-process input and select appropriate filter.
function B = bfilter2(A,w,sigma)

% Verify that the input image exists and is valid.
if ~exist('A','var') || isempty(A)
   error('Input image A is undefined or invalid.');
end
if ~isfloat(A) || ~sum([1,3] == size(A,3)) || ...
      min(A(:)) < 0 || max(A(:)) > 1
   error(['Input image A must be a double precision ',...
          'matrix of size NxMx1 or NxMx3 on the closed ',...
          'interval [0,1].']);
end

% Verify bilateral filter window size.
if ~exist('w','var') || isempty(w) || ...
      numel(w) ~= 1 || w < 1
   w = 5;
end
w = ceil(w);

% Verify bilateral filter standard deviations.
```

```
if ~exist('sigma','var') || isempty(sigma) || ...
      numel(sigma) ~= 2 || sigma(1) <= 0 || sigma(2) <= 0
   sigma = [3 0.1];
end

% Apply either grayscale or color bilateral filtering.
if size(A,3) == 1
   B = bfltGray(A,w,sigma(1),sigma(2));
else
   B = bfltColor(A,w,sigma(1),sigma(2));
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Implements bilateral filtering for grayscale images.
function B = bfltGray(A,w,sigma_d,sigma_r)

% Pre-compute Gaussian distance weights.
[X,Y] = meshgrid(-w:w,-w:w);
G = exp(-(X.^2+Y.^2)/(2*sigma_d^2));

% Create waitbar.
h = waitbar(0,'Applying bilateral filter...');
set(h,'Name','Bilateral Filter Progress');

% Apply bilateral filter.
dim = size(A);
B = zeros(dim);
for i = 1:dim(1)
   for j = 1:dim(2)

         % Extract local region.
         iMin = max(i-w,1);
         iMax = min(i+w,dim(1));
         jMin = max(j-w,1);
         jMax = min(j+w,dim(2));
         I = A(iMin:iMax,jMin:jMax);

         % Compute Gaussian intensity weights.
         H = exp(-(I-A(i,j)).^2/(2*sigma_r^2));

         % Calculate bilateral filter response.
         F = H.*G((iMin:iMax)-i+w+1,(jMin:jMax)-j+w+1);
         B(i,j) = sum(F(:).*I(:))/sum(F(:));

   end
   waitbar(i/dim(1));
end

% Close waitbar.
close(h);


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

```
% Implements bilateral filter for color images.
function B = bfltColor(A,w,sigma_d,sigma_r)

% Convert input sRGB image to CIELab color space.
if exist('applycform','file')
   A = applycform(A,makecform('srgb2lab'));
else
   A = colorspace('Lab<-RGB',A);
end

% Pre-compute Gaussian domain weights.
[X,Y] = meshgrid(-w:w,-w:w);
G = exp(-(X.^2+Y.^2)/(2*sigma_d^2));

% Rescale range variance (using maximum luminance).
sigma_r = 100*sigma_r;

% Create waitbar.
h = waitbar(0,'Applying bilateral filter...');
set(h,'Name','Bilateral Filter Progress');

% Apply bilateral filter.
dim = size(A);
B = zeros(dim);
for i = 1:dim(1)
   for j = 1:dim(2)

         % Extract local region.
         iMin = max(i-w,1);
         iMax = min(i+w,dim(1));
         jMin = max(j-w,1);
         jMax = min(j+w,dim(2));
         I = A(iMin:iMax,jMin:jMax,:);

         % Compute Gaussian range weights.
         dL = I(:,:,1)-A(i,j,1);
         da = I(:,:,2)-A(i,j,2);
         db = I(:,:,3)-A(i,j,3);
         H = exp(-(dL.^2+da.^2+db.^2)/(2*sigma_r^2));

         % Calculate bilateral filter response.
         F = H.*G((iMin:iMax)-i+w+1,(jMin:jMax)-j+w+1);
         norm_F = sum(F(:));
         B(i,j,1) = sum(sum(F.*I(:,:,1)))/norm_F;
         B(i,j,2) = sum(sum(F.*I(:,:,2)))/norm_F;
         B(i,j,3) = sum(sum(F.*I(:,:,3)))/norm_F;

   end
   waitbar(i/dim(1));
end

% Convert filtered image back to sRGB color space.
if exist('applycform','file')
   B = applycform(B,makecform('lab2srgb'));
else
```

```
   B = colorspace('RGB<-Lab',B);
end

% Close waitbar.
close(h);
% BFILTER2 Two dimensional bilateral filtering for feature enhancement.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Pre-process input and select appropriate filter.
function B = bfilter2(A,w,sigma)

% Verify that the input image exists and is valid.
if ~exist('A','var') || isempty(A)
   error('Input image A is undefined or invalid.');
end
if ~isfloat(A) || ~sum([1,3] == size(A,3)) || ...
      min(A(:)) < 0 || max(A(:)) > 1
   error(['Input image A must be a double precision ',...
          'matrix of size NxMx1 or NxMx3 on the closed ',...
          'interval [0,1].']);
end

% Verify bilateral filter window size.
if ~exist('w','var') || isempty(w) || ...
      numel(w) ~= 1 || w < 1
   w = 5;
end
w = ceil(w);

% Verify bilateral filter standard deviations.
if ~exist('sigma','var') || isempty(sigma) || ...
      numel(sigma) ~= 2 || sigma(1) <= 0 || sigma(2) <= 0
   sigma = [3 0.1];
end

% Apply either grayscale or color bilateral filtering.
if size(A,3) == 1
   B = bfltGray(A,w,sigma(1),sigma(2));
else
   B = bfltColor(A,w,sigma(1),sigma(2));
end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Implements bilateral filtering for grayscale images.
function B = bfltGray(A,w,sigma_d,sigma_r)

% Pre-compute Gaussian distance weights.
[X,Y] = meshgrid(-w:w,-w:w);
G = exp(-(X.^2+Y.^2)/(2*sigma_d^2));

% Create waitbar.
h = waitbar(0,'Applying bilateral filter...');
set(h,'Name','Bilateral Filter Progress');
```

```
% Apply bilateral filter.
dim = size(A);
B = zeros(dim);
for i = 1:dim(1)
   for j = 1:dim(2)

         % Extract local region.
         iMin = max(i-w,1);
         iMax = min(i+w,dim(1));
         jMin = max(j-w,1);
         jMax = min(j+w,dim(2));
         I = A(iMin:iMax,jMin:jMax);

         % Compute Gaussian intensity weights.
         H = exp(-(I-A(i,j)).^2/(2*sigma_r^2));

         % Calculate bilateral filter response.
         F = H.*G((iMin:iMax)-i+w+1,(jMin:jMax)-j+w+1);
         B(i,j) = sum(F(:).*I(:))/sum(F(:));

   end
   waitbar(i/dim(1));
end

% Close waitbar.
close(h);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Implements bilateral filter for color images.
function B = bfltColor(A,w,sigma_d,sigma_r)

% Convert input sRGB image to CIELab color space.
if exist('applycform','file')
   A = applycform(A,makecform('srgb2lab'));
else
   A = colorspace('Lab<-RGB',A);
end

% Pre-compute Gaussian domain weights.
[X,Y] = meshgrid(-w:w,-w:w);
G = exp(-(X.^2+Y.^2)/(2*sigma_d^2));

% Rescale range variance (using maximum luminance).
sigma_r = 100*sigma_r;

% Create waitbar.
h = waitbar(0,'Applying bilateral filter...');
set(h,'Name','Bilateral Filter Progress');

% Apply bilateral filter.
dim = size(A);
B = zeros(dim);
for i = 1:dim(1)
```

```
   for j = 1:dim(2)

         % Extract local region.
         iMin = max(i-w,1);
         iMax = min(i+w,dim(1));
         jMin = max(j-w,1);
         jMax = min(j+w,dim(2));
         I = A(iMin:iMax,jMin:jMax,:);

         % Compute Gaussian range weights.
         dL = I(:,:,1)-A(i,j,1);
         da = I(:,:,2)-A(i,j,2);
         db = I(:,:,3)-A(i,j,3);
         H = exp(-(dL.^2+da.^2+db.^2)/(2*sigma_r^2));

         % Calculate bilateral filter response.
         F = H.*G((iMin:iMax)-i+w+1,(jMin:jMax)-j+w+1);
         norm_F = sum(F(:));
         B(i,j,1) = sum(sum(F.*I(:,:,1)))/norm_F;
         B(i,j,2) = sum(sum(F.*I(:,:,2)))/norm_F;
         B(i,j,3) = sum(sum(F.*I(:,:,3)))/norm_F;

   end
   waitbar(i/dim(1));
end

% Convert filtered image back to sRGB color space.
if exist('applycform','file')
   B = applycform(B,makecform('lab2srgb'));
else
   B = colorspace('RGB<-Lab',B);
end

% Close waitbar.
close(h);

Im=L
I_est=MRI_lmmse(Im,Ws,varargin)

function I_est=MRI_lmmse(Im,Ws,varargin)

%MRI_LMMSE Linear minimum Mean Square error Estimation of MRI data
%
% Filter assumes a Rician distribution with a Rayleigh Background
% Noise estimation is automatically performed using the background.
%
% Usage
%
%   I_est=MRI_lmmse(Im,[7,7],[Noise method]);
%
%
% DEFAULT:
%
%   I_est=MRI_lmmse(Im,[7,7]);
%         Noise estimationmethod: mode2N
```

```
%
%   INPUTS:
%        - Im: input image
%        - Ws: Size [M,N] of the square window used for local estimation
%              Odd number recomended
%
%  Noise Estimation Method:
%       y= MRI_lmmse(...,'sigma',sigma) Standard deviation of noise given
%                    - sigma: Standar deviation of noise
%       y= MRI_lmmse(...,'bckN2',threshold) Estimate the noise from the
%                background of the image based on order 2 moment
%                    - threshold: Threshold value for the mask
%       y= MRI_lmmse(...,'bckNm',threshold) Estimate the noise from the
%                background of the image based on the mean
%                    - threshold: Threshold value for the mask
%       y= MRI_lmmse(...,'momentN') Estimate the noise using the methods
of
%                moments
%
%   The following methods may use a background mask
%
%       y= MRI_lmmse(...,'mask',threshold) A background mask is used.
%                    - threshold: Threshold value for the mask
%       y= MRI_lmmse(...,'histoN') Estimate the noise using the method
%                based on the mode of the histogram [Sijbers06]
%       y= MRI_lmmse(...,'mode2N') Estimate the noise using the mode of
the
%                local order 2 moment [Aja06]
%       y= MRI_lmmse(...,'modeMN') Estimate the noise using the mode of
the
%                local mean [Aja06]
%       y= MRI_lmmse(...,'modeVN') Estimate the noise using the mode of
the
%                local variance [Aja06]
%       y= MRI_lmmse(...,'modeVN_NI') Estimate the noise using the mode of
the
%

[mask,thresM,noise,sigma] = parse_inputs(varargin{:});

%Noise Estimation-------------------------------------------
%LOCAL STATISTICS
%Order 2 moment
En=filter2(ones(Ws), Im.^2) / (prod(Ws));
%Mean
Mn=filter2(ones(Ws), Im) / (prod(Ws));
%Variance
Vn=(prod(Ws)/(prod(Ws)-1)).*(En-Mn.^2);


if noise==0
%Sigma given
    sigma2=sigma.^2;
elseif noise==1
    mask =im2bw(1-double(imfill(Im>thresM,'holes')));
    sigma2=0.5.*(sum((Im(mask)).^2))./sum(mask(:));
```

```
    sigma=sqrt(sigma2);
elseif noise==2
    mask =im2bw(1-double(imfill(Im>thresM,'holes')));
    sigma=sqrt(2/pi).*(sum(Im(mask))./sum(mask(:));
    sigma2=sigma^2;
elseif noise==3
    M2=mean(Im(:).^2);
    M4=mean(Im(:).^4);
    sigma2=0.5.*(M2-sqrt(sqrt(abs(2*M2^2-M4))));
    sigma=sqrt(sigma2);
elseif noise==4
    if mask==0
        I2=round(Im);
        Tp=max(I2(:));
        Tpm=min(I2(:));
        [h,x]=hist(I2(:),Tp);
        sigma=x(argmax(h));
    else
       mask =im2bw(1-double(imfill(Im>thresM,'holes')));
       Tp=max(I2(mask));
       [h,x]=hist(I2(mask),Tp);
       sigma=x(argmax(h));
    end
    sigma2=sigma^2;
elseif noise==5
    if mask==0
        sigma2=(prod(Ws)/(prod(Ws)-1)).*(moda(En,1000)./2);
    else
       mask =im2bw(1-double(imfill(Im>thresM,'holes')));
       sigma2=(prod(Ws)/(prod(Ws)-1)).*(moda(En(mask),1000)./2);
    end
    sigma=sqrt(sigma2);
elseif noise==6
    if mask==0
        sigma=sqrt(2/pi).*moda(Mn,1000);
    else
        mask =im2bw(1-double(imfill(Im>thresM,'holes')));
        sigma=sqrt(2/pi).*moda(Mn(mask),1000);
    end
    sigma2=sigma^2;
elseif noise==7
    if mask==0
        sigma2=((2/(4-pi)).*moda(Vn,1000));
    else
        mask =im2bw(1-double(imfill(Im>thresM,'holes')));
        sigma2=(2/(4-pi)).*moda(Vn(mask),1000);
    end
    sigma=sqrt(sigma2);
elseif noise==8
    if mask==0
        sigma2=((prod(Ws)-1)/(prod(Ws)-3)).*moda(Vn,1000);
    else
        mask =im2bw(double(imfill(Im>thresM,'holes')));
        sigma2=((prod(Ws)-1)/(prod(Ws)-3)).*moda(Vn,1000);
    end
    sigma=sqrt(sigma2);
end
```

```
%End Noise estimation-----------------------------------------------


%FILTERING----------------------------------------------

Qua=filter2(ones(Ws),Im.^4)./prod(Ws);
%Squ=filter2(ones(Ws),Im.^2)./prod(Ws);
Squ=En;
%Squ=Squ.*(Squ>2.*sigma2)+(Squ<=2.*sigma2).*2.*sigma2;
%Qua=Qua.*(Qua>8.*sigma2^2)+(Qua<=8.*sigma2^2).*8.*sigma2^2;


K1=1+(4.*sigma2^2-4.*sigma2.*Squ)./(Qua-Squ.^2);
K1=max(K1,0);
I_est=sqrt(Squ-2.*sigma2+K1.*(Im.^2-Squ));
%I_est= sqrt(Im.^2-2.*sigma2+0.5.*(1-K1).*(Squ-Ac.^2));
I_est=abs(I_est);


%-------------------------------------------------------------------------
-
function [mask,thresM,noise,sigma] = parse_inputs(varargin)
dfsteppos = -1;
mask=0;
thresM=0;
noise=5;
sigma=0;



for i = 1 : length(varargin)
   flag = 0;
   if i == dfsteppos
      flag = 1;
   end
   if strcmp(varargin{i},'mask')
       mask=1;
      thresM = varargin{i+1};
      flag = 1;
      dfsteppos = i+1;
    elseif strcmp(varargin{i},'sigma')
      noise=0;
      sigma = varargin{i+1};
      flag = 1;
      dfsteppos = i+1;
   elseif strcmp(varargin{i},'bckN2')
      noise=1;
      thresM = varargin{i+1};
      flag = 1;
      dfsteppos = i+1;
   elseif strcmp(varargin{i},'bckNm')
      noise=2;
      thresM = varargin{i+1};
      flag = 1;
```

```
      dfsteppos = i+1;
   elseif strcmp(varargin{i},'momentN')
      noise=3;
      flag = 1;
    elseif strcmp(varargin{i},'histoN')
      noise=4;
      flag = 1;
   elseif strcmp(varargin{i},'mode2N')
      noise=5;
      flag = 1;
   elseif strcmp(varargin{i},'modeMN')
      noise=6;
      flag = 1;
   elseif strcmp(varargin{i},'modeVN')
      noise=7;
      flag = 1;
    elseif strcmp(varargin{i},'modeVN_NI')
      noise=8;
      flag = 1;
    end
   if flag == 0
      error('Too many parameters !')
      return
   end
end

%-----------------------------------------------------------------
function m=moda(u,N)
% MODA   Mode of a distribution
%
%    m=MODE(u,N) calculates the mode of the set of data "u" using the
histogram.
%    To avoid outliers, for the calculation are only taken into account
those
%    values in [mean-2sigma, mean+2sigma];
%
%    INPUT:
%
%   - u (set of data)
%       - N: Number of points for the histogram. If N=0 then 5000 points
are
%            considered
%
%   Author: Santiago Aja Fernandez
%   www.lpi.tel.uva.es/~santi
%   LOCAL STATISTICS TOOLBOX
%
%   Modified: Feb 01 2008
%
u=double(u);
if N==0
    N=5000;
end
M1=mean(u(:));
V1=std(u(:));
C2=u( (u(:)>=(M1-2*V1)) & (u(:)<=(M1+2*V1)) ) ;
%C2=u;
```

```
[h,x]=hist(C2,N);
[M,M2]=max(h);
m=x(M2);
%
function [fima]=mixingsubband(fimau,fimao)

s = size(fimau);

p(1) = 2^(ceil(log2(s(1))));
p(2) = 2^(ceil(log2(s(2))));
p(3) = 2^(ceil(log2(s(3))));

pad1 = zeros(p(1),p(2),p(3));
pad2 = pad1;
pad1(1:s(1),1:s(2),1:s(3)) = fimau(:,:,:);
pad2(1:s(1),1:s(2),1:s(3)) = fimao(:,:,:);

[af, sf] = farras;
w1 = dwt3D(pad1,1,af);
w2 = dwt3D(pad2,1,af);

w1{1}{3} = w2{1}{3};
w1{1}{5} = w2{1}{5};
w1{1}{6} = w2{1}{6};
w1{1}{7} = w2{1}{7};

fima = idwt3D(w1,1,sf);
fima = fima(1:s(1),1:s(2),1:s(3));

% NAN checking
ind=find(isnan(fima(:)));
fima(ind)=fimau(ind);

% negative checking (only for rician noise mixing)
ind=find(fima<0);
fima(ind)=0;



% s = size(fimau);
%
% p(1) = 2^(ceil(log2(s(1))));
% p(2) = 2^(ceil(log2(s(2))));
% p(3) = 2^(ceil(log2(s(3))));
%
% pad1 = zeros(p(1),p(2),p(3));
% pad2=pad1;
% pad1(1:s(1),1:s(2),1:s(3)) = fimau(:,:,:);
% pad2(1:s(1),1:s(2),1:s(3)) = fimao(:,:,:);
%
% [af, sf] = farras;
% w1 = dwt3D(pad1,1,af);
% w2 = dwt3D(pad2,1,af);
%
%   w1{1}{1} = (w1{1}{1} + w2{1}{1})/2;
%   w1{1}{2} = (w1{1}{2} + w2{1}{2})/2;
```

```
%   w1{1}{3} = w2{1}{3};
%   w1{1}{4} = (w1{1}{4} + w2{1}{4})/2;
%   w1{1}{5} = w2{1}{5};
%   w1{1}{6} = w2{1}{6};
%   w1{1}{7} = w2{1}{7};
%
% fima = idwt3D(w1,1,sf);
% fima = fima(1:s(1),1:s(2),1:s(3));

% BFILTER2 Two dimensional bilateral filtering   final stage.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Pre-process input and select appropriate filter.
function B = bfilter2(A,w,sigma)

% Verify that the input image exists and is valid.
if ~exist('A','var') || isempty(A)
   error('Input image A is undefined or invalid.');
end
if ~isfloat(A) || ~sum([1,3] == size(A,3)) || ...
      min(A(:)) < 0 || max(A(:)) > 1
   error(['Input image A must be a double precision ',...
          'matrix of size NxMx1 or NxMx3 on the closed ',...
          'interval [0,1].']);
end

% Verify bilateral filter window size.
if ~exist('w','var') || isempty(w) || ...
      numel(w) ~= 1 || w < 1
   w = 5;
end
w = ceil(w);

% Verify bilateral filter standard deviations.
if ~exist('sigma','var') || isempty(sigma) || ...
      numel(sigma) ~= 2 || sigma(1) <= 0 || sigma(2) <= 0
   sigma = [3 0.1];
end

% Apply either grayscale or color bilateral filtering.
if size(A,3) == 1
   B = bfltGray(A,w,sigma(1),sigma(2));
else
   B = bfltColor(A,w,sigma(1),sigma(2));
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Implements bilateral filtering for grayscale images.
function B = bfltGray(A,w,sigma_d,sigma_r)

% Pre-compute Gaussian distance weights.
[X,Y] = meshgrid(-w:w,-w:w);
G = exp(-(X.^2+Y.^2)/(2*sigma_d^2));
```

```
% Create waitbar.
h = waitbar(0,'Applying bilateral filter...');
set(h,'Name','Bilateral Filter Progress');

% Apply bilateral filter.
dim = size(A);
B = zeros(dim);
for i = 1:dim(1)
   for j = 1:dim(2)

         % Extract local region.
         iMin = max(i-w,1);
         iMax = min(i+w,dim(1));
         jMin = max(j-w,1);
         jMax = min(j+w,dim(2));
         I = A(iMin:iMax,jMin:jMax);

         % Compute Gaussian intensity weights.
         H = exp(-(I-A(i,j)).^2/(2*sigma_r^2));

         % Calculate bilateral filter response.
         F = H.*G((iMin:iMax)-i+w+1,(jMin:jMax)-j+w+1);
         B(i,j) = sum(F(:).*I(:))/sum(F(:));

   end
   waitbar(i/dim(1));
end

% Close waitbar.
close(h);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Implements bilateral filter for color images.
function B = bfltColor(A,w,sigma_d,sigma_r)

% Convert input sRGB image to CIELab color space.
if exist('applycform','file')
   A = applycform(A,makecform('srgb2lab'));
else
   A = colorspace('Lab<-RGB',A);
end

% Pre-compute Gaussian domain weights.
[X,Y] = meshgrid(-w:w,-w:w);
G = exp(-(X.^2+Y.^2)/(2*sigma_d^2));

% Rescale range variance (using maximum luminance).
sigma_r = 100*sigma_r;

% Create waitbar.
h = waitbar(0,'Applying bilateral filter...');
set(h,'Name','Bilateral Filter Progress');
```

```
% Apply bilateral filter.
dim = size(A);
B = zeros(dim);
for i = 1:dim(1)
   for j = 1:dim(2)

         % Extract local region.
         iMin = max(i-w,1);
         iMax = min(i+w,dim(1));
         jMin = max(j-w,1);
         jMax = min(j+w,dim(2));
         I = A(iMin:iMax,jMin:jMax,:);

         % Compute Gaussian range weights.
         dL = I(:,:,1)-A(i,j,1);
         da = I(:,:,2)-A(i,j,2);
         db = I(:,:,3)-A(i,j,3);
         H = exp(-(dL.^2+da.^2+db.^2)/(2*sigma_r^2));

         % Calculate bilateral filter response.
         F = H.*G((iMin:iMax)-i+w+1,(jMin:jMax)-j+w+1);
         norm_F = sum(F(:));
         B(i,j,1) = sum(sum(F.*I(:,:,1)))/norm_F;
         B(i,j,2) = sum(sum(F.*I(:,:,2)))/norm_F;
         B(i,j,3) = sum(sum(F.*I(:,:,3)))/norm_F;

   end
   waitbar(i/dim(1));
end

% Convert filtered image back to sRGB color space.
if exist('applycform','file')
   B = applycform(B,makecform('lab2srgb'));
else
   B = colorspace('RGB<-Lab',B);
end

% Close waitbar.
close(h);
```

APPENDIX 6: TOTAL VARIATIONAL COMBINATION

```
=========================================================================
% Rician noise generation using two orthogal Gaussian  generators
% copyright © 2016 Kenneth Kagoiya
% Permission to use, copy or modify this software and its document.

% This program implements method method four a total valuation wavelet
based structural MRI denoising method with bilateral  feature enhancement
-------------------------------------------------------------------
I = imread('C:/torso.bmp');
I = rgb2gray(I);
for phi= 0:0.375:1.5
II=I*cos(phi)
J = imnoise(II,'gaussian',0.0,0.0005);
J = imresize(J,[ 480 480])
figure,imshow(J)
K=I*sin(phi)
K = imnoise(K,'gaussian',0.0,0.0005);
K = imresize(K,[ 480 480])
figure,imshow(K)
L=(J+K)*0.75
figure,imshow(L)
end
val1 = std2(I)
val2 = mean2(I)
val3 = std2(II)
val4 = mean2(II)
val5 = std2(K)
val6 = mean2(K)
val7 = std2(K)
val8 = mean2(K)
% BFILTER2 Two dimensional bilateral filtering   boundary enhancement.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Pre-process input and select appropriate filter.
function B = bfilter2(A,w,sigma)

% Verify that the input image exists and is valid.
if ~exist('A','var') || isempty(A)
   error('Input image A is undefined or invalid.');
end
if ~isfloat(A) || ~sum([1,3] == size(A,3)) || ...
      min(A(:)) < 0 || max(A(:)) > 1
   error(['Input image A must be a double precision ',...
          'matrix of size NxMx1 or NxMx3 on the closed ',...
          'interval [0,1].']);
end

% Verify bilateral filter window size.
if ~exist('w','var') || isempty(w) || ...
      numel(w) ~= 1 || w < 1
   w = 5;
end
```

```
w = ceil(w);

% Verify bilateral filter standard deviations.
if ~exist('sigma','var') || isempty(sigma) || ...
      numel(sigma) ~= 2 || sigma(1) <= 0 || sigma(2) <= 0
   sigma = [3 0.1];
end

% Apply either grayscale or color bilateral filtering.
if size(A,3) == 1
   B = bfltGray(A,w,sigma(1),sigma(2));
else
   B = bfltColor(A,w,sigma(1),sigma(2));
end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Implements bilateral filtering for grayscale images.
function B = bfltGray(A,w,sigma_d,sigma_r)

% Pre-compute Gaussian distance weights.
[X,Y] = meshgrid(-w:w,-w:w);
G = exp(-(X.^2+Y.^2)/(2*sigma_d^2));

% Create waitbar.
h = waitbar(0,'Applying bilateral filter...');
set(h,'Name','Bilateral Filter Progress');

% Apply bilateral filter.
dim = size(A);
B = zeros(dim);
for i = 1:dim(1)
   for j = 1:dim(2)

         % Extract local region.
         iMin = max(i-w,1);
         iMax = min(i+w,dim(1));
         jMin = max(j-w,1);
         jMax = min(j+w,dim(2));
         I = A(iMin:iMax,jMin:jMax);

         % Compute Gaussian intensity weights.
         H = exp(-(I-A(i,j)).^2/(2*sigma_r^2));

         % Calculate bilateral filter response.
         F = H.*G((iMin:iMax)-i+w+1,(jMin:jMax)-j+w+1);
         B(i,j) = sum(F(:).*I(:))/sum(F(:));

   end
   waitbar(i/dim(1));
end
```

```
% Close waitbar.
close(h);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Implements bilateral filter for color images.
function B = bfltColor(A,w,sigma_d,sigma_r)

% Convert input sRGB image to CIELab color space.
if exist('applycform','file')
   A = applycform(A,makecform('srgb2lab'));
else
   A = colorspace('Lab<-RGB',A);
end

% Pre-compute Gaussian domain weights.
[X,Y] = meshgrid(-w:w,-w:w);
G = exp(-(X.^2+Y.^2)/(2*sigma_d^2));

% Rescale range variance (using maximum luminance).
sigma_r = 100*sigma_r;

% Create waitbar.
h = waitbar(0,'Applying bilateral filter...');
set(h,'Name','Bilateral Filter Progress');

% Apply bilateral filter.
dim = size(A);
B = zeros(dim);
for i = 1:dim(1)
   for j = 1:dim(2)

         % Extract local region.
         iMin = max(i-w,1);
         iMax = min(i+w,dim(1));
         jMin = max(j-w,1);
         jMax = min(j+w,dim(2));
         I = A(iMin:iMax,jMin:jMax,:);

         % Compute Gaussian range weights.
         dL = I(:,:,1)-A(i,j,1);
         da = I(:,:,2)-A(i,j,2);
         db = I(:,:,3)-A(i,j,3);
         H = exp(-(dL.^2+da.^2+db.^2)/(2*sigma_r^2));

         % Calculate bilateral filter response.
         F = H.*G((iMin:iMax)-i+w+1,(jMin:jMax)-j+w+1);
         norm_F = sum(F(:));
         B(i,j,1) = sum(sum(F.*I(:,:,1)))/norm_F;
         B(i,j,2) = sum(sum(F.*I(:,:,2)))/norm_F;
         B(i,j,3) = sum(sum(F.*I(:,:,3)))/norm_F;

   end
```

```
   waitbar(i/dim(1));
end

% Convert filtered image back to sRGB color space.
if exist('applycform','file')
   B = applycform(B,makecform('lab2srgb'));
else
   B = colorspace('RGB<-Lab',B);
end

% Close waitbar.
close(h);

% BFILTER2 Two dimensional bilateral filtering   edge enhancement.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Pre-process input and select appropriate filter.
function B = bfilter2(A,w,sigma)

% Verify that the input image exists and is valid.
if ~exist('A','var') || isempty(A)
   error('Input image A is undefined or invalid.');
end
if ~isfloat(A) || ~sum([1,3] == size(A,3)) || ...
      min(A(:)) < 0 || max(A(:)) > 1
   error(['Input image A must be a double precision ',...
          'matrix of size NxMx1 or NxMx3 on the closed ',...
          'interval [0,1].']);
end

% Verify bilateral filter window size.
if ~exist('w','var') || isempty(w) || ...
      numel(w) ~= 1 || w < 1
   w = 5;
end
w = ceil(w);

% Verify bilateral filter standard deviations.
if ~exist('sigma','var') || isempty(sigma) || ...
      numel(sigma) ~= 2 || sigma(1) <= 0 || sigma(2) <= 0
   sigma = [3 0.1];
end

% Apply either grayscale or color bilateral filtering.
if size(A,3) == 1
   B = bfltGray(A,w,sigma(1),sigma(2));
else
   B = bfltColor(A,w,sigma(1),sigma(2));
end


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

```
% Implements bilateral filtering for grayscale images.
function B = bfltGray(A,w,sigma_d,sigma_r)

% Pre-compute Gaussian distance weights.
[X,Y] = meshgrid(-w:w,-w:w);
G = exp(-(X.^2+Y.^2)/(2*sigma_d^2));

% Create waitbar.
h = waitbar(0,'Applying bilateral filter...');
set(h,'Name','Bilateral Filter Progress');

% Apply bilateral filter.
dim = size(A);
B = zeros(dim);
for i = 1:dim(1)
   for j = 1:dim(2)

         % Extract local region.
         iMin = max(i-w,1);
         iMax = min(i+w,dim(1));
         jMin = max(j-w,1);
         jMax = min(j+w,dim(2));
         I = A(iMin:iMax,jMin:jMax);

         % Compute Gaussian intensity weights.
         H = exp(-(I-A(i,j)).^2/(2*sigma_r^2));

         % Calculate bilateral filter response.
         F = H.*G((iMin:iMax)-i+w+1,(jMin:jMax)-j+w+1);
         B(i,j) = sum(F(:).*I(:))/sum(F(:));

   end
   waitbar(i/dim(1));
end

% Close waitbar.
close(h);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Implements bilateral filter for color images.
function B = bfltColor(A,w,sigma_d,sigma_r)

% Convert input sRGB image to CIELab color space.
if exist('applycform','file')
   A = applycform(A,makecform('srgb2lab'));
else
   A = colorspace('Lab<-RGB',A);
end

% Pre-compute Gaussian domain weights.
[X,Y] = meshgrid(-w:w,-w:w);
```

```
G = exp(-(X.^2+Y.^2)/(2*sigma_d^2));

% Rescale range variance (using maximum luminance).
sigma_r = 100*sigma_r;

% Create waitbar.
h = waitbar(0,'Applying bilateral filter...');
set(h,'Name','Bilateral Filter Progress');

% Apply bilateral filter.
dim = size(A);
B = zeros(dim);
for i = 1:dim(1)
   for j = 1:dim(2)

         % Extract local region.
         iMin = max(i-w,1);
         iMax = min(i+w,dim(1));
         jMin = max(j-w,1);
         jMax = min(j+w,dim(2));
         I = A(iMin:iMax,jMin:jMax,:);

         % Compute Gaussian range weights.
         dL = I(:,:,1)-A(i,j,1);
         da = I(:,:,2)-A(i,j,2);
         db = I(:,:,3)-A(i,j,3);
         H = exp(-(dL.^2+da.^2+db.^2)/(2*sigma_r^2));

         % Calculate bilateral filter response.
         F = H.*G((iMin:iMax)-i+w+1,(jMin:jMax)-j+w+1);
         norm_F = sum(F(:));
         B(i,j,1) = sum(sum(F.*I(:,:,1)))/norm_F;
         B(i,j,2) = sum(sum(F.*I(:,:,2)))/norm_F;
         B(i,j,3) = sum(sum(F.*I(:,:,3)))/norm_F;

   end
   waitbar(i/dim(1));
end

% Convert filtered image back to sRGB color space.
if exist('applycform','file')
   B = applycform(B,makecform('lab2srgb'));
else
   B = colorspace('RGB<-Lab',B);
end

% Close waitbar.
close(h);

clear all;
close all;
```

```
Ref=input('Reference noise-free image available? (y/n) ','s');

% Read reference, noise-free image
if (Ref=='y')
    InputType=0;
    while (InputType~=1 & InputType~=2 & InputType~=3 & InputType~=4)
     InputType=input('\nReference image format:\n1. "*.tif"\n2.
"*.sg2"\n3. "*.mat"\n4. "*.bin"\n>>  ');
    end
    imload;
    Xclean=SCAN;
    clear SCAN;
    [M,N]=size(Xclean);

    Choice=input('Choose 1 or 2:\n1. An arbitrary noise variance\n2. A
stored noisy image\n ');

    if(Choice==1)
       NoiseStd=input('Standard Deviation of added noise in each qadrature
channel: ');
       Xnoisy=Xclean+randn(M,N)*NoiseStd+sqrt(-1)*randn(M,N)*NoiseStd;
       Xnoisy=abs(Xnoisy);
       clean_image=input('Reference image name: ','s');
       image_name=strcat(clean_image,'_std',num2str(NoiseStd));
    else
       InputType=0;
       while (InputType~=1 & InputType~=2 & InputType~=3 & InputType~=4)
         InputType=input('\nNoisy image format:\n1. "*.tif"\n2.
"*.sg2"\n3. "*.mat"\n4. "*.bin"\n>>  ');
       end
       imload;
       Xnoisy=SCAN;
       clear SCAN;

       %Extract the image name from the filename
       image_name=filename(1:(length(filename)-4))
    end

else % No reference noise-free image image available

    % Read noisy image
    InputType=0;
    while (InputType~=1 & InputType~=2 & InputType~=3 & InputType~=4)
        InputType=input('\nNoisy image format:\n1. "*.tif"\n2. "*.sg2"\n3.
"*.mat"\n4. "*.bin"\n>>  ');
    end
    imload;
    Xnoisy=SCAN;
    clear SCAN;

    %Extract the image name from the filename
    image_name=filename(1:(length(filename)-4))
end
```

```
 [M,N]=size(Xnoisy);
 figure('units','normalized','position',[0.05 0.1 0.3 0.8]);
 subplot(2,1,1),imagesc(abs(Xnoisy));title('Input image');
 colormap(gray);

 Xsqmag=abs(Xnoisy).^2;


 %---------------------------------------------------------------
 %Parameters

 Thr_factor=2;
 Thr_factorINFO='s';
 while((Thr_factorINFO~='y')&(Thr_factorINFO~='n'))
    Thr_factorINFO=input('The threshold multiplication factor is set to
default value K=2. \nChoose other value (y/n)? ','s');
 end
 if(Thr_factorINFO=='y')
   Thr_factor=input('Multiplication factor for threshold (1-5): ');
 end

 WS=5;
 WS_INFO='s';
 while((WS_INFO~='y')&(WS_INFO~='n'))
   WS_INFO=input('Window size is set to default 5. \nChoose other value
(y/n)? ','s');
 end
 if(WS_INFO=='y')
   WS=1;
   while((WS~=3)&(WS~=5)&(WS~=7))
      WS=input('Window size (3, 5, 7): ');
   end
end
W=(WS-1)/2;


 %---------------------------------------------------------------

 %-------------------------------------------------------------------------
 %Linear rescaling  - for the sake of a faster computation in later steps
 R=1000/max(max(Xsqmag));
 Xsqmag=Xsqmag*R;
 %-------------------------------------------------------------------------


   [A1,HL1,LH1,HH1]=wt3det_spline(Xsqmag,0);
   [A2,HL2,LH2,HH2]=wt3det_spline(A1,1);
   [A3,HL3,LH3,HH3]=wt3det_spline(A2,2);
   [A4,HL4,LH4,HH4]=wt3det_spline(A3,3);
```

```
   sigma=thr_univ(HH1);
   SIGMA_HH=[1.00 0.22 0.09 0.04]*sigma;
   SIGMA_LH=[0.79 0.25 0.11 0.05]*sigma;

   sig_hh1=SIGMA_HH(1);
   sig_hh2=SIGMA_HH(2);
   sig_hh3=SIGMA_HH(3);
   sig_hh4=SIGMA_HH(4);

   sig_lh1=SIGMA_LH(1);
   sig_lh2=SIGMA_LH(2);
   sig_lh3=SIGMA_LH(3);
   sig_lh4=SIGMA_LH(4);

   sigma=sig_hh1  %This is an estimate of R*2*NoiseStd^2 because we are
processing the square magnitude!


%====================================
% Processing scale 2^3

Scale=3

[HL3p,M_hl3]=rem_noise_adapt(HL3,HL4,sig_lh3,Thr_factor,W);
[LH3p,M_lh3]=rem_noise_adapt(LH3,LH4,sig_lh3,Thr_factor,W);
[HH3p,M_hh3]=rem_noise_adapt(HH3,HH4,sig_hh3,Thr_factor,W);

%figure
%subplot(3,2,1),imagesc(HL3);
%subplot(3,2,3),imagesc(LH3);
%subplot(3,2,5),imagesc(HH3);
%colormap(gray);
%subplot(3,2,2),imagesc(HL3p);
%subplot(3,2,4),imagesc(LH3p);
%subplot(3,2,6),imagesc(HH3p);
%colormap(gray);
clear HL4 LH4 HH4 z;
%====================================
% Processing scale 2^2


[HL2p,M_hl2]=rem_noise_adapt(HL2,HL3,sig_lh2,Thr_factor,W);
[LH2p,M_lh2]=rem_noise_adapt(LH2,LH3,sig_lh2,Thr_factor,W);
[HH2p,M_hh2]=rem_noise_adapt(HH2,HH3,sig_hh2,Thr_factor,W);

Scale=2
%figure
%subplot(3,2,1),imagesc(HL2);
%subplot(3,2,3),imagesc(LH2);
%subplot(3,2,5),imagesc(HH2);
%colormap(gray);
```

```
%subplot(3,2,2),imagesc(HL2p);
%subplot(3,2,4),imagesc(LH2p);
%subplot(3,2,6),imagesc(HH2p);
%colormap(gray);

clear HL3 LH3 HH3;
%====================================
% Processing scale 2^1

Scale=1

[HL1p,M_hl1]=rem_noise_adapt(HL1,HL2p,sig_lh1,Thr_factor,W);
[LH1p,M_lh1]=rem_noise_adapt(LH1,LH2p,sig_lh1,Thr_factor,W);
[HH1p,M_hh1]=rem_noise_adapt(HH1,HH2p,sig_hh1,Thr_factor,W);
%figure
%subplot(3,2,1),imagesc(HL1);
%subplot(3,2,3),imagesc(LH1);
%subplot(3,2,5),imagesc(HH1);
%colormap(gray);
%subplot(3,2,2),imagesc(HL1p);
%subplot(3,2,4),imagesc(LH1p);
%subplot(3,2,6),imagesc(HH1p);
%colormap(gray);

clear LH2 HL2 HH2 LH1 HL1 HH1;

%========================================
% Reconstruction
%========================================

 %A2p=iwt3det_spline(A3-sigma,HL3p,LH3p,HH3p,2);
 A1p=iwt3det_spline(A2-sigma,HL2p,LH2p,HH2p,1);
 A0p=iwt3det_spline(A1p,HL1p,LH1p,HH1p,0);

 Xden=abs(A0p/R).^0.5;

figure('units','normalized','position',[0.05 0.1 0.3 0.8]);
subplot(2,1,1),imagesc(Xnoisy); title('Input image');
subplot(2,1,2),imagesc(Xden); title('Denoised image');
colormap(gray);

clear A3 A4
clear HL1p HL2p HL3p LH1p LH2p LH3p HH1p HH2p HH3p;



if(Ref=='y')
  Ps=std2(Xclean);
  Pn=std2(Xclean-Xnoisy);
  Prn=std2(Xclean-Xden);

  snr_input=20*log10(Ps/Pn)
  snr_res=20*log10(Ps/Prn)
```

```
  MSE_res=mean(mean((Xclean-Xden).^2));
  MSE_input=mean(mean((Xclean-Xnoisy).^2));
  PSNR_input=10*log10(255^2/MSE_input)
  PSNR_res=10*log10(255^2/MSE_res)
end


%==========================================
% Save result

Save_image_mat=input('Save the result as a mat file? y/n ','s')

if (Save_image_mat=='y');

Filename=strcat(image_name,'_linfit_T',num2str(Thr_factor),'_W',num2str(WS
),'.mat');
   X=Xden;
   WindowSize=WS;
   if(Ref=='y')

save(Filename,'X','snr_input','snr_res','PSNR_input','PSNR_res','WindowSiz
e','Thr_factor');
   else
      save(Filename,'X','WindowSize','Thr_factor');
   end
end

Save_image_tif=input('Save the result as tif image? y/n ','s')

if (Save_image_tif=='y');

Filename=strcat(image_name,'_linfit_T',num2str(Thr_factor),'_W',num2str(WS
),'.tif');
   X=Xden;
   X(X<0)=0;
   X(X>255)=255;
   imwrite(uint8(X),gray(256),Filename,'tif');
end
function Xden=genlik_MRI_fun(Xnoisy,Thr_factor,WindowSize);

 [M,N]=size(Xnoisy);
 Xsqmag=abs(Xnoisy).^2;
 W=round((WindowSize-1)/2);

 %--------------------------------------------------------------------------
 %Linear rescaling  - for the sake of a faster computation in later steps
 R=1000/max(max(Xsqmag));
 Xsqmag=Xsqmag*R;
 %--------------------------------------------------------------------------
```

```
   [A1,HL1,LH1,HH1]=wt3det_spline(Xsqmag,0);
   [A2,HL2,LH2,HH2]=wt3det_spline(A1,1);
   [A3,HL3,LH3,HH3]=wt3det_spline(A2,2);
   [A4,HL4,LH4,HH4]=wt3det_spline(A3,3);

   sigma=thr_univ(HH1);
   SIGMA_HH=[1.00 0.22 0.09 0.04]*sigma;
   SIGMA_LH=[0.79 0.25 0.11 0.05]*sigma;

   sig_hh1=SIGMA_HH(1);
   sig_hh2=SIGMA_HH(2);
   sig_hh3=SIGMA_HH(3);
   sig_hh4=SIGMA_HH(4);

   sig_lh1=SIGMA_LH(1);
   sig_lh2=SIGMA_LH(2);
   sig_lh3=SIGMA_LH(3);
   sig_lh4=SIGMA_LH(4);

   sigma=sig_hh1;  %This is an estimate of R*2*NoiseStd^2 because we are
processing the square magnitude!

%====================================
% Processing scale 2^3
[HL3p,M_hl3]=rem_noise_adapt(HL3,HL4,sig_lh3,Thr_factor,W);
[LH3p,M_lh3]=rem_noise_adapt(LH3,LH4,sig_lh3,Thr_factor,W);
[HH3p,M_hh3]=rem_noise_adapt(HH3,HH4,sig_hh3,Thr_factor,W);
clear HL4 LH4 HH4 z;
%====================================
% Processing scale 2^2
[HL2p,M_hl2]=rem_noise_adapt(HL2,HL3,sig_lh2,Thr_factor,W);
[LH2p,M_lh2]=rem_noise_adapt(LH2,LH3,sig_lh2,Thr_factor,W);
[HH2p,M_hh2]=rem_noise_adapt(HH2,HH3,sig_hh2,Thr_factor,W);
clear HL3 LH3 HH3;
%====================================
% Processing scale 2^1
[HL1p,M_hl1]=rem_noise_adapt(HL1,HL2p,sig_lh1,Thr_factor,W);
[LH1p,M_lh1]=rem_noise_adapt(LH1,LH2p,sig_lh1,Thr_factor,W);
[HH1p,M_hh1]=rem_noise_adapt(HH1,HH2p,sig_hh1,Thr_factor,W);
clear LH2 HL2 HH2 LH1 HL1 HH1;
%=======================================
% Reconstruction
%=======================================
 %A2p=iwt3det_spline(A3-sigma,HL3p,LH3p,HH3p,2);
 A1p=iwt3det_spline(A2-sigma,HL2p,LH2p,HH2p,1);
 A0p=iwt3det_spline(A1p,HL1p,LH1p,HH1p,0);

 Xden=abs(A0p/R).^0.5;
 clear A3 A4
 clear HL1p HL2p HL3p LH1p LH2p LH3p HH1p HH2p HH3p;
function [fima]=mixingsubband(fimau,fimao)

s = size(fimau);
```

```
p(1) = 2^(ceil(log2(s(1))));
p(2) = 2^(ceil(log2(s(2))));
p(3) = 2^(ceil(log2(s(3))));

pad1 = zeros(p(1),p(2),p(3));
pad2 = pad1;
pad1(1:s(1),1:s(2),1:s(3)) = fimau(:,:,:);
pad2(1:s(1),1:s(2),1:s(3)) = fimao(:,:,:);

[af, sf] = farras;
w1 = dwt3D(pad1,1,af);
w2 = dwt3D(pad2,1,af);

w1{1}{3} = w2{1}{3};
w1{1}{5} = w2{1}{5};
w1{1}{6} = w2{1}{6};
w1{1}{7} = w2{1}{7};

fima = idwt3D(w1,1,sf);
fima = fima(1:s(1),1:s(2),1:s(3));

% NAN checking
ind=find(isnan(fima(:)));
fima(ind)=fimau(ind);

% negative checking (only for rician noise mixing)
ind=find(fima<0);
fima(ind)=0;

% s = size(fimau);
%
% p(1) = 2^(ceil(log2(s(1))));
% p(2) = 2^(ceil(log2(s(2))));
% p(3) = 2^(ceil(log2(s(3))));
%
% pad1 = zeros(p(1),p(2),p(3));
% pad2=pad1;
% pad1(1:s(1),1:s(2),1:s(3)) = fimau(:,:,:);
% pad2(1:s(1),1:s(2),1:s(3)) = fimao(:,:,:);
%
% [af, sf] = farras;
% w1 = dwt3D(pad1,1,af);
% w2 = dwt3D(pad2,1,af);
%
%   w1{1}{1} = (w1{1}{1} + w2{1}{1})/2;
%   w1{1}{2} = (w1{1}{2} + w2{1}{2})/2;
%   w1{1}{3} = w2{1}{3};
%   w1{1}{4} = (w1{1}{4} + w2{1}{4})/2;
%   w1{1}{5} = w2{1}{5};
%   w1{1}{6} = w2{1}{6};
%   w1{1}{7} = w2{1}{7};
%
% fima = idwt3D(w1,1,sf);
```

```
% fima = fima(1:s(1),1:s(2),1:s(3));

% BFILTER2 Two dimensional bilateral filtering   final stage.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Pre-process input and select appropriate filter.
function B = bfilter2(A,w,sigma)

% Verify that the input image exists and is valid.
if ~exist('A','var') || isempty(A)
   error('Input image A is undefined or invalid.');
end
if ~isfloat(A) || ~sum([1,3] == size(A,3)) || ...
      min(A(:)) < 0 || max(A(:)) > 1
   error(['Input image A must be a double precision ',...
          'matrix of size NxMx1 or NxMx3 on the closed ',...
          'interval [0,1].']);
end

% Verify bilateral filter window size.
if ~exist('w','var') || isempty(w) || ...
      numel(w) ~= 1 || w < 1
   w = 5;
end
w = ceil(w);

% Verify bilateral filter standard deviations.
if ~exist('sigma','var') || isempty(sigma) || ...
      numel(sigma) ~= 2 || sigma(1) <= 0 || sigma(2) <= 0
   sigma = [3 0.1];
end

% Apply either grayscale or color bilateral filtering.
if size(A,3) == 1
   B = bfltGray(A,w,sigma(1),sigma(2));
else
   B = bfltColor(A,w,sigma(1),sigma(2));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Implements bilateral filtering for grayscale images.
function B = bfltGray(A,w,sigma_d,sigma_r)

% Pre-compute Gaussian distance weights.
[X,Y] = meshgrid(-w:w,-w:w);
G = exp(-(X.^2+Y.^2)/(2*sigma_d^2));

% Create waitbar.
h = waitbar(0,'Applying bilateral filter...');
set(h,'Name','Bilateral Filter Progress');
```

```
% Apply bilateral filter.
dim = size(A);
B = zeros(dim);
for i = 1:dim(1)
   for j = 1:dim(2)

         % Extract local region.
         iMin = max(i-w,1);
         iMax = min(i+w,dim(1));
         jMin = max(j-w,1);
         jMax = min(j+w,dim(2));
         I = A(iMin:iMax,jMin:jMax);

         % Compute Gaussian intensity weights.
         H = exp(-(I-A(i,j)).^2/(2*sigma_r^2));

         % Calculate bilateral filter response.
         F = H.*G((iMin:iMax)-i+w+1,(jMin:jMax)-j+w+1);
         B(i,j) = sum(F(:).*I(:))/sum(F(:));

   end
   waitbar(i/dim(1));
end

% Close waitbar.
close(h);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Implements bilateral filter for color images.
function B = bfltColor(A,w,sigma_d,sigma_r)

% Convert input sRGB image to CIELab color space.
if exist('applycform','file')
   A = applycform(A,makecform('srgb2lab'));
else
   A = colorspace('Lab<-RGB',A);
end

% Pre-compute Gaussian domain weights.
[X,Y] = meshgrid(-w:w,-w:w);
G = exp(-(X.^2+Y.^2)/(2*sigma_d^2));

% Rescale range variance (using maximum luminance).
sigma_r = 100*sigma_r;

% Create waitbar.
h = waitbar(0,'Applying bilateral filter...');
set(h,'Name','Bilateral Filter Progress');

% Apply bilateral filter.
dim = size(A);
```

```
B = zeros(dim);
for i = 1:dim(1)
   for j = 1:dim(2)

         % Extract local region.
         iMin = max(i-w,1);
         iMax = min(i+w,dim(1));
         jMin = max(j-w,1);
         jMax = min(j+w,dim(2));
         I = A(iMin:iMax,jMin:jMax,:);

         % Compute Gaussian range weights.
         dL = I(:,:,1)-A(i,j,1);
         da = I(:,:,2)-A(i,j,2);
         db = I(:,:,3)-A(i,j,3);
         H = exp(-(dL.^2+da.^2+db.^2)/(2*sigma_r^2));

         % Calculate bilateral filter response.
         F = H.*G((iMin:iMax)-i+w+1,(jMin:jMax)-j+w+1);
         norm_F = sum(F(:));
         B(i,j,1) = sum(sum(F.*I(:,:,1)))/norm_F;
         B(i,j,2) = sum(sum(F.*I(:,:,2)))/norm_F;
         B(i,j,3) = sum(sum(F.*I(:,:,3)))/norm_F;

   end
   waitbar(i/dim(1));
end

% Convert filtered image back to sRGB color space.
if exist('applycform','file')
   B = applycform(B,makecform('lab2srgb'));
else
   B = colorspace('RGB<-Lab',B);
end

% Close waitbar.
close(h);
% BFILTER2 Two dimensional bilateral filtering   final stage.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Pre-process input and select appropriate filter.
function B = bfilter2(A,w,sigma)

% Verify that the input image exists and is valid.
if ~exist('A','var') || isempty(A)
   error('Input image A is undefined or invalid.');
end
if ~isfloat(A) || ~sum([1,3] == size(A,3)) || ...
      min(A(:)) < 0 || max(A(:)) > 1
   error(['Input image A must be a double precision ',...
          'matrix of size NxMx1 or NxMx3 on the closed ',...
          'interval [0,1].']);
end
```

```
% Verify bilateral filter window size.
if ~exist('w','var') || isempty(w) || ...
      numel(w) ~= 1 || w < 1
   w = 5;
end
w = ceil(w);

% Verify bilateral filter standard deviations.
if ~exist('sigma','var') || isempty(sigma) || ...
      numel(sigma) ~= 2 || sigma(1) <= 0 || sigma(2) <= 0
   sigma = [3 0.1];
end

% Apply either grayscale or color bilateral filtering.
if size(A,3) == 1
   B = bfltGray(A,w,sigma(1),sigma(2));
else
   B = bfltColor(A,w,sigma(1),sigma(2));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Implements bilateral filtering for grayscale images.
function B = bfltGray(A,w,sigma_d,sigma_r)

% Pre-compute Gaussian distance weights.
[X,Y] = meshgrid(-w:w,-w:w);
G = exp(-(X.^2+Y.^2)/(2*sigma_d^2));

% Create waitbar.
h = waitbar(0,'Applying bilateral filter...');
set(h,'Name','Bilateral Filter Progress');

% Apply bilateral filter.
dim = size(A);
B = zeros(dim);
for i = 1:dim(1)
   for j = 1:dim(2)

         % Extract local region.
         iMin = max(i-w,1);
         iMax = min(i+w,dim(1));
         jMin = max(j-w,1);
         jMax = min(j+w,dim(2));
         I = A(iMin:iMax,jMin:jMax);

         % Compute Gaussian intensity weights.
         H = exp(-(I-A(i,j)).^2/(2*sigma_r^2));
         % Calculate bilateral filter response.
         F = H.*G((iMin:iMax)-i+w+1,(jMin:jMax)-j+w+1);
         B(i,j) = sum(F(:).*I(:))/sum(F(:));

   end
```

```
   waitbar(i/dim(1));
end

% Close waitbar.
close(h);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Implements bilateral filter for color images.
function B = bfltColor(A,w,sigma_d,sigma_r)

% Convert input sRGB image to CIELab color space.
if exist('applycform','file')
   A = applycform(A,makecform('srgb2lab'));
else
   A = colorspace('Lab<-RGB',A);
end

% Pre-compute Gaussian domain weights.
[X,Y] = meshgrid(-w:w,-w:w);
G = exp(-(X.^2+Y.^2)/(2*sigma_d^2));

% Rescale range variance (using maximum luminance).
sigma_r = 100*sigma_r;

% Create waitbar.
h = waitbar(0,'Applying bilateral filter...');
set(h,'Name','Bilateral Filter Progress');

% Apply bilateral filter.
dim = size(A);
B = zeros(dim);
for i = 1:dim(1)
   for j = 1:dim(2)

         % Extract local region.
         iMin = max(i-w,1);
         iMax = min(i+w,dim(1));
         jMin = max(j-w,1);
         jMax = min(j+w,dim(2));
         I = A(iMin:iMax,jMin:jMax,:);

         % Compute Gaussian range weights.
         dL = I(:,:,1)-A(i,j,1);
         da = I(:,:,2)-A(i,j,2);
         db = I(:,:,3)-A(i,j,3);
         H = exp(-(dL.^2+da.^2+db.^2)/(2*sigma_r^2));

         % Calculate bilateral filter response.
         F = H.*G((iMin:iMax)-i+w+1,(jMin:jMax)-j+w+1);
         norm_F = sum(F(:));
         B(i,j,1) = sum(sum(F.*I(:,:,1)))/norm_F;
         B(i,j,2) = sum(sum(F.*I(:,:,2)))/norm_F;
         B(i,j,3) = sum(sum(F.*I(:,:,3)))/norm_F;
   End
```

```
   waitbar(i/dim(1));
end

% Convert filtered image back to sRGB color space.
if exist('applycform','file')
   B = applycform(B,makecform('lab2srgb'));
else
   B = colorspace('RGB<-Lab',B);
end

% Close waitbar.
close(h);
```