

2005

## Partition-based filters for image restoration

Yong Lin  
*University of Dayton*

Follow this and additional works at: [https://ecommons.udayton.edu/graduate\\_theses](https://ecommons.udayton.edu/graduate_theses)

---

### Recommended Citation

Lin, Yong, "Partition-based filters for image restoration" (2005). *Graduate Theses and Dissertations*. 3814.  
[https://ecommons.udayton.edu/graduate\\_theses/3814](https://ecommons.udayton.edu/graduate_theses/3814)

This Dissertation is brought to you for free and open access by the Theses and Dissertations at eCommons. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of eCommons. For more information, please contact [mschlange1@udayton.edu](mailto:mschlange1@udayton.edu), [ecommons@udayton.edu](mailto:ecommons@udayton.edu).

PARTITION-BASED FILTERS FOR IMAGE  
RESTORATION

DISSERTATION

Submitted to

The School of Engineering of the  
UNIVERSITY OF DAYTON

In Partial Fulfillment of the Requirements for  
The Degree

Doctor of Philosophy in Electrical Engineering

by

Yong Lin

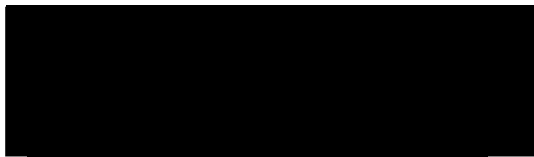
UNIVERSITY OF DAYTON

Dayton, Ohio

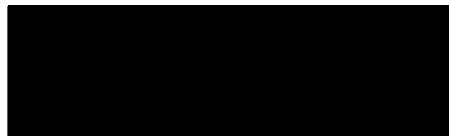
August, 2005

# PARTITION-BASED FILTERS FOR IMAGE RESTORATION

APPROVED BY:



Russell C. Hardie, Ph.D.  
Advisor and Committee Chairman  
Associate Professor  
Department of Electrical and  
Computer Engineering



Raúl Ordóñez, Ph.D.  
Committee Member  
Associate Professor  
Department of Electrical and  
Computer Engineering



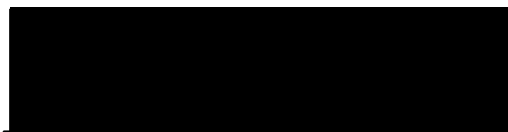
John S. Loomis, Ph.D.  
Committee Member  
Associate Professor  
Department of Electrical and  
Computer Engineering



Qin Sheng, Ph.D.  
Committee Member  
Associate Professor  
Department of Mathematics



Donald L. Moon, Ph.D.  
Associate Dean  
Graduate Engineering Programs &  
Research  
School of Engineering



Joseph E. Saliba, Ph.D., P.E.  
Dean, School of Engineering

© Copyright by

Yong Lin

2005

## ABSTRACT

Wiener filters are commonly used for image restoration. They are mean square based optimization filters, and are effective when the signal and noise are jointly Gaussian and stationary. The previously proposed partition-based weighted sum (PWS) filters combine vector quantization (VQ) and linear finite impulse response (FIR) Wiener filtering concepts. By partitioning the observation space and applying a tuned Wiener filter to each partition, the PWS filter is spatially adaptive and has been shown to perform well in noise reduction applications.

In this dissertation, subspace PWS (SPWS) filters are proposed, and their efficacy in image deconvolution and noise reduction applications is evaluated. In the SPWS filter, the observation vectors are projected into a subspace using principal component analysis (PCA), or other methods, prior to partitioning. This subspace projection can dramatically reduce the computational burden associated with partitioning, especially for large window sizes. In some cases, performance is also enhanced due to improved partitioning.

The Soft-PWS filters use information from all of the partition filters to process each observation vector. Soft-PWS filters have been shown to be capable of outperforming PWS filters in noise reduction applications. To date, however, the optimization of the Soft-PWS filters has received only limited attention. Prior work has focused on a stochastic-gradient based method that requires a large number of iterations, making

it computationally prohibitive in many applications. Furthermore, its convergence is highly sensitive to the step size and no reliable method of determining a suitable step size has been presented.

In this dissertation, a novel radial basis function interpretation of the Soft-PWS filters is described, and an efficient optimization procedure that we believe makes the Soft-PWS filters far more practical to implement is presented. To demonstrate the efficiency of the Soft-PWS filters with the new optimization procedure, we apply the filters to the problem of noise reduction. Simulation results show that, for the data used here, the Soft-PWS filter outperforms the standard PWS filter and Wiener filter under the mean squared error criterion.

To my parents Tai Lin and JinXiang Huang, my wife Tao He, and our son Astin.

## ACKNOWLEDGMENTS

I would like to thank a number of people for their help, support and love through the years in my Ph.D. program. Most importantly my parents – Tai Lin and JinXiang Huang, and my wife - Tao He. The list also contains my brother – Xiang Lin, my parents-in-law – ChuanTu He and XingJue Liang, my cousins – MinJie Wang and JingJie Wang, and many family members and friends. Without their support for so many years, I would not be able to finish this work.

Many thanks also go to my advisor - Dr. Russell C. Hardie. Dr. Hardie has been very patient, supportive and understanding. Without his guidance for so many years, I would not be able to finish the work. I would like to express my gratitude to my Committee - Dr. Qin Sheng, Dr. Raúl Ordóñez, and Dr. John S. Loomis, the DAGSI (The Dayton Area Graduate Studies Institute) scholarship and all those who gave me the ability to complete this work.

I also would like to express my gratitude to my friend – Mr. Ian Cavanagh who recommended me to the Ph.D. program.

## TABLE OF CONTENTS

	Page
Abstract . . . . .	iii
Acknowledgments . . . . .	vi
List of Figures . . . . .	x
List of Tables . . . . .	xv
 Chapters:	
1. Introduction . . . . .	1
1.1 Motivation and Contribution . . . . .	1
1.2 Outline of Dissertation . . . . .	4
2. Partition Weighted Sum Filter . . . . .	6
2.1 Partition Weighted Sum Filter . . . . .	7
2.1.1 PWS Filter Definition . . . . .	7
2.1.2 VQ Partitioning . . . . .	8
2.1.3 PWS Filter Optimization . . . . .	10
2.2 Soft Partition Weighted Sum Filter . . . . .	11
3. Subspace Partition Weighted Sum Filter . . . . .	14
3.1 Subspace PWS Filter . . . . .	14
3.2 Computational Complexity . . . . .	17
3.3 Summary . . . . .	18

4.	Interpretations and Optimization of Soft Partition Weighted Sum Filter	19
4.1	Radial Basis Function Interpretation of Soft-PWS Filter . . . . .	19
4.2	Optimization of Soft-PWS Filters . . . . .	20
4.2.1	Cost Function Definition . . . . .	20
4.2.2	Optimization With Respect to $\mathbf{w}$ . . . . .	21
4.2.3	Optimization With Respect to $\mathbf{c}$ and $\mathbf{s}$ . . . . .	22
4.3	Summary . . . . .	23
5.	Subspace Partition Weighted Sum Filter Experimental Results . . . . .	26
5.1	Image Deconvolution . . . . .	27
5.2	Image Noise Reduction . . . . .	33
5.2.1	Aerial Images . . . . .	39
5.2.2	Car Images . . . . .	43
5.3	Summary . . . . .	46
6.	Improved Optimization of Soft Partition Weighted Sum Filters Experimental Results . . . . .	48
6.1	Natural Image Noise reduction Results . . . . .	49
6.1.1	Training Process . . . . .	49
6.1.2	Aerial Test Images Results . . . . .	53
6.1.3	Car Test Image Results . . . . .	58
6.1.4	Tools Test Image Results . . . . .	62
6.1.5	Building Test Image Results . . . . .	65
6.2	Biomedical Images Results . . . . .	69
6.2.1	Two Dimensional CT Images Results . . . . .	70
6.2.2	Three Dimensional CT Images Results . . . . .	75
6.3	Summary . . . . .	79
7.	Conclusions . . . . .	80
Appendices:		
A.	Gradient with respect to codewords . . . . .	82
B.	Gradient with respect to radial basis parameters . . . . .	85

C. Matlab Codes Used for Subspace PWS Filter . . . . .	87
D. Matlab Codes Used for Soft-PWS Filter . . . . .	112
Bibliography . . . . .	137
Vita . . . . .	140

## LIST OF FIGURES

Figure	Page
2.1 PWS filter operation overview. . . . .	8
2.2 PWS filters voronoi regions/partitions. . . . .	9
2.3 Soft-PWS filters voronoi regions/partitions. . . . .	12
3.1 Center-PCA method: final $K=5$ samples selected from observation vector $\mathbf{x}(n)$ , $N=81$ . . . . .	16
3.2 Floating point operation count for the filters as a function of $M$ , with $N = 61$ , $L = 31$ , and $K = 5$ . . . . .	18
5.1 (a) The true training image ( $600 \times 600$ ); (b) the true test image ( $400$ $\times 400$ ). . . . .	27
5.2 Enlarged portion ( $100 \times 100$ ) of (a) desired test image ; (b) motion blurred image with Gaussian noise (standard deviation=1%, MSE=36.68); (c) Wiener output; and (d) PWS output ( $N = 1 \times 51$ , $M = 50$ ). . . .	29
5.3 Enlarged portion ( $100 \times 100$ ) of image deconvolution results: (a) SPWS (Center) output; (b) SPWS (PCA) output; and (c) SPWS (Center-PCA) output ( $N = 1 \times 51$ , $M = 50$ , $K = 5$ , and $L = 25$ ). . .	30
5.4 MSE analysis for the various filters applied to a motion blurred image with Gaussian noise ( $N = 1 \times 51$ , $M = 50$ , $K = 5$ , and $L = 25$ ). . . .	31
5.5 Training implementation time. . . . .	31
5.6 Filter implementation time. . . . .	32

5.7	Total implementation time. . . . .	33
5.8	Enlarged portion ( $100 \times 100$ ) of (a) test image with Gaussian noise (standard deviation=10%, non-blur, $MSE = 39.66$ ); (b) Wiener output ( $MSE = 22.48$ , $N = 7 \times 7$ ); (c) PWS output ( $MSE = 20.98$ ); and (d) SPWS (Center) output ( $MSE = 20.76$ , $N = 7 \times 7$ , $M = 50$ , and $L = 5 \times 5$ ). . . . .	34
5.9	Enlarged portion ( $100 \times 100$ ) of image noise reduction results: (a) SPWS (PCA) ( $MSE=21.01$ ); (b) SPWS (Center-PCA) ( $MSE = 20.76$ , $N = 7 \times 7$ , $M = 50$ , $K = 3 \times 3$ , and $L = 5 \times 5$ ). . . . .	36
5.10	MSE analysis for the various filters ( $N = 7 \times 7$ , $M = 50$ , $K = 3 \times 3$ , and $L = 5 \times 5$ ). . . . .	36
5.11	Image noise reduction training time. . . . .	37
5.12	Image noise reduction filter implementation time. . . . .	37
5.13	Image noise reduction total implementation time. . . . .	38
5.14	(a) The true test aerial image; and (b) the noisy test aerial image ( $512 \times 512$ ). . . . .	39
5.15	Enlarged portion ( $100 \times 100$ ) of (a) Image with Gaussian noise (standard deviation=10%, non-blur); (b) Wiener filter output; (c) PWS output; and (d) SPWS (Center) output ( $N = 7 \times 7$ , $M = 50$ , and $L = 5 \times 5$ ). . . . .	41
5.16	Enlarged portion ( $100 \times 100$ ) of image noise reduction results: (a) SPWS (PCA); and (b) SPWS (Center-PCA) ( $N = 7 \times 7$ , $M = 50$ , $K = 3 \times 3$ , and $L = 5 \times 5$ ). . . . .	42
5.17	MSE for the various filters ( $N = 7 \times 7$ , $M = 50$ , $K = 3 \times 3$ , and $L = 5 \times 5$ ). . . . .	42
5.18	(a) The true test car image; and (b) the noisy test car image ( $256 \times 256$ ). . . . .	43

5.19	Enlarged portion ( $100 \times 100$ ) of (a) Image with Gaussian noise (standard deviation=10%); Image noise reduction results: (b) Wiener filter output ( $N = 7 \times 7$ ); (c) PWS output; and (d) SPWS (Center) output ( $N = 7 \times 7$ , $M = 50$ , and $L = 5 \times 5$ ).	44
5.20	Enlarged portion ( $100 \times 100$ ) of image noise reduction results: (a) SPWS (PCA); and (b) SPWS (Center-PCA) ( $N = 7 \times 7$ , $M = 50$ , $K = 3 \times 3$ , and $L = 5 \times 5$ ).	45
5.21	MSE for the various filters ( $N = 7 \times 7$ , $M = 50$ , $K = 3 \times 3$ , and $L = 5 \times 5$ ).	46
6.1	The desired training image.	50
6.2	The $M = 50$ codebook.	50
6.3	The SPWS Center method's $M = 50$ codebook ( $N = 3 \times 3$ ).	51
6.4	The $M = 10$ codebook.	52
6.5	The SPWS Center method's $M = 10$ codebook ( $N = 3 \times 3$ ).	52
6.6	(a) The desired test aerial image; and (b) the noisy test aerial image ( $512 \times 512$ ).	54
6.7	Enlarged portions ( $200 \times 200$ ) of (a) the desired test aerial image; (b) the noisy test aerial image; (c) median filter output; and (d) moving average filter output ( $N = 5 \times 5$ ).	55
6.8	Enlarged portions ( $200 \times 200$ ) of (a) Wiener filter output (MSE = 220.90); (b) PWS filter output (MSE = 202.79); (c) Soft-PWS filter output with updated $\mathbf{w}$ only (MSE = 193.83); (d) Soft-PWS filter output with updated $\mathbf{w}$ , $\mathbf{c}$ and $\mathbf{s}$ (MSE = 192.69, $N = 5 \times 5$ , $M = 50$ ).	57
6.9	(a) The true test car image; and (c) the noisy test car image ( $256 \times 256$ ).	58
6.10	Enlarged center portion ( $150 \times 150$ ) of (a) true test car image; (b) noisy test car image; (c) median filter output; and (d) moving average filter output ( $N = 5 \times 5$ ).	59

6.11	Enlarged center portion ( $150 \times 150$ ) of (a) Wiener filter output, (b) PWS filter output; (c) Soft-PWS filter output with updated $\mathbf{w}$ only, and (d) Soft-PWS filter output with updated $\mathbf{w}$ , $\mathbf{c}$ and $\mathbf{s}$ ( $N = 5 \times 5$ , $M = 50$ ). . . . .	60
6.12	(a) The true test tool image; and (c) the noisy tool image ( $512 \times 512$ ). . . . .	62
6.13	Enlarged center portion ( $150 \times 150$ ) of (a) true test tool image; (b) noisy test tool image; (c) median filter output; and (d) moving average filter output ( $N = 5 \times 5$ ). . . . .	63
6.14	Enlarged center portion ( $150 \times 150$ ) of (a) Wiener filter output, (b) PWS filter output; (c) Soft-PWS filter output with updated $\mathbf{w}$ only, and (d) Soft-PWS filter output with updated $\mathbf{w}$ , $\mathbf{c}$ and $\mathbf{s}$ ( $N = 5 \times 5$ , $M = 50$ ). . . . .	64
6.15	(a) The true test building image, and (b) the noisy building image ( $512 \times 512$ ). . . . .	66
6.16	Enlarged center portion ( $150 \times 150$ ) of (a) true test building image; (b) noisy test building image; (c) median filter output; and (d) moving average filter output ( $N = 5 \times 5$ ). . . . .	67
6.17	Enlarged center portion ( $150 \times 150$ ) of (a) Wiener Filter output, (b) PWS filter output; (c) Soft-PWS filter output with updated $\mathbf{w}$ only, and (d) Soft-PWS filter output with updated $\mathbf{w}$ , $\mathbf{c}$ and $\mathbf{s}$ ( $N = 5 \times 5$ , $M = 50$ ). . . . .	68
6.18	The true training CT image from the first patient CT images. . . . .	71
6.19	(a) The desired test CT image from the second patient; and (b) the test CT image from the third patient ( $512 \times 512$ ). . . . .	72
6.20	Enlarged center portions ( $100 \times 100$ ) of (a) test CT image; (b) Wiener Filter output; (c) median filter output; and (d) moving average filter output ( $N = 5 \times 5$ ). . . . .	73
6.21	Enlarged center portions ( $100 \times 100$ ) of (a) 2D PWS filter output; and (b) 2D Soft-PWS filter output with updated $\mathbf{w}$ , $\mathbf{c}$ and $\mathbf{s}$ ( $N = 5 \times 5$ , $M = 50$ ). . . . .	74

6.22 Enlarged center portion ( $100 \times 100$ ) of (a) 3D Wiener filter output, (b) 3D PWS filter output, (c) 3D Soft-PWS filter output with updated $\mathbf{w}$ , $\mathbf{c}$ and $\mathbf{s}$ , and (d) 2D Soft-PWS filter output with updated $\mathbf{w}$ , $\mathbf{c}$ and $\mathbf{s}$ ( $N = 5 \times 5 \times 3$ , $L = 5 \times 5$ , $M = 50$ ). . . . .	78
---	----

## LIST OF TABLES

Table	Page
3.1 Computational complexity analysis. . . . .	17
4.1 Optimization overview using a cyclic coordinate descent method. . . .	21
4.2 Quasi-Newton Optimization Method Overview. . . . .	24
4.3 Steepest Descent Optimization Overview. . . . .	25
6.1 MSE Results for Training Process. . . . .	53
6.2 MSE Results for Test Aerial Image. . . . .	56
6.3 MSE Results for Car Test Images. . . . .	61
6.4 MSE Results for Tool Test Images. . . . .	65
6.5 MSE Results for Building Test Images. . . . .	66
6.6 MSE Results for 2D images. . . . .	71
6.7 MSE Results for 3D Images. . . . .	75
6.8 MSE Improvements of 3D Images from 2D Images. . . . .	76

# CHAPTER 1

## INTRODUCTION

### 1.1 Motivation and Contribution

For signal and image restoration, the most commonly used technique is the Wiener filter. The Wiener filter is optimum, in a mean-squared error (MSE) sense, if the signal and noise are jointly Gaussian and stationary.<sup>1</sup> It may, however, be suboptimal for a spatially-varying degradation function or with non-stationary signals and noise. Barner, Sarhan, and Hardie,<sup>2</sup> proposed the partition-based weighted sum (PWS) filter, which combines vector quantization (VQ)<sup>3</sup> and Wiener filter concepts. Alternative partitioning and optimization schemes for the PWS filter have been explored by Shao, Barner and Hardie in<sup>4-7</sup> and applications of the PWS to super-resolution and demosaicing have been studied in.<sup>7</sup> A blind deconvolution method<sup>8-12</sup> using VQ partitioning was recently proposed by Nagasaki and Katsaggelos.<sup>13</sup> More broadly, the concept of partitioning the observation space and performing decision directed filtering can be found in many prior works.<sup>14-19</sup>

The PWS filter uses a moving window operation (window size  $N$ ). The observation vector at each position in the image is classified into one of the  $M$  partitions. Associated with each partition is a finite impulse response (FIR) Wiener filter that is “tuned” for data falling into that partition. After an observation vector is classified,

the corresponding Wiener filter is applied. The standard Wiener filter can be considered a special case in which a PWS filter has only one partition ( $M = 1$ ). Previous work <sup>2</sup> has demonstrated the effectiveness of the PWS filter in an image denoising application using relatively small observation windows. In many applications, large span restoration filters are desirable as they tend to outperform smaller window filters. However, using large window sizes dramatically increases the computational cost of the VQ partitioning and VQ codebook generation to a point where it may become impractical.

In this dissertation, we introduce subspace PWS (SPWS) filters and evaluate its efficacy in image deconvolution and noise reduction applications. With the SPWS filter, we project the observation vectors into a subspace using principal component analysis (PCA), or other linear transformation. VQ based on the Linde-Buzo-Grey (LBG) algorithm<sup>3</sup> is used in the subspace for partitioning. This subspace projection greatly reduces the computational burden associated with the large window size PWS filters. In some cases, it also improves performance due to better partitioning of the observation space.

The need to determine both the optimum partitions (defined by the VQ code-words) and filter weights makes the global optimization of the PWS very challenging. In order to make the problem more tractable, the approach in<sup>2</sup> was to decouple the optimization of the partitioning and filter weights. First, the VQ partitions were designed using the LBG algorithm<sup>3,20</sup> to minimize the mean squared reconstruction error from a training image. Next, given the partitions are fixed, the Wiener weights for a given partition are determined using only data falling into that particular partition. This creates a very practical and effective training procedure. However, since

the optimization of the partitions and filter weights are decoupled, the result may be suboptimal. In an effort to obtain improved performance, a joint optimization of the partitions and filter weights using a genetic algorithm was proposed in.<sup>5</sup> However, this approach is very computationally demanding and only a small gain in performance over the decoupled optimization was observed.

The main barrier to employing a straightforward and efficient gradient-based joint optimization procedure is the fact that the filter error is not differentiable with respect to the VQ codewords. To circumvent this problem, a class of soft partition weighted sum (Soft-PWS) filters was proposed.<sup>6</sup> Here, the observation vector is processed with each partition filter and these estimates are combined in a weighted sum. As the observation vector gets closer to a given codeword (partition center), the weight for the corresponding partition filter output increases and the contribution from the others decreases. The weights vary continuously based on distance and do not go to zero. This soft partitioning approach has the advantage of yielding a filter with an MSE that is differentiable with respect to the codewords and the filter weights. A stochastic gradient optimization procedure for the Soft-PWS filters was developed in.<sup>6</sup> However, the procedure requires a very large number of iterations, making it impractical in many applications. Furthermore, as with any stochastic procedure, it is very sensitive to the choice of step size and no systematic procedure for identifying a suitable step size is provided in.<sup>6</sup>

In this dissertation, we present a novel interpretation of the Soft-PWS filters using radial basis functions.<sup>21,22</sup> From this new perspective, the Soft-PWS filter applies a single weight vector to each observation vector. This weight vector is the result

of interpolating from the set of fixed partition weight vectors using a Gaussian radial basis function. We believe this interpretation provides additional insight into the nature of the Soft-PWS filters. More significantly, we develop an efficient joint MSE optimization procedure for the weights, VQ codewords, and Gaussian radial basis function parameters. This procedure uses cyclic coordinate descent<sup>23-25</sup> where the weights are optimized with the other parameters fixed. Next the codewords are optimized with the weights and other parameters fixed. Finally, the Gaussian parameters are optimized with the other parameters fixed. This process continues cyclically until a halting criterion is met. For the optimization of the partition weights, we derive a closed form solution. The optimization of the VQ codewords and radial basis function parameters is achieved with a compound optimization strategy that includes the Quasi-Newton optimization method.<sup>22,23,25-29</sup> The overall optimization method proposed is significantly more efficient and reliable than the previously proposed stochastic gradient method.

To demonstrate the efficacy of the soft partition PWS filters with the new optimization method proposed, we present a number of image restoration experimental results. These results focus on the restoration of images corrupted with additive Gaussian noise.<sup>30-36</sup> We present quantitative results that show that the Soft-PWS filter, with the new optimization technique, outperforms the Wiener and PWS filter. We also present several images for subjective evaluation.

## 1.2 Outline of Dissertation

The remainder of this dissertation is organized as follows. In Chapter 2, the PWS and Soft-PWS filters are defined and the optimizations for PWS filter is discussed.

The proposed subspace PWS filters are defined in Chapter 3. An analysis of SPWS filters' computational complexity is also presented in Chapter 3. In Chapter 4, the novel interpretation of Soft-PWS filters via radial basis functions is proposed. The proposed numerical solution of the Soft-PWS optimization procedures are also given in Chapter 4. Chapters 5 and 6 are devoted to experimental results for proposed SPWS filters and Soft-PWS numerical solutions. Experimental results of the proposed SPWS filters with the Wiener and PWS filters in image deconvolution and noise reduction applications are provided in Chapter 5. Comparisons of proposed Soft-PWS numerical solutions with Wiener filter, the PWS filter and other filters are given in Chapter 6. 2D and 3D biomedical image results <sup>37,38</sup> are also presented in Chapter 6. Finally, conclusions and the possible future work are summarized in Chapter 7. The gradient with respect to codewords and the gradient with respect to radial basis parameters used in Chapter 4 are derived in Appendix A and B, respectively. The Matlab source codes used for generate the figures in Chapter 5 and 6 are presented in Appendix C and D, respectively.

## CHAPTER 2

### PARTITION WEIGHTED SUM FILTER

The well-known Wiener filter is optimum, in a MSE sense, if the signal and noise are jointly Gaussian and stationary. It may, however, be suboptimal for a spatially-varying degradation function or with non-stationary signals and noise. Barner, Sarhan, and Hardie,<sup>2</sup> proposed the PWS filter, which combines VQ<sup>3</sup> and Wiener filter concepts. Alternative partitioning and optimization schemes for the PWS have been explored and applications of the PWS to superresolution and demosaicing have been studied in by Shao, Barner and Hardie in.<sup>7</sup> To acquire a better optimized solution, Shao and Barner<sup>4,6</sup> considered a modified PWS filter – soft partition weighted sum (Soft-PWS) filter. The Soft-PWS filter utilizes a differentiable Gaussian-based partition function instead of the step function in the original PWS filters. In this chapter, we define the aforementioned PWS and Soft-PWS filters based on descriptions in .<sup>2,4,6</sup> The remainder of this chapter is organized as follows: Section 2.1 presents filter definitions for the PWS filters. The optimization of the PWS filter is also discussed in Section 2.1, and Soft-PWS filters are defined in Section 2.2.

## 2.1 Partition Weighted Sum Filter

We start with the definitions and notations used. Let  $\{d(\mathbf{n})\}$   $\{x(\mathbf{n})\}$  be the  $P$ -dimensional discrete data sequences representing desired and observed signal data, where the index  $\mathbf{n} = [n_1, n_2, \dots, n_P]^T$  ( $P = 2$  in case of 2D images). The PWS filters use moving window operations. At each location  $\mathbf{n}$ , the  $N$  samples of the observation window can be expressed as a vector

$$\mathbf{x}(\mathbf{n}) = [x_1(\mathbf{n}), x_2(\mathbf{n}), \dots, x_N(\mathbf{n})]^T. \quad (2.1)$$

For simplicity, the index  $\mathbf{n}$  will be assumed and not be written explicitly.

### 2.1.1 PWS Filter Definition

The PWS filter was proposed by Barner, Sarhan and Hardie<sup>2</sup> in 1999 to perform better than Wiener filter operations when the signal and noise are not jointly Gaussian. It uses a moving window operation and forms a nonlinear filter by using the partition information of the observation window samples, and applying associated Wiener filter weights to overcome the limitations of a linear Wiener filter. Deriving from Ref.,<sup>2</sup> the output of a PWS filter can be expressed as

$$F_{PWS}(\mathbf{x}) = \mathbf{w}_{p(\mathbf{x})}^T \mathbf{x}, \quad (2.2)$$

where  $\mathbf{w}_i = [w_{i,1}, w_{i,2}, \dots, w_{i,N}]^T$  is a filter weight vector of the partitions  $i$  for  $i = 1, 2, \dots, M$ . There are total  $M$  partitions.

Figure 2.1 shows an operation overview for the PWS filters. The filter first determines the partition index  $p(\mathbf{x})$ , and then forms the filter output by taking the inner product of  $\mathbf{x}$  and the corresponding weight vector  $\mathbf{w}_{p(\mathbf{x})}$ .

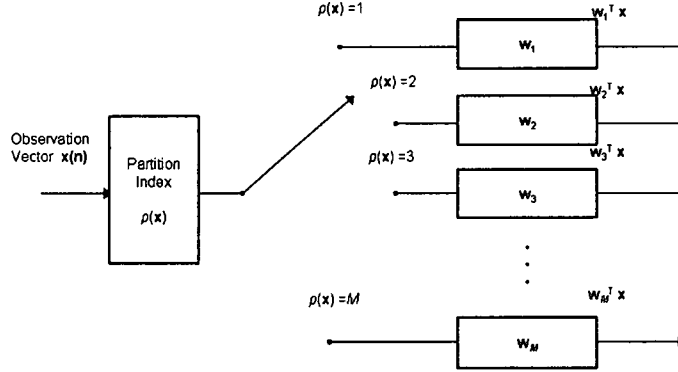


Figure 2.1: PWS filter operation overview.

### 2.1.2 VQ Partitioning

The encoder or partition function,  $p(\cdot) : R^N \mapsto \{1, 2, \dots, M\}$ , generates the partition index needed, and is given by

$$p(\mathbf{x}) = \arg \min_i \|\mathbf{x} - \mathbf{c}_i\|^2, \quad (2.3)$$

where the vector  $\mathbf{c}_i = [c_{i,1}, c_{i,2}, \dots, c_{i,N}]^T$  is the codeword for partition  $i$  from the codebook  $\mathbf{c} = \{\mathbf{c}_i, i = 1, \dots, M\}$ . The LBG algorithm<sup>3</sup> was used to generate the codebook from a training image in Ref.<sup>2</sup>

Figure 2.2 shows an voronoi region example of PWS filters. The filter determines the partition index  $p(\mathbf{x})$  by the distance to each partition. At the sample showing in Figure 2.2, the partition index  $p(\mathbf{x})$  is determined as 8 since the voronoi region 8 is the closest region.

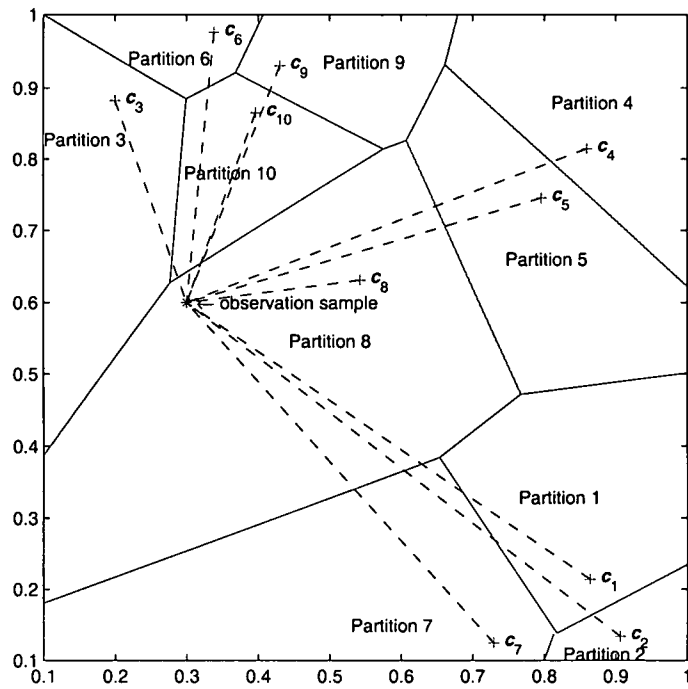


Figure 2.2: PWS filters voronoi regions/partitions.

### 2.1.3 PWS Filter Optimization

Since the performance of the PWS filtering is determined by the observation space partitions and the setting of each partition filtering operations jointly, a joint optimization is desired. However, the joint optimization of these components results in a set of nonlinearly coupled equations. A two-stage sub-optimal decoupled optimization approach was taken in.<sup>2</sup> That is, first minimizing the partitioning (quantization) error using the LBG algorithm, and then optimizing a filtering operation within each partitions.

With this methodology, we begin to rewrite the output of PWS filter given by Equ. (2.2) as follows:

$$F_{PWS}(\mathbf{x}) = \sum_{i=1}^M \mathbf{w}_i^T \mathbf{x} I_{PWS}, \quad (2.4)$$

where  $I_{PWS}(\cdot)$  is an event function.  $I_{PWS}(\cdot)$  is given by

$$I_{PWS} = \begin{cases} 1 & \text{if event is true;} \\ 0 & \text{if event is false.} \end{cases} \quad (2.5)$$

The cost function using the estimate MSE can be written as:

$$\begin{aligned} J &= E\{(d - F_{PWS})^2\} \\ &= E\{d^2 - 2dF_{PWS} + F_{PWS}^2\} \\ &= E\{d^2\} - 2E\{dF_{PWS}\} + E\{F_{PWS}^2\} \\ &= \sigma_d^2 - 2E\{dF_{PWS}\} + E\{F_{PWS}^2\} \\ &= \sigma_d^2 - 2E\left\{d \sum_{i=1}^M \mathbf{w}_i^T \mathbf{x} I_{PWS}\right\} + E\left\{\left(\sum_{i=1}^M \mathbf{w}_i^T \mathbf{x} I_{PWS}\right)^2\right\} \\ &= \sigma_d^2 - 2E\left\{\sum_{i=1}^M \mathbf{w}_i^T (d\mathbf{x}) I_{PWS}\right\} + E\left\{\sum_{i=1}^M \mathbf{w}_i^T \mathbf{x} \mathbf{x}^T \mathbf{w}_i I_{PWS}\right\}, \end{aligned} \quad (2.6)$$

where  $\sigma_d^2 = E\{d^2\}$ .

Since the auto correlation matrix  $\mathbf{R}_i$  of the  $i$ th partition,  $\Omega_i$ , can be defined as  $\mathbf{R}_i = E[\mathbf{x}\mathbf{x}^T \mid \mathbf{x} \in \Omega_i]$  and the cross correlation vector for  $\Omega_i$  can be defined as  $\mathbf{p}_i = E[d\mathbf{x}^T \mid \mathbf{x} \in \Omega_i]$ , the above cost function Equ. (2.6) can be written as:

$$J = \sigma_d^2 - 2E\left\{\sum_{i=1}^M \mathbf{w}_i^T \mathbf{p}_i I_{PWS}\right\} + E\left\{\sum_{i=1}^M \mathbf{w}_i^T \mathbf{R}_i \mathbf{w}_i I_{PWS}\right\}. \quad (2.7)$$

The optimized weights,  $\mathbf{w}_i^*$ , can be obtained by setting the gradient of the above cost function Equ. (2.7) with respect to each weight vector  $\mathbf{w}_i$ , to zero. That is,

$$\mathbf{w}_i^* = \mathbf{R}_i^{-1} \mathbf{p}_i. \quad (2.8)$$

where  $i = 1, 2, \dots, M$ ,  $\mathbf{R}_i = E[\mathbf{x}\mathbf{x}^T \mid \mathbf{x} \in \Omega_i]$  and  $\mathbf{p}_i = E[d\mathbf{x}^T \mid \mathbf{x} \in \Omega_i]$ .

## 2.2 Soft Partition Weighted Sum Filter

A modified partition weighted sum filter – soft partition weighted sum filter – was proposed by Shao.<sup>4</sup> The traditional partition function of the PWS filters,  $p(\cdot)$  or  $I_{PWS}(\cdot)$ , can be referred to as a hard partition function since it was a distance-based step function and not differentiable. In order to acquire better optimized results, a differentiable distance-based soft partition function – Gaussian membership function  $\hat{p}(\cdot)$  was introduced to replace the hard partition function. From Ref.,<sup>7</sup> the output of the Soft-PWS filter can then be expressed as

$$F_{Soft-PWS}(\mathbf{x}) = \sum_{i=1}^M \mathbf{w}_i^T \mathbf{x} \hat{p}_i(\mathbf{x}), \quad (2.9)$$

where

$$\hat{p}_i(\mathbf{x}) = e^{-\frac{\|\mathbf{x} - \mathbf{c}_i\|^2}{s_i}} \quad (2.10)$$

is the soft partition function which uses distance-based Gaussian membership functions, and  $s_i$  is the variance of the codeword,  $\mathbf{c}_i : \mathbf{c}_i \in \mathcal{C}$ .

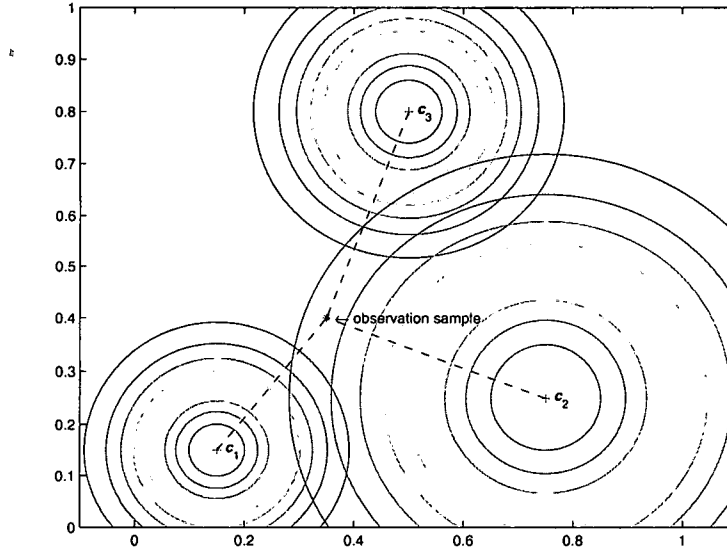


Figure 2.3: Soft-PWS filters voronoi regions/partitions.

In order to ensure the sum of all membership functions is equal to one, a normalization is applied. The normalized soft partition function can be expressed as,

$$\hat{p}_i(\mathbf{x}) = \frac{e^{-\frac{\|\mathbf{x}-\mathbf{c}_i\|^2}{s_i}}}{\sum_{k=1}^M e^{-\frac{\|\mathbf{x}-\mathbf{c}_k\|^2}{s_k}}} \quad (2.11)$$

Soft-PWS filters use same partitions/voronoi regions from the PWS filters.

Fig. 2.3 shows how the Soft-PWS filter differs from a PWS filter. Instead of partitioning an observation  $\mathbf{x}$  into one of partitions, the new soft partition function actually performs a weighted sum of all partitions.

In,<sup>4</sup> stochastic gradient-based optimization approaches were adopted to perform Soft-PWS filter optimizations. However, the convergence of the stochastic gradient algorithms is not guaranteed, and also the convergence fully depends on parameters selected. The parameter selection process is difficult and the optimization process

can be time-consuming. In Chapter 4, the optimization of the Soft-PWS filter will be studied further, and better numerical solutions are proposed.

## CHAPTER 3

### SUBSPACE PARTITION WEIGHTED SUM FILTER

In Chapter 2, we define the PWS filter. In many cases such as image deconvolution and 3D image processing, a larger observation window is required. However, the larger observation window dramatically increases the computational cost. In some cases, it may become impractical to implement the PWS filters. In this chapter, we propose subspace PWS filters which are targeted to reduce the computational complexity with little or no loss of performance. Three types of SPWS filters are proposed: center SPWS filters, the PCA SPWS filters and the combined center SPWS filter and PCA SPWS method – the Center-PCA SPWS filters.

The remainder of this chapter is organized as follows: Section 3.1 presents SPWS filter definitions. Section 3.2 presents the analysis of the computation complexities for each SPWS filter. Finally, a summary is given in Section 3.3.

#### 3.1 Subspace PWS Filter

To reduce the computational complexity of partitioning the  $R^N$  observation space, we propose projecting the observation vectors into an  $R^K$  subspace ( $K < N$ ) through the linear transformation

$$\tilde{\mathbf{x}} = \mathbf{A}\mathbf{x}, \tag{3.1}$$

where  $\mathbf{A}$  is a  $K \times N$  matrix. Thus the output of SPWS filter can be rewritten as

$$F_{SPWS}(\mathbf{x}) = \mathbf{w}_{\tilde{p}(\bar{\mathbf{x}})}^T \mathbf{x}, \quad (3.2)$$

where  $\tilde{p}(\cdot) : R^K \mapsto \{1, 2, \dots, M\}$ . We now consider and describe three choices for  $\mathbf{A}$ .

The first case is where  $\mathbf{A}$  is based on PCA using the Karhunen-Loeve (KL) transform.<sup>33</sup> The PCA algorithm, or KL transform, is an elegant and efficient way to reduce data dimensionality. In this case, the rows of  $\mathbf{A}$  are made up of the  $K$  eigenvectors with the largest eigen values of the covariance matrix for  $\mathbf{x}(\mathbf{n})$ . The bulk of the variation in the data can often be captured in a subspace using this method, as shown in Chapter 5. We refer to this as the SPWS (PCA) method. A simpler method selects  $K$  observation sample locations to form the subspace. In that case,  $\mathbf{A}$  contains one one in each row located in a unique column (the other entries are zero). This serves as a simple selection function. In Chapter 5, we consider the case where the most central  $K$  samples are selected, and we refer to this as the SPWS (Center) method. Finally, we consider combining the PCA and center selection method to form the SPWS (Center-PCA) method. In this case,  $\mathbf{A} = \mathbf{A}_{PCA} \mathbf{A}_S$  where  $\mathbf{A}_S$  is an  $L \times N$  matrix that selects the  $L$  most central samples and  $\mathbf{A}_{PCA}$  is a  $K \times L$  PCA subspace transformation matrix where  $K < L < N$ . Note with the Center-PCA method, we reduce the dimensionality from  $N$  to  $L$  through center selection. We then further reduce the dimensionality from  $L$  to  $K$  using PCA. The emphasis on the center may be justified by the observation that image restoration filters tend to have the largest magnitude impulse response values near the center. All of these methods are designed to allow one to use a large filter window (i.e., large  $N$ ), which may be desirable in many applications, without the very high cost of partitioning this high dimensional space.

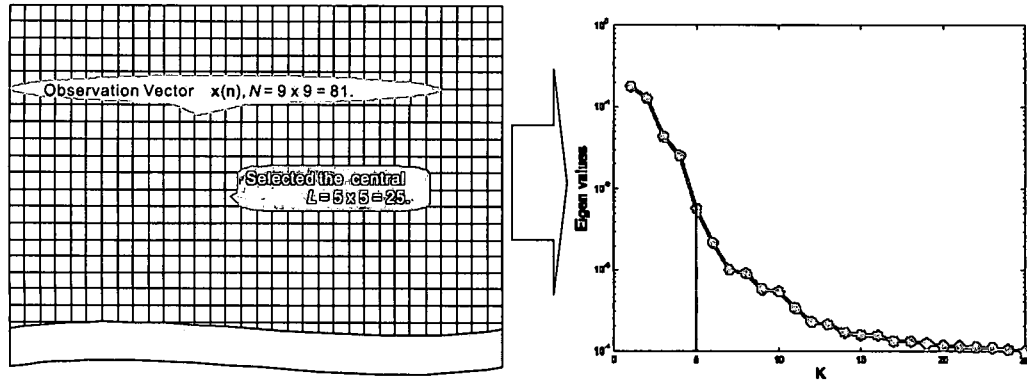


Figure 3.1: Center-PCA method: final  $K=5$  samples selected from observation vector  $x(n)$ ,  $N=81$ .

Fig. 3.1 shows an example of how final  $K=5$  samples selected from observation vector  $x(n)$ ,  $N=81$  by Center-PCA method. For the PCA method, setting  $K = 5$  preserves approximately 93% of the signal variance from the full observation space. For the Center-PCA method, setting  $K = 5$  and  $L = 25$  preserves approximately 98%. Since the PCA method only keeps  $K$  components with large eigenvalues, it is important to know the quality of the approximation. Principally,  $K$  can be any value as far as  $K < N$ . The summation of total eigen values is 0.9651, the summation of top 5 eigen values is 0.8981, which results in an approximation performance equal to 93%.

The selection of the parameters  $N$ ,  $M$ ,  $L$  and  $K$  is highly much dependent on the application at hand. We have observed that larger window sizes,  $N$ , generally outperform smaller window sizes, provided sufficient statistically representative data are available for training. In our applications, we have obtained useful results selecting  $M$  (number of partitions) in the range of 10 to 60, and using  $L \approx N/2$ . When using

Table 3.1: Computational complexity analysis.

Method	Flops
FIR Wiener	$\mathcal{O}(N)$
PWS	$\mathcal{O}(N + NM)$
SPWS (PCA)	$\mathcal{O}(N + KM + KN)$
SPWS (Center-PCA)	$\mathcal{O}(N + KM + KL)$
SPWS (Center)	$\mathcal{O}(N + KM)$

PCA analysis, we have selected  $K$  such that at least 90% of the signal variance is captured in the subspace.

### 3.2 Computational Complexity

In this section we consider the computational complexity of the methods described above in terms of floating point operations (flops) per output pixel. First, implementing the FIR Wiener filter as a convolution process requires  $N$  flops, where one add and one multiply together constitute one flop. Standard VQ on the full  $R^N$  space using Euclidean distances requires  $\mathcal{O}(NM)$  flops. In the subspace, VQ partitioning is performed in  $\mathcal{O}(KM)$  flops (where  $K < N$ ). Projection into the subspace is accomplished in  $KN$  flops using the PCA method and  $KL$  flops when using PCA on the  $L$  selected samples. Based on this analysis, Table 3.1 lists the computational complexity of each of the methods proposed here. Also, the flop counts for the various methods, with  $N = 61$ ,  $L = 31$ , and  $K = 5$ , are plotted in Fig. 3.2 as a function of  $M$ . Note the significant reduction in flops for the SPWS filters for large values of  $M$ . Note that for

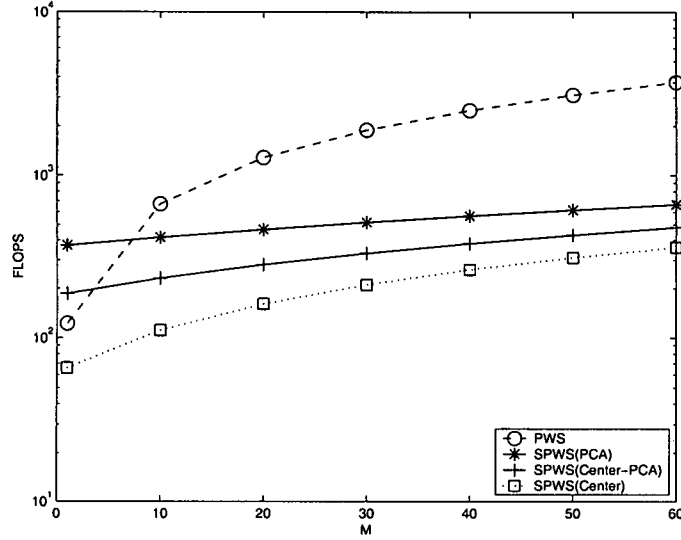


Figure 3.2: Floating point operation count for the filters as a function of  $M$ , with  $N = 61$ ,  $L = 31$ , and  $K = 5$ .

small values of  $M$ , the PCA methods actually increase the computational complexity because of the matrix multiplication discussed in Equ.(3.1). However, this overhead cost quickly becomes negligible as a dramatic reduction in flops is produced for larger values of  $M$ . The SPWS (Center) method has the lowest computational complexity.

### 3.3 Summary

In this chapter, we proposed three types of SPWS filters. We also performed the analysis of the computation complexities of the three SPWS filters. All three types of SPWS filters are approximations of PWS filters. However, in some cases, improved performance of SPWS filters is expected due to improved partitioning.

## CHAPTER 4

### INTERPRETATIONS AND OPTIMIZATION OF SOFT PARTITION WEIGHTED SUM FILTER

In Chapter 2, the Soft-PWS filters have been defined. In this chapter, we first propose novel radial basis function interpretations of the Soft-PWS filters.<sup>4,6</sup> Then, we continue the development of the theory in areas of optimization of the Soft-PWS filters, and derive Numerical solutions<sup>22,23,25</sup> of the optimization processes. The cyclic coordinate descent (CCD) method is introduced to implement the optimization procedure for each parameter. Numerical solutions such as Quasi-Newton method, modified Newton optimization method, the gradient descent method, the golden section search method, and Monte Carlo method are utilized in the study for optimizations.<sup>24,26</sup>

The remainder of the chapter is organized as follows: Section 4.1 presents novel radial basis function interpretations of the Soft-PWS filters. Section 4.2 presents numerical solutions for optimization of Soft-PWS filters. Finally, a summary is given in Section 4.3.

#### 4.1 Radial Basis Function Interpretation of Soft-PWS Filter

It is insightful to note that the Soft-PWS filter can be expressed in terms of a radial basis function.<sup>21,22</sup> To show this, we modify the expression for the Soft-PWS

filter in (2.9) yielding

$$\begin{aligned}
F_{Soft-PWS}(\mathbf{x}) &= \sum_{i=1}^M \hat{p}_i(\mathbf{x}) \mathbf{w}_i^T \mathbf{x} \\
&= \left( \sum_{i=1}^M \hat{p}_i(\mathbf{x}) \mathbf{w}_i^T \right) \mathbf{x} \\
&= \mathbf{w}_{RBF}^T \mathbf{x},
\end{aligned} \tag{4.1}$$

where the weights are formed with a radial basis function as

$$\mathbf{w}_{RBF}^T = \sum_{i=1}^M \hat{p}_i(\mathbf{x}) \mathbf{w}_i^T. \tag{4.2}$$

As shown above, the Soft-PWS partition function,  $\hat{p}_i(\cdot)$ , does not simply quantize the observation samples to one of the  $M$  partitions. But rather, serves as a radial basis function for interpolating the filter weights from the partition weights, allowing each to contribute to the output.

## 4.2 Optimization of Soft-PWS Filters

In this section, we derive an MSE optimization strategy for the Soft-PWS filter. We wish to perform the optimization jointly with respect to the filter weights,  $\mathbf{w}$ , the VQ codebook,  $\mathbf{c}$ , and the radial basis function parameters,  $\mathbf{s} = [s_1, s_2, \dots, s_M]^T$ .

### 4.2.1 Cost Function Definition

To begin the derivation of the optimization procedure, we define the cost function as follows:

$$\begin{aligned}
J(\mathbf{w}, \mathbf{c}, \mathbf{s}) &= E\{(d - F_{Soft-PWS})^2\} \\
&= E\{d^2 - 2dF_{Soft-PWS} + F_{Soft-PWS}^2\} \\
&= E\{d^2\} - 2E\{dF_{Soft-PWS}\} + E\{F_{Soft-PWS}^2\} \\
&= \sigma^2 - 2E\{dF_{Soft-PWS}\} + E\{F_{Soft-PWS}^2\}.
\end{aligned} \tag{4.3}$$

Table 4.1: Optimization overview using a cyclic coordinate descent method.

Step 1	Use LBG algorithm to initialize codebook $\mathbf{c}$ . Use sample variance of observation vectors in each partition to initialize $\mathbf{s}$ .
Step 2	Perform optimization with respect to $\mathbf{w}$ , with $\mathbf{c}$ and $\mathbf{s}$ fixed.
Step 3	Perform optimization with respect to $\mathbf{c}$ , with $\mathbf{w}$ and $\mathbf{s}$ fixed.
Step 4	Perform optimization with respect to $\mathbf{s}$ , with $\mathbf{w}$ and $\mathbf{c}$ fixed.
Step 5	Go to Step 2 if the ending condition is not met.

To make the minimization of (4.3) more tractable, we adopt the cyclic coordinate descent method<sup>23</sup> summarized in Table 4.1. We now focus on the Steps 2-4 in Table 4.1.

#### 4.2.2 Optimization With Respect to $\mathbf{w}$

In order to derive the optimization with respect to  $\mathbf{w}$ , it is helpful to redefine the Soft-PWS filter in (4.1) as

$$\begin{aligned}
 F_{Soft-PWS}(\mathbf{x}) &= \sum_{i=1}^M \mathbf{w}_i^T \tilde{\mathbf{x}}_i, \\
 &= \mathbf{w}^T \tilde{\mathbf{x}},
 \end{aligned} \tag{4.4}$$

where  $\tilde{\mathbf{x}} = [\tilde{\mathbf{x}}_1^T, \tilde{\mathbf{x}}_2^T, \dots, \tilde{\mathbf{x}}_M^T]^T$  and  $\tilde{\mathbf{x}}_i = \mathbf{x} \hat{p}_i(\mathbf{x})$ , for  $i = 1, 2, \dots, M$ . Using (4.4), the cost function in (4.3) can be expressed as

$$\begin{aligned}
 J(\mathbf{w}, \mathbf{c}, \mathbf{s}) &= \sigma^2 - 2E\{d\mathbf{w}^T \tilde{\mathbf{x}}\} + E\{(\mathbf{w}^T \tilde{\mathbf{x}})^2\} \\
 &= \sigma^2 - 2\mathbf{w}^T E\{d\tilde{\mathbf{x}}\} + \mathbf{w}^T E\{\tilde{\mathbf{x}}\tilde{\mathbf{x}}^T\} \mathbf{w}.
 \end{aligned} \tag{4.5}$$

Let us define  $\tilde{\mathbf{p}} = E\{d\tilde{\mathbf{x}}\}$  to be the cross correlation vector between the desired sample and the observation vector. Also define  $\tilde{\mathbf{R}} = E\{\tilde{\mathbf{x}}\tilde{\mathbf{x}}^T\}$  to be the auto correlation matrix of the observation samples. The cost function in (4.5) then becomes

$$J(\mathbf{w}, \mathbf{c}, s) = \sigma^2 - 2\mathbf{w}^T \tilde{\mathbf{p}} + \mathbf{w}^T \tilde{\mathbf{R}} \mathbf{w}. \quad (4.6)$$

Then the gradient of (4.6) with respect to  $\mathbf{w}$  can be expressed as

$$\nabla_{\mathbf{w}} J(\mathbf{w}, \mathbf{c}, s) = -2\tilde{\mathbf{p}} + 2\tilde{\mathbf{R}}\mathbf{w}. \quad (4.7)$$

By setting the gradient to zero and solving for  $\mathbf{w}$ , the optimum Soft-PWS weights (for the fixed  $\mathbf{c}$  and  $s$ ) can be expressed as

$$\mathbf{w}_{Soft-PWS} = \tilde{\mathbf{R}}^{-1} \tilde{\mathbf{p}}. \quad (4.8)$$

Note that  $\tilde{\mathbf{p}}$  and  $\tilde{\mathbf{R}}$  can be estimated using sample estimates from statistically representative training data.

### 4.2.3 Optimization With Respect to $\mathbf{c}$ and $s$

It is very difficult to obtain a closed form solution for minimizing the cost function (4.3) with respect to  $\mathbf{c}$  or  $s$ . However, it is possible to derive the gradient of (4.3) with respect to these parameters. Therefore, we have investigated a variety of gradient-based iterative methods. This has led us to a compound numerical solution that uses alternating application of the Quasi-Newton and steepest descent methods.<sup>23</sup> While both the Quasi-Newton and steepest descent methods individually are promising and achieve good results, we have found that better results can be obtained with the compound numerical solution.

In particular, we begin with using the Quasi-Newton method, which is summarized in Table 4.2, and we continue until the halting criterion is met. The gradient,

$\nabla_{\mathbf{c}} J(\mathbf{w}, \mathbf{c}, \mathbf{s})$ , used in Steps 3 and 6 is derived in Appendix A. Note in Step 4 of Table 4.2, we select  $\alpha$  using golden section search method.<sup>24</sup> We have found that the further improvement is possible by following the Quasi-Newton method with the method of steepest descent summarized in Table 4.3. As before, the step size used in Step 3 in Table 4.3 is computed using the golden section search method. The gradient descent method is continue until the halting criterion shown in Step 4 is met. At this point we return to the Quasi-Newton method for further improvement. We continue this alternating process until the halting criteria of both methods are met.

The proposed optimization of the radial basis function parameters,  $\mathbf{s}$ , is similar to that described above for  $\mathbf{c}$ . For the sake of brevity, we will not repeat all the steps presented above. However, since the key to the procedure is obtaining the the gradient of the cost function with respect to  $\mathbf{s}$ , we present that development in Appendix B.

### 4.3 Summary

In this chapter, we proposed novel radial basis function interpretations of the Soft-PWS filters. We also continued the development of the theory in areas of optimization of the Soft-PWS filters. Numerical solutions of the optimization processes are derived. Compound numerical solutions are used in the study for optimizations.

Table 4.2: Quasi-Newton Optimization Method Overview.

Step 1	Set initial values of $\mathbf{c}^{(0)}$ .
Step 2	Set initial value of $\mathbf{H}^{(0)} = \mathbf{I}$ . $\mathbf{H}^{(k)}$ is a symmetric positive definite matrix, and is used to approximate the second derivative function. $\mathbf{H}^{(k)}$ is updated by iterations.
Step 3	Determine the direction of descent $\delta^{(k)}$ by $\delta^{(k)} = -\mathbf{H}^{(k)} \nabla_{\mathbf{c}} J(\mathbf{w}, \mathbf{c}^{(k)}, \mathbf{s})$ at point of $\mathbf{c}^{(k)}$ .
Step 4	Determine the best step size $\alpha$ using a line search method by minimizing $J(\mathbf{w}, \mathbf{c}^{(k)} + \alpha \delta^{(k)}, \mathbf{s})$ .
Step 5	Set $\mathbf{c}^{(k+1)} = \mathbf{c}^{(k)} + \alpha \delta^{(k)}$ .
Step 6	Update $\theta^{(k)}$ and $\gamma^{(k)}$ . $\theta^{(k)}$ and $\gamma^{(k)}$ are used for updating $\mathbf{H}^{(k+1)}$ . $\theta^{(k)} = \alpha^k \delta^{(k)} = \mathbf{c}^{(k+1)} - \mathbf{c}^{(k)}$ , and $\gamma^{(k)} = \nabla_{\mathbf{c}} J(\mathbf{w}, \mathbf{c}^{(k+1)}, \mathbf{s}) - \nabla_{\mathbf{c}} J(\mathbf{w}, \mathbf{c}^{(k)}, \mathbf{s})$ . If $\ \gamma^{(k)}\  \leq \epsilon$ , then stop.
Step 7	Update $\mathbf{H}^{(k+1)}$ by using $\mathbf{H}^{(k+1)} = \mathbf{H}^{(k)} + \frac{(\theta^{(k)} - \mathbf{H}^{(k)} \gamma^{(k)})(\theta^{(k)} - \mathbf{H}^{(k)} \gamma^{(k)})^T}{(\theta^{(k)} - \mathbf{H}^{(k)} \gamma^{(k)})^T \gamma^{(k)}}$ .

Table 4.3: Steepest Descent Optimization Overview.

Step 1	Set initial values of $\mathbf{c}^{(0)}$ .
Step 2	Determine the direction of descent $\delta^{(k)}$ by calculating the first derivative values $\delta^{(k)} = -\nabla_{\mathbf{c}} J(\mathbf{w}, \mathbf{c}^{(k)}, \mathbf{s})$ at point of $\mathbf{c}^{(k)}$ .
Step 3	Find the best step size $\alpha$ using a line search algorithm by minimizing the cost function $J(\mathbf{c}^{(k)} + \alpha\delta^{(k)})$ .
Step 4	Set $\mathbf{c}^{(k+1)} = \mathbf{c}^{(k)} + \alpha\delta^{(k)}$ . If $\ J^{(k)} - J^{(k+1)}\  \leq \varepsilon$ , then stop. Otherwise, go to Step 2.

## CHAPTER 5

### SUBSPACE PARTITION WEIGHTED SUM FILTER EXPERIMENTAL RESULTS

In this chapter, we evaluate the performance of the SPWS filter. The subspace PWS filters proposed in Chapter 3 are targeted to reduce the computational complexity with little or no loss in performance of the PWS filters. In this chapter, three types of SPWS filters are evaluated and compared with the FIR Wiener filter and PWS filter in image deconvolution and image noise reduction applications. Center SPWS filters are the simplest algorithm to reduce the observation window size and its computation complexity. PCA SPWS filters can be implemented to further reduce the observation window size. The Center-PCA SPWS filters are combinations of the Center SPWS filters and the PCA SPWS filters.

The remainder of this chapter is organized as follows: Section 5.1 presents the image deconvolution experimental results of the SPWS filters. In this section, the comparisons of the three SPWS filters with the Wiener linear filters and generic PWS non-linear filters show that the ability of the SPWS filters reduce the computation complexities is promising. In some cases, they produce better results than the generic PWS filters. Section 5.2 presents the image noise reduction experimental results of

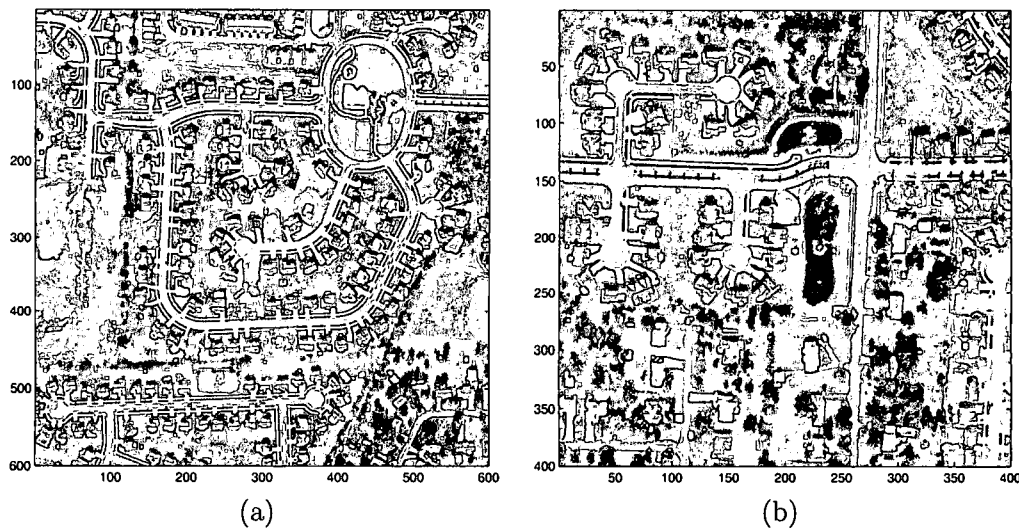


Figure 5.1: (a) The true training image ( $600 \times 600$ ); (b) the true test image ( $400 \times 400$ ).

the SPWS filters. Finally, a summary is given in Section 5.3. Note that the Matlab source codes used to generate the results in this chapter are shown in Appendix C.

## 5.1 Image Deconvolution

A  $600 \times 600$  8 bit training image and a separate  $400 \times 400$  test image are obtained from different parts of a single high-resolution aerial image. The true training image is shown in Fig. 5.1(a), and Fig. 5.1(b) shows the true test image. The images are considered ideal and are artificially degraded to allow for quantitative error analysis. The training image is used to generate the VQ codebook, the PCA transformation, and determine the weights of the SPWS filters for each partition. The test image is used exclusively for quantitative analysis and visual demonstration.

We begin by considering the problem of restoring an image with linear motion blur. Fig. 5.2(a) shows the enlarged true test image ( $100 \times 100$ ). Fig. 5.2(b) shows the enlarged motion blurred image with Gaussian noise (standard deviation is 1% of the dynamic range,  $\text{MSE} = 36.68$ ). Fig. 5.2(c) shows the Wiener output ( $\text{MSE} = 21.09$ ), and Fig. 5.2(d) shows the PWS output ( $\text{MSE} = 20.55$ ). Fig. 5.3(a) shows SPWS filter output using the Center method ( $\text{MSE} = 20.40$ ), Fig. 5.3(b) shows SPWS filter output using the PCA method ( $\text{MSE} = 20.53$ ), and Fig. 5.3(c) shows the SPWS filter output using the Center-PCA method ( $\text{MSE} = 20.37$ ). Note that in all of these results, the observation window size is  $N = 1 \times 51$ , and the partition numbers are  $M = 50$ ,  $K = 5$ , and  $L = 25$ .

Quantitative error analysis is shown in Fig. 5.4. In particular, MSE is plotted as a function of the number of partitions,  $M$ , for the various filters. Note that for  $M = 1$ , the PWS and SPWS filters are equivalent to the Wiener filter. The results shown in Figs. 5.2 and 5.3 show improvement with an increase in the number of VQ partitions. The SPWS (Center and Center-PCA) had lower MSE than the PWS filter with larger values of  $N$ . This may be attributable to improved partitioning. The results show an improvement of 2 - 7% versus the Wiener filter, which was expected.

Fig. 5.5 shows the time consumed during training process. The results show that for small values of  $M$ , the PCA methods actually increase the computation complexity. However, this overhead cost quickly becomes negligible as a dramatic reduction in training time is produced for larger values of  $M$ . As discussed above, the Center SPWS filters has the lowest training time.

Fig. 5.6 shows the time consumed during the filtering process. The results show all three SPWS filters reduced filtering time, with the Center SPWS filters having the

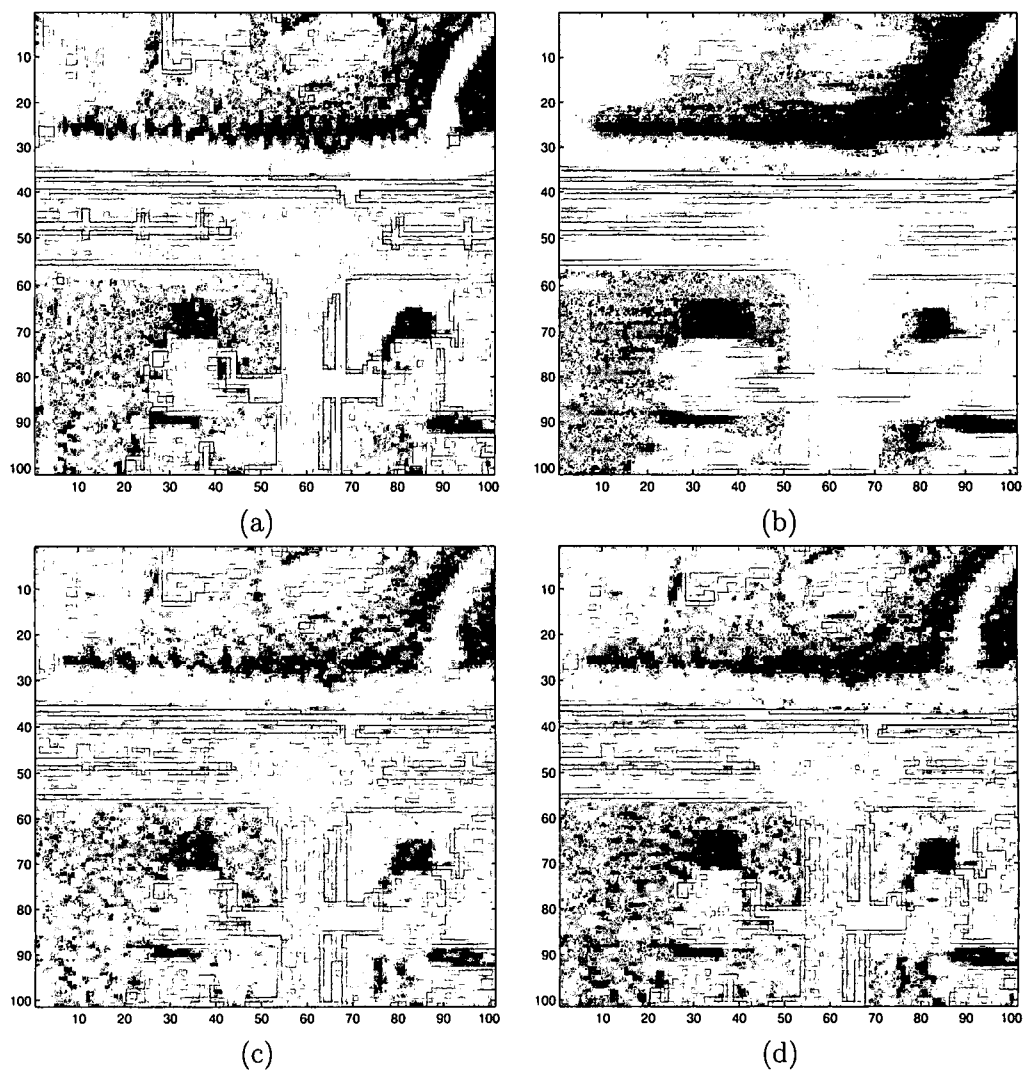


Figure 5.2: Enlarged portion ( $100 \times 100$ ) of (a) desired test image ; (b) motion blurred image with Gaussian noise (standard deviation=1%, MSE=36.68); (c) Wiener output; and (d) PWS output ( $N = 1 \times 51$ ,  $M = 50$ ).

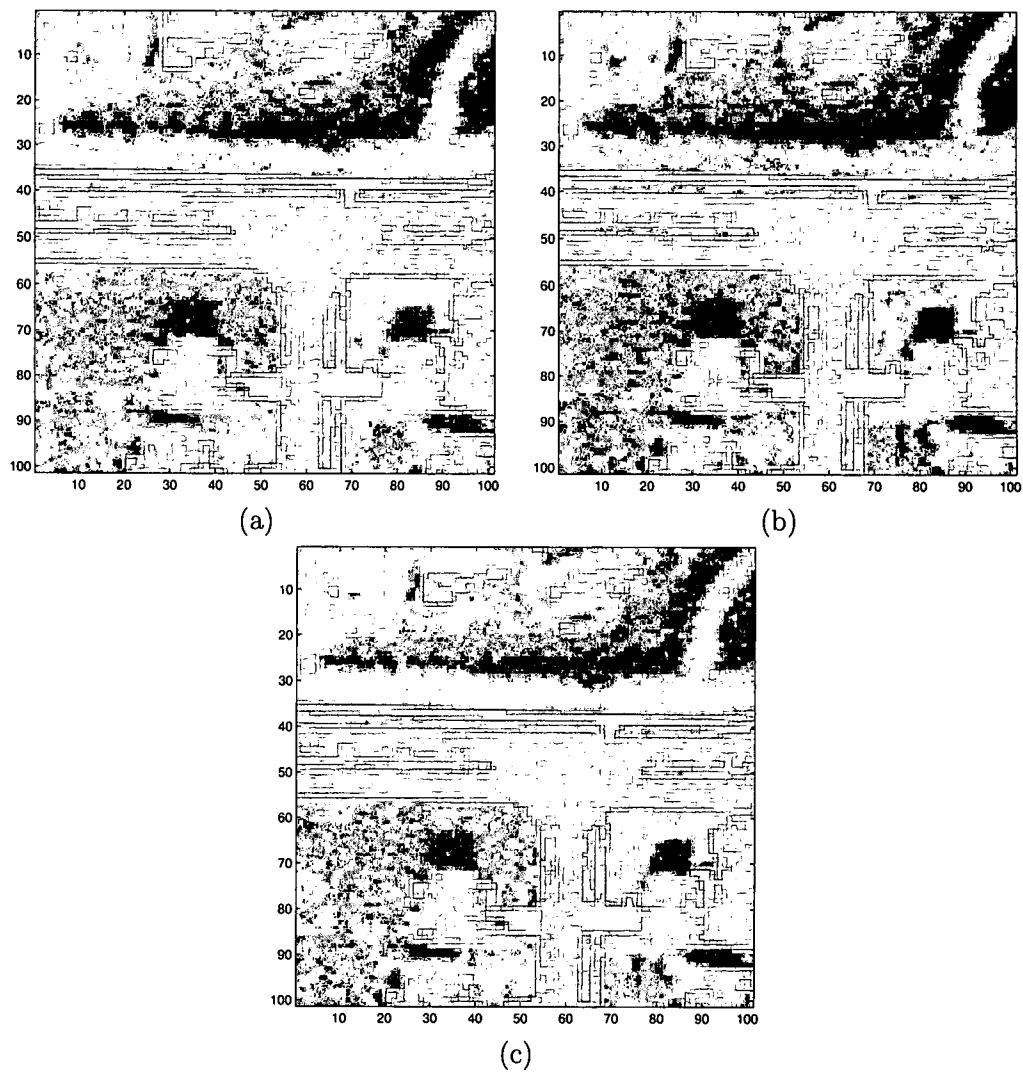


Figure 5.3: Enlarged portion ( $100 \times 100$ ) of image deconvolution results: (a) SPWS (Center) output; (b) SPWS (PCA) output; and (c) SPWS (Center-PCA) output ( $N = 1 \times 51$ ,  $M = 50$ ,  $K = 5$ , and  $L = 25$ ).

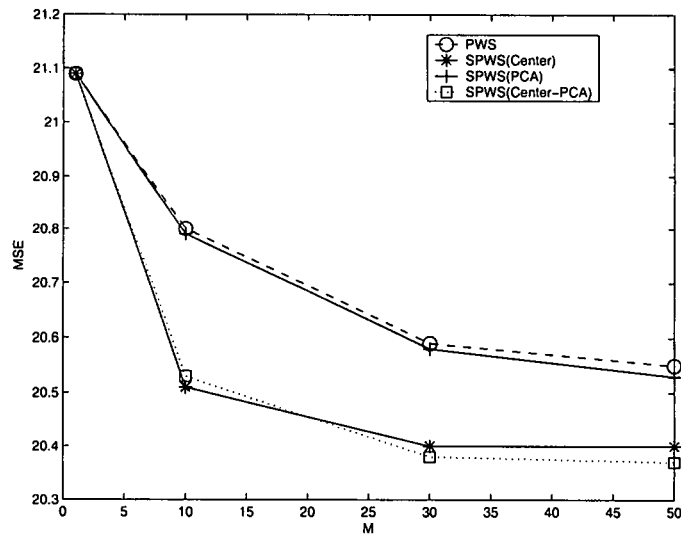


Figure 5.4: MSE analysis for the various filters applied to a motion blurred image with Gaussian noise ( $N = 1 \times 51$ ,  $M = 50$ ,  $K = 5$ , and  $L = 25$ ).

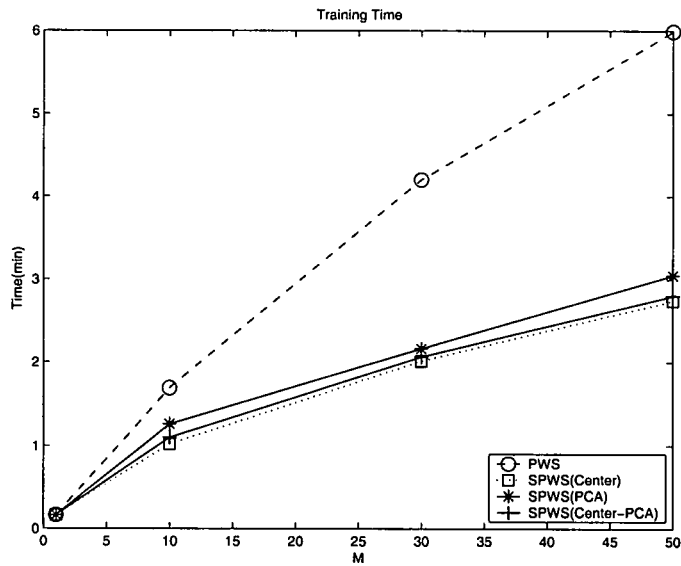


Figure 5.5: Training implementation time.

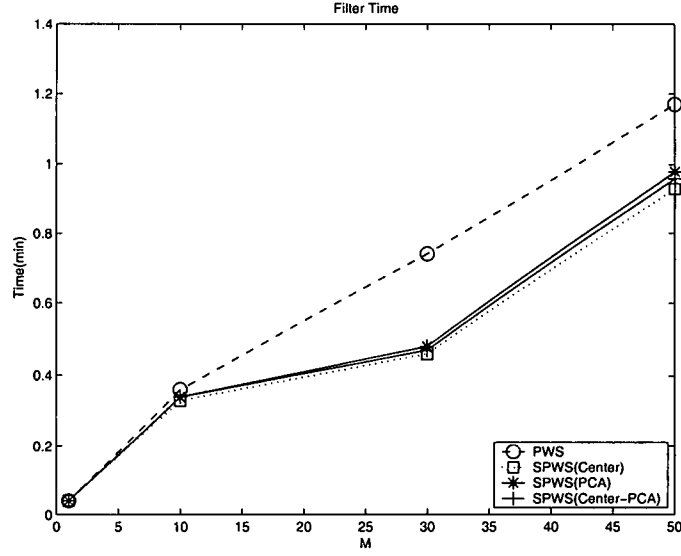


Figure 5.6: Filter implementation time.

lowest filtering time. Fig. 5.7 shows the total implementation time which combined both training and filtering. From this, we can conclude that all three SPWS filters dramatic reduced computation complexities as promised, and the Center SPWS filters have the lowest computation complexities.

For the PCA method, setting  $K = 5$  preserves approximately 93% of the signal variance from the full observation space. For the Center-PCA method, setting  $K = 5$  and  $L = 25$  preserves approximately 98%. Note that the SPWS and PWS images appear to be very similar. Since the PCA method only keeps  $K$  components with large eigenvalues, it is important to know the quality of the approximation. Principally,  $K$  can be any value as far as  $K < N$ . In this dissertation,  $K = 5$  is selected to maintain approximation performance over 90%. The summation of total eigen values is 0.9651,

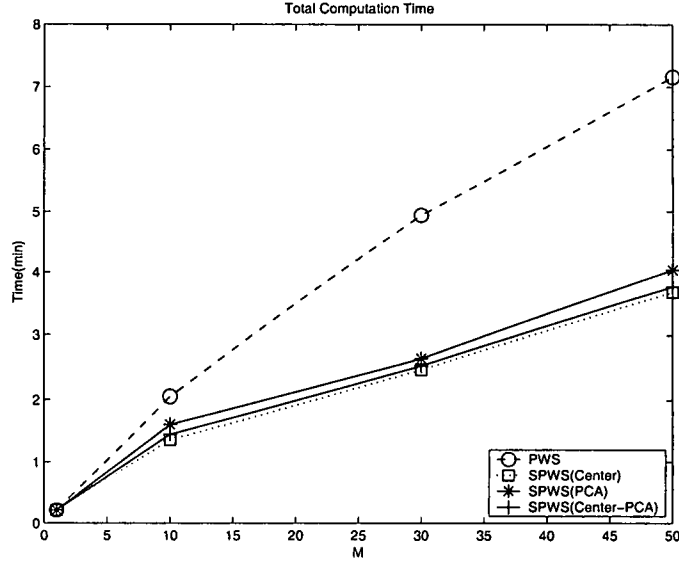


Figure 5.7: Total implementation time.

the summation of top 5 eigen values is 0.8981, which results in an approximation performance equal to 93%.

## 5.2 Image Noise Reduction

In this section, we apply the SPWS filter to image noise reduction applications. The results are expected to be similar to those in previous sections. The same training and test image set used in the previous section are used in this case. The noisy images are generated from the desired images shown in Fig. 5.1 by artificially adding white Gaussian noise (standard deviation is 10% of the dynamic range). Fig. 5.8(a) shows the enlarged test image with Gaussian noise (standard deviation is 10% of the dynamic range, MSE = 39.66). Fig. 5.8(b) shows Wiener filter output (MSE = 22.48,  $N = 7 \times 7$ ). The PWS output is shown in Fig. 5.8(c) (MSE = 20.98), and the SPWS

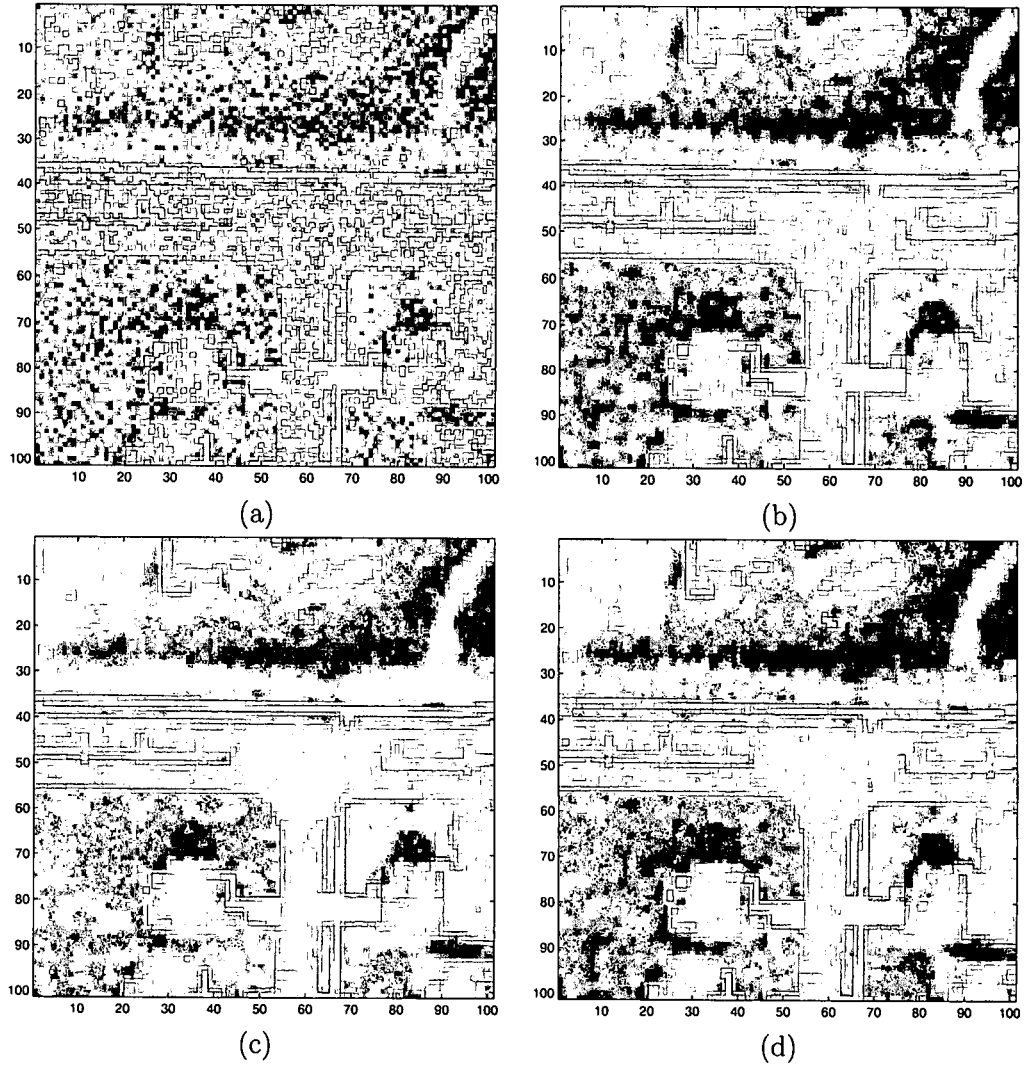


Figure 5.8: Enlarged portion ( $100 \times 100$ ) of (a) test image with Gaussian noise (standard deviation=10%, non-blur,  $\text{MSE} = 39.66$ ); (b) Wiener output ( $\text{MSE} = 22.48$ ,  $N = 7 \times 7$ ); (c) PWS output ( $\text{MSE} = 20.98$ ); and (d) SPWS (Center) output ( $\text{MSE} = 20.76$ ,  $N = 7 \times 7$ ,  $M = 50$ , and  $L = 5 \times 5$ ).

(Center method) output is shown in Fig. 5.8(d) ( $\text{MSE} = 21.08$ ). The SPWS PCA method output is shown in Fig. 5.9(a) ( $\text{MSE} = 21.01$ ), and the SPWS (Center-PCA) method output is shown in Fig. 5.9(b) ( $\text{MSE} = 20.76$ ). Note that in all of these results, the observation window size is  $N = 7 \times 7$ , the partition number is  $M = 50$ ,  $K = 3 \times 3$ , and  $L = 5 \times 5$ . The SPWS center method showed a 6.2% improvement over the Wiener filter, and the SPWS PCA method and the SPWS Center-PCA method showed a 6.5% and 7.7% improvement vs a 6.7% improvement in the generic PWS. As discussed in Chapter 3 and shown in Fig. 3.2, from the computation complexities, the SPWS Center method has the lowest, and the SPWS Center-PCA method. The SPWS PCA has the highest computation requirement. The results shown in Figs. 5.8 and 5.9 showed that the performance of all three SPWS filters are very similar to the PWS filter results, which is what we expected.

Quantitative error analysis is shown in Fig. 5.10. In particular, MSE is plotted as a function of the number of partitions,  $M$ , for the various filters. Note that for  $M = 1$ , the PWS and SPWS filters are equivalent to the Wiener filter. As shown in previous section, the SPWS Center-PCA method had a lower MSE than the PWS filter with larger values of  $N$ . This may be attributable to improved partitioning. The results show improvement of 6.2 - 7.7% versus the Wiener filter.

Fig. 5.11 shows the time consumed during training process. The results show a trend similar to the one discussed in the previous section: for small values of  $M$ , the PCA method actually increases the computation complexity. However, this overhead cost quickly becomes negligible, as a dramatic reduction in training time is produced for larger values of  $M$ . As discussed above, the Center SPWS filters has the lowest training time.

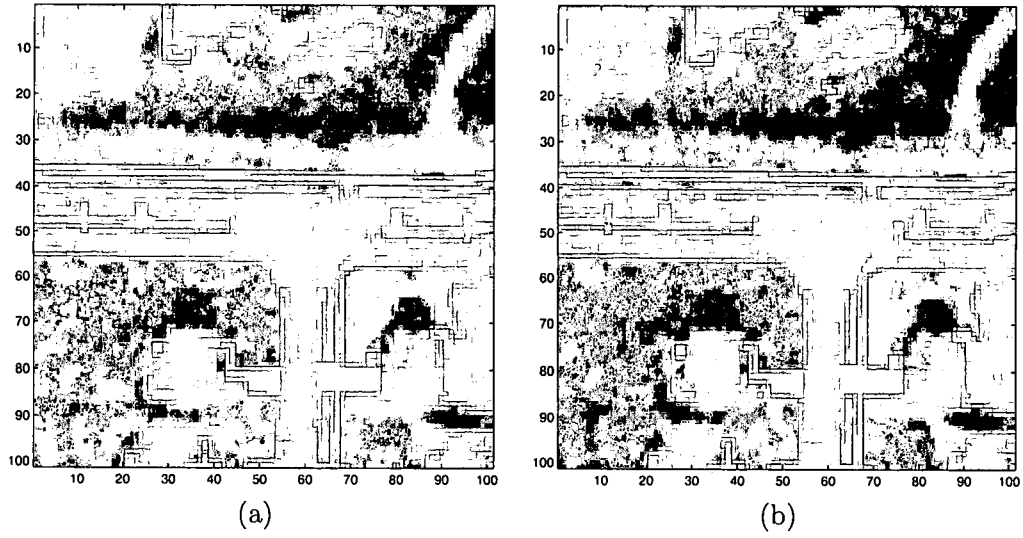


Figure 5.9: Enlarged portion ( $100 \times 100$ ) of image noise reduction results: (a) SPWS (PCA) (MSE=21.01); (b) SPWS (Center-PCA) (MSE = 20.76,  $N = 7 \times 7$ ,  $M = 50$ ,  $K = 3 \times 3$ , and  $L = 5 \times 5$ ).

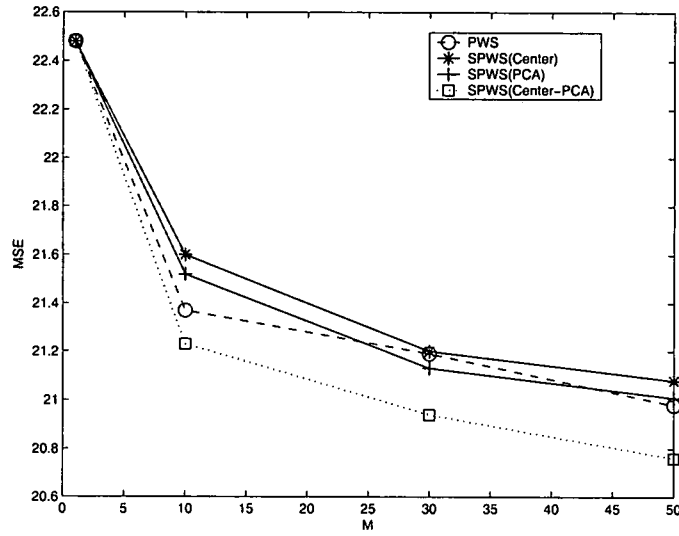


Figure 5.10: MSE analysis for the various filters ( $N = 7 \times 7$ ,  $M = 50$ ,  $K = 3 \times 3$ , and  $L = 5 \times 5$ ).

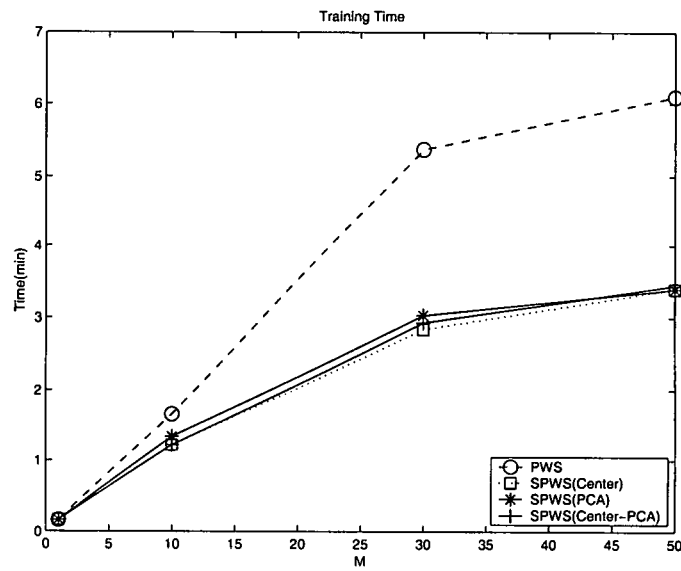


Figure 5.11: Image noise reduction training time.

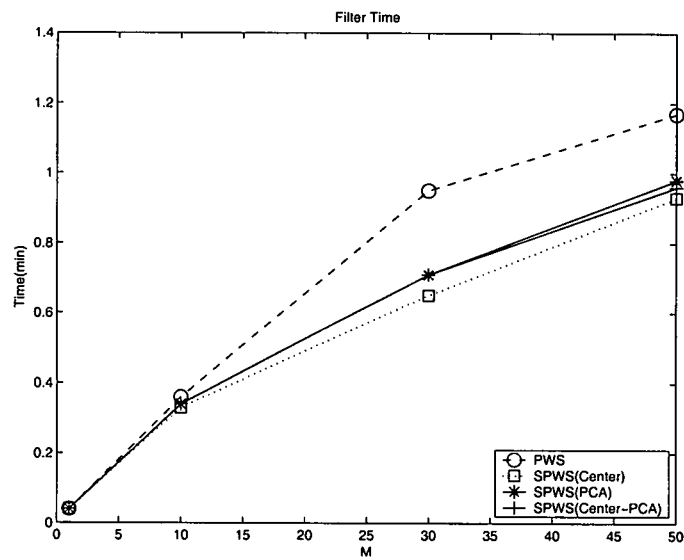


Figure 5.12: Image noise reduction filter implementation time.

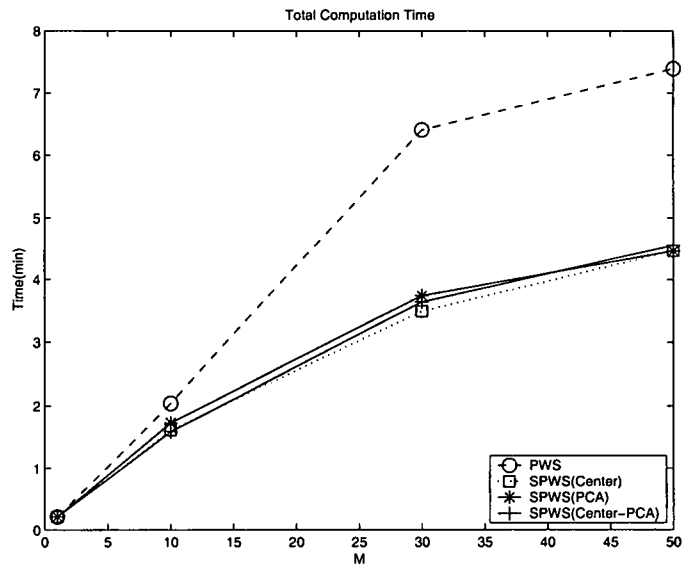


Figure 5.13: Image noise reduction total implementation time.

Fig. 5.12 shows the time consumed during the filtering process, and the total implementation time which combines training and filtering is shown in Fig. 5.13. The results show a trend similar to the one discussed in the previous section.

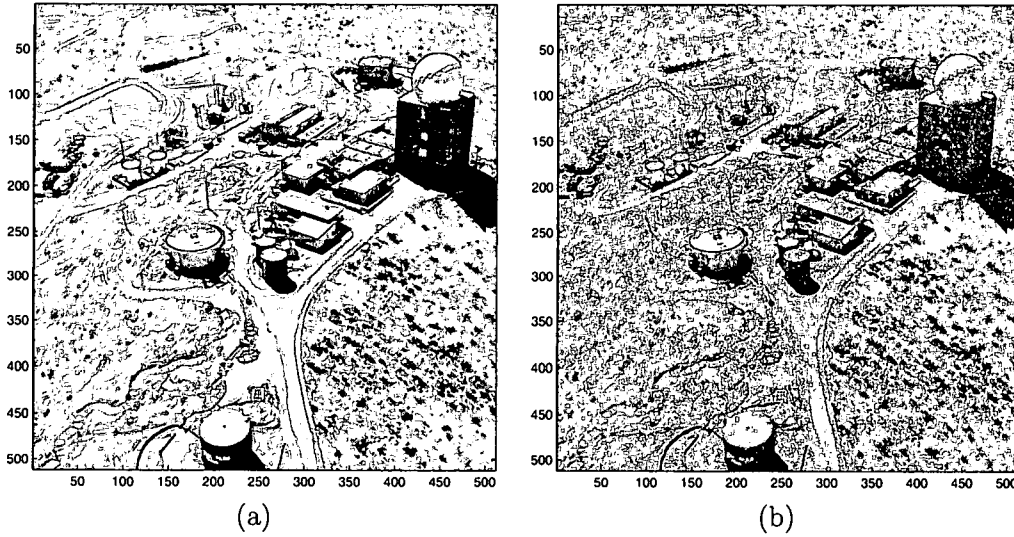


Figure 5.14: (a) The true test aerial image; and (b) the noisy test aerial image ( $512 \times 512$ ).

### 5.2.1 Aerial Images

We applied the three SPWS filters to the aerial images shown in Fig. 5.14. Fig. 5.14(a) shows the desired test aerial image ( $512 \times 512$ ), and the noisy test aerial image ( $512 \times 512$ ) is shown in Fig. 5.14(b).

Fig. 5.15(a) shows the enlarged test image with Gaussian noise (standard deviation is 10% of the dynamic range,  $MSE = 77459$ ). Fig. 5.15(b) shows the Wiener filter output ( $MSE = 7224$ ,  $N = 7 \times 7$ ). The PWS output is shown in Fig. 5.15(c) ( $MSE = 6855$ ), and the SPWS Center method output is shown in Fig. 5.15(d) ( $MSE = 6900$ ). The SPWS PCA method output is shown in Fig. 5.16(a) ( $MSE = 6869$ ), and the SPWS (Center-PCA) method output is shown in Fig. 5.16(b) ( $MSE = 6845$ ). Note that in all of these results, the observation window size is  $N = 7 \times 7$ , the partition number is  $M = 50$ ,  $K = 3 \times 3$ , and  $L = 5 \times 5$ . The SPWS Center method showed a

4.5% improvement over the Wiener filter, and the SPWS PCA method and the SPWS Center-PCA method showed a 4.9% and 5.2% improvement vs a 5.1% improvement of the PWS. As discussed in Chapter 3 and shown in Fig. 3.2, from the computation complexities, the SPWS center method has the lowest, and the SPWS PCA has the highest computation requirement. Figs. 5.15 and 5.16 showed that the performance of all three SPWS filters are very similar to the PWS filter results, which is what we expected.

Quantitative error analysis is shown in Fig. 5.17. In particular, MSE is plotted as a function of the number of partitions,  $M$ , for the various filters. As shown in the previous section, the SPWS Center-PCA method had lower MSE than the PWS filter with larger values of  $N$ . This may be attributable to improved partitioning. The SPWS filter results showed an improvement of 4.5 - 5.2% versus the Wiener filter.

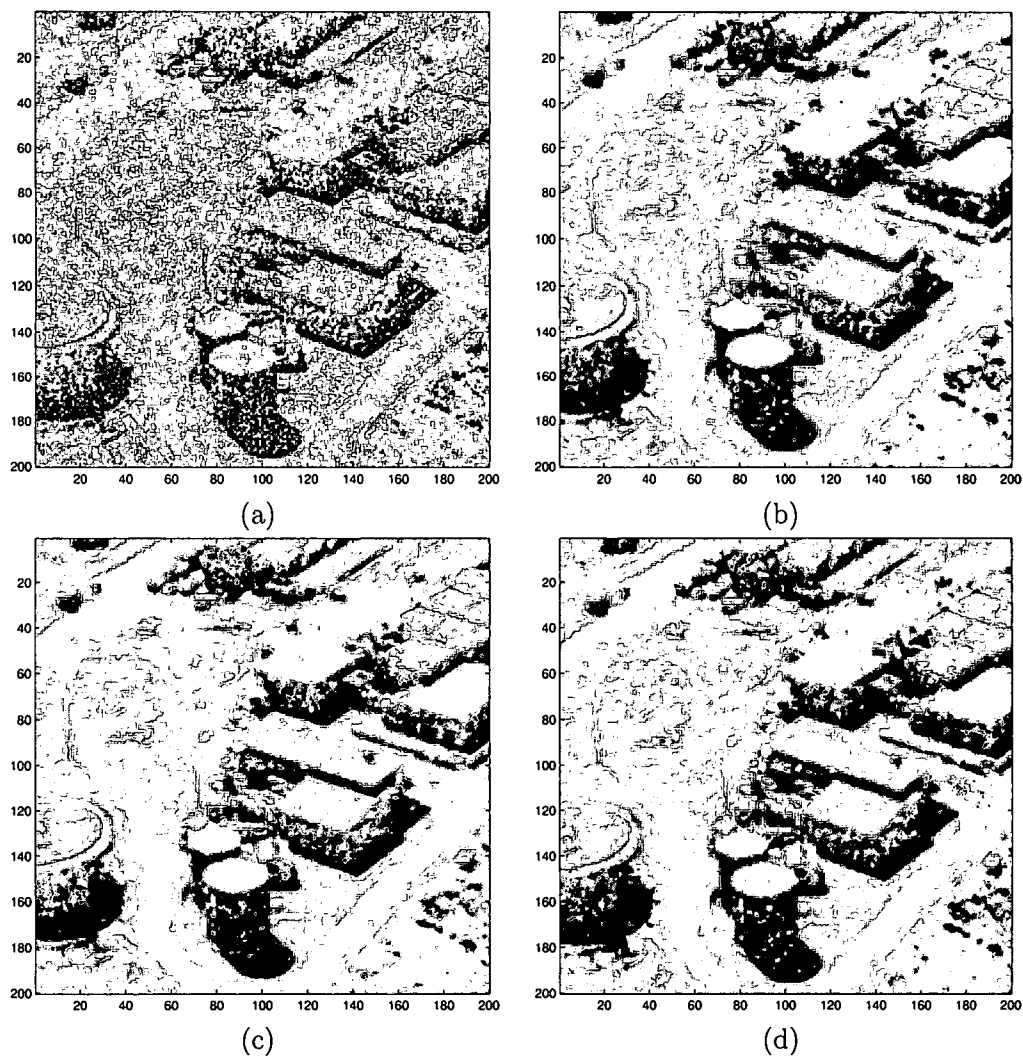


Figure 5.15: Enlarged portion ( $100 \times 100$ ) of (a) Image with Gaussian noise (standard deviation=10%, non-blur); (b) Wiener filter output; (c) PWS output; and (d) SPWS (Center) output ( $N = 7 \times 7$ ,  $M = 50$ , and  $L = 5 \times 5$ ).

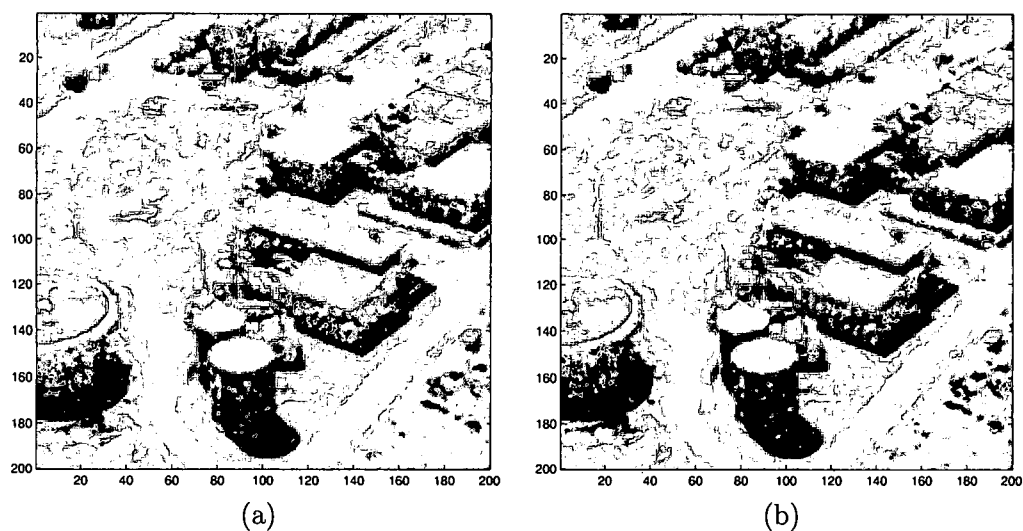


Figure 5.16: Enlarged portion ( $100 \times 100$ ) of image noise reduction results: (a) SPWS (PCA); and (b) SPWS (Center-PCA) ( $N = 7 \times 7$ ,  $M = 50$ ,  $K = 3 \times 3$ , and  $L = 5 \times 5$ ).

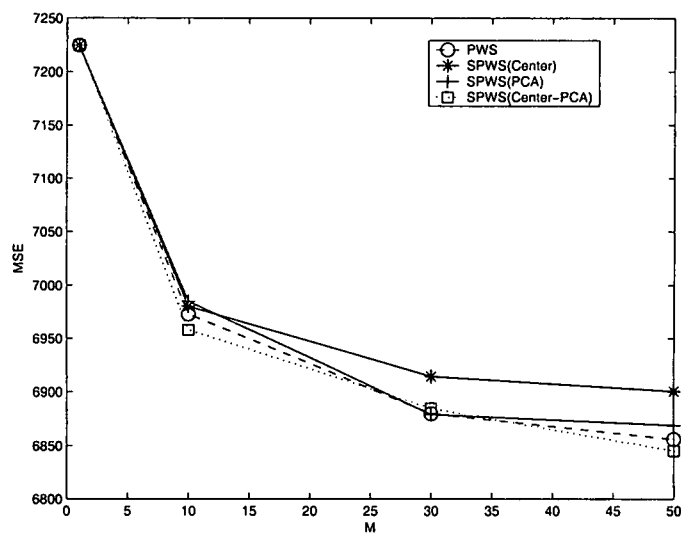


Figure 5.17: MSE for the various filters ( $N = 7 \times 7$ ,  $M = 50$ ,  $K = 3 \times 3$ , and  $L = 5 \times 5$ ).

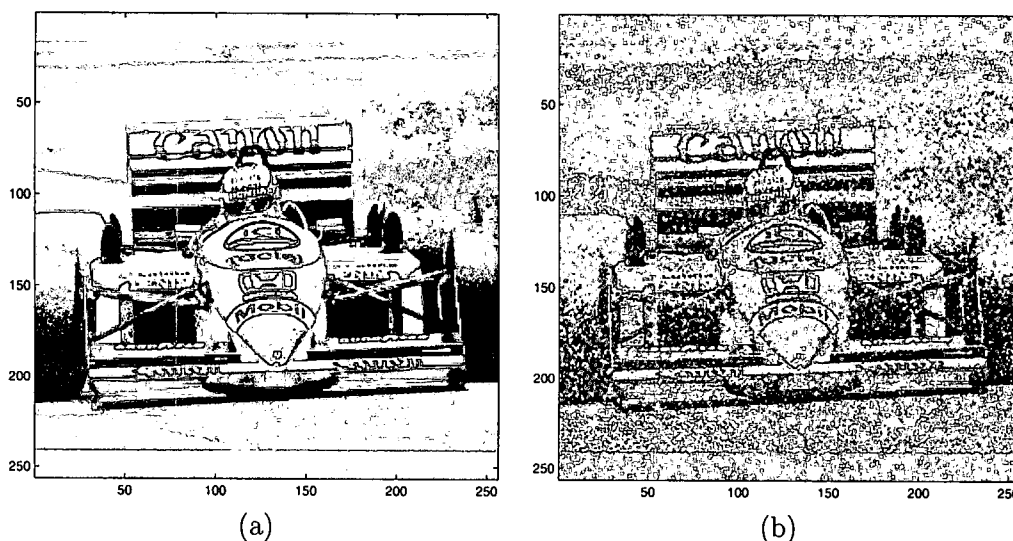


Figure 5.18: (a) The true test car image; and (b) the noisy test car image ( $256 \times 256$ ).

### 5.2.2 Car Images

We applied the three SPWS filters to the car images shown in Fig. 5.18. Fig. 5.18(a) shows the desired test car image ( $256 \times 256$ ), and the noisy test car image ( $256 \times 256$ ) is shown in Fig. 5.18 (b). The test results are shown in Figs. 5.19 and 5.20.

Fig. 5.19(a) shows the enlarged test image with Gaussian noise (standard deviation is 10% of the dynamic range,  $MSE = 33658$ ). Fig. 5.19(b) shows the Wiener filter output ( $MSE = 5087$ ,  $N = 7 \times 7$ ). The PWS output is shown in Fig. 5.19(c) ( $MSE = 4841$ ), and the SPWS Center method output is shown in Fig. 5.19(d) ( $MSE = 4867$ ). The SPWS PCA method output is shown in Fig. 5.20(a) ( $MSE = 4836$ ), and the SPWS (Center-PCA) method output is shown in Fig. 5.20(b) ( $MSE = 4861$ ).

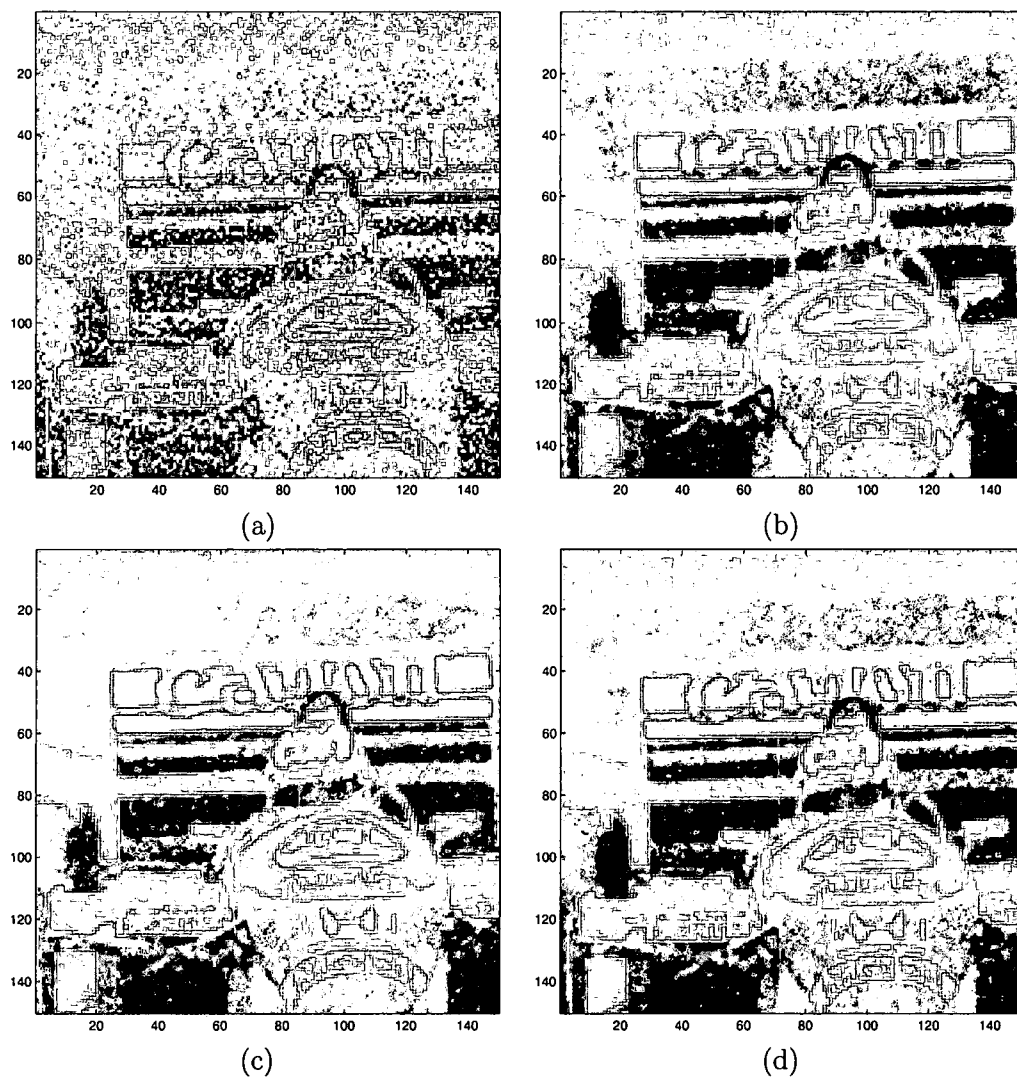


Figure 5.19: Enlarged portion ( $100 \times 100$ ) of (a) Image with Gaussian noise (standard deviation=10%); Image noise reduction results: (b) Wiener filter output ( $N = 7 \times 7$ ); (c) PWS output; and (d) SPWS (Center) output ( $N = 7 \times 7$ ,  $M = 50$ , and  $L = 5 \times 5$ ).

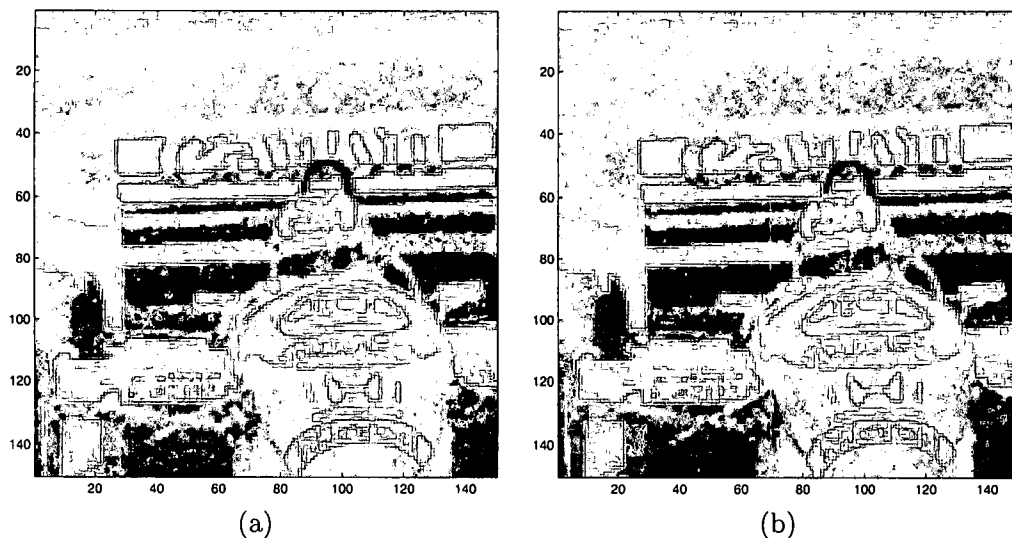


Figure 5.20: Enlarged portion ( $100 \times 100$ ) of image noise reduction results: (a) SPWS (PCA); and (b) SPWS (Center-PCA) ( $N = 7 \times 7$ ,  $M = 50$ ,  $K = 3 \times 3$ , and  $L = 5 \times 5$ ).

Note that in all of these results, the observation window size is  $N = 7 \times 7$ , the partition number is  $M = 50$ ,  $K = 3 \times 3$ , and  $L = 5 \times 5$ . The SPWS Center method showed a 4.3% improvement over the Wiener filter, and the SPWS PCA method and the SPWS Center-PCA method showed a 5.0% and 4.5% improvement vs a 4.8% improvement of the PWS. Figs. 5.19 and 5.20 results showed that the performance of all three SPWS filters are very similar to the PWS filter results, which is what we expected.

Quantitative error analysis is shown in Fig. 5.21.

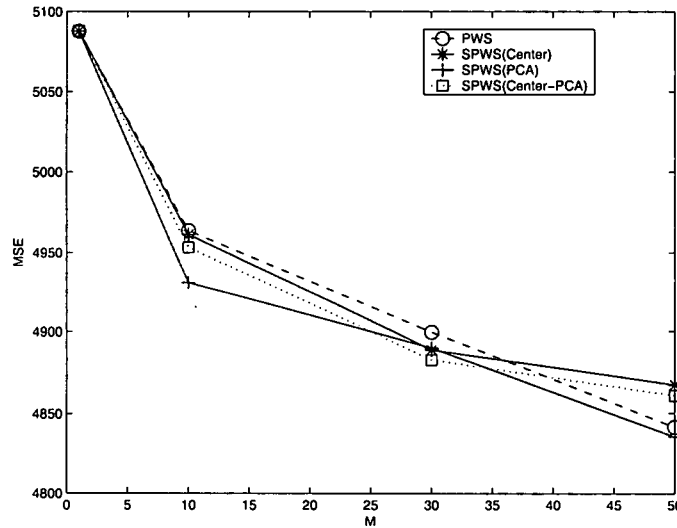


Figure 5.21: MSE for the various filters ( $N = 7 \times 7$ ,  $M = 50$ ,  $K = 3 \times 3$ , and  $L = 5 \times 5$ ).

### 5.3 Summary

The bulk of the computational complexity of the PWS comes from the partitioning of the observation space. We have proposed the SPWS filter, which significantly reduces the computational and memory demands of the partitioning process by using PCA, selecting the center of the filter window, or both. Because of these reductions, the SPWS filters allows for larger filter window sizes than would be practical with the PWS filter.

The quantitative error analysis results show that the performance of SPWS filter is essentially equivalent to, or slightly better than, PWS filters of the same size. However, the SPWS filter requires less memory and can be implemented with significantly fewer computations. In the software implementation used for the results presented

here, the SPWS (Center, PCA and Center-PCA methods) required less than half of the computation time of the PWS for training and testing together. This observation result is consistent with the analysis in Table 3.1.

We have observed that the performance of these partition filters is heavily dependent on the VQ codebook. With the SPWS filter (Center method), which incorporates only the center portion of the observation window into the partitioning scheme not only provides computational savings, but actually improves performance in most cases. We believe that the reason for this is that the center portion of the observation window contains samples that are most correlated to the desired sample, and generally corresponds to the largest magnitude filter weights. This lends increased “importance” to the center of the observation vector for partitioning. We have also shown that reduction in the dimensionality of the partition space can be achieved through PCA with little or no increase in MSE.

## CHAPTER 6

### IMPROVED OPTIMIZATION OF SOFT PARTITION WEIGHTED SUM FILTERS EXPERIMENTAL RESULTS

The Soft-PWS filter using the proposed optimization procedure may be employed in a variety of signal and image restoration applications. Here we consider the application of noise reduction in natural visible images. The performance of the Soft-PWS filter using the proposed optimization methods is compared with that of the FIR Wiener filter and the standard PWS filter.

The remainder of this chapter is organized as follows: Section 6.1 presents the image noise reduction experimental results for natural images. In this section, the comparisons of linear filters, such as Wiener filter, median filter, moving average filter, and PWS non-linear filter show that the Soft-PWS filters produce better results. Four different image sets are used to demonstrate the effectiveness of the Soft-PWS filters. Section 6.2 presents the image noise reduction experimental results for biomedical images. Results from Soft-PWS filters are also presented and compared with both linear and PWS non-linear filters. Two- and three-dimensional CT biomedical images are used. Finally, a summary is given in Section 6.3. Note that the Matlab source codes used to generate the results in this chapter are shown in Appendix D.

## 6.1 Natural Image Noise reduction Results

In this section, the performance of the Soft-PWS filter optimized by proposed closed form solutions and numerical solutions is compared with that of the commonly-used imaged noise reduction linear filters: the FIR Wiener filters, median filter, the moving average filter, and the generic PWS non-linear filter. Natural images are used to demonstrate the effectiveness of the Soft-PWS filters. The images are taken from University of Dayton and University of Delaware image processing databases.

### 6.1.1 Training Process

The only training image used this section is shown in Fig. 6.1. It is a 8 bit  $434 \times 491$  aerial image. The training image is used exclusively for training purposes to generate the Soft-PWS filters partitions/codebook and the Soft-PWS filters coefficients for each partition, and to perform optimization process.

Fig. 6.2 shows 50 sets of  $7 \times 7$  codewords which obtained from the training image. The observation window size is  $N = 7 \times 7$ , and the partition number  $M = 50$ . The codebook shows a trend of structure elements which are used to optimize the filtering. The SPWS Center method's 50 sets of  $3 \times 3$  codewords which obtained from the  $7 \times 7$  observation window are shown in Fig. 6.3.

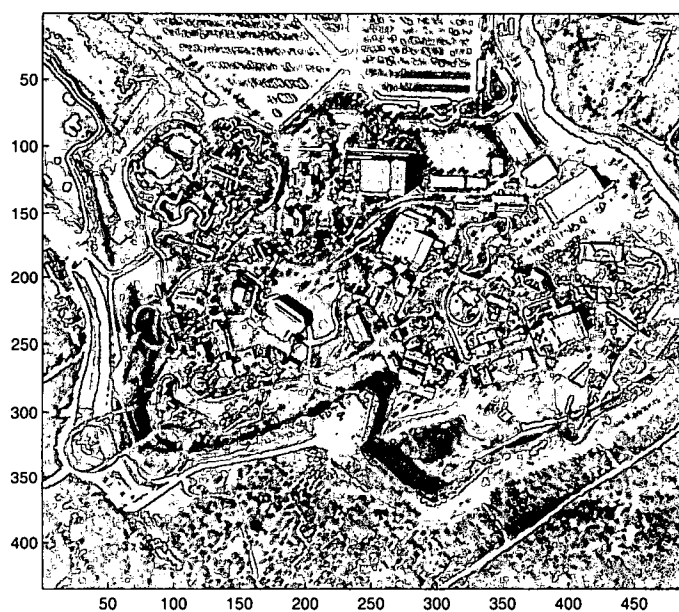


Figure 6.1: The desired training image.

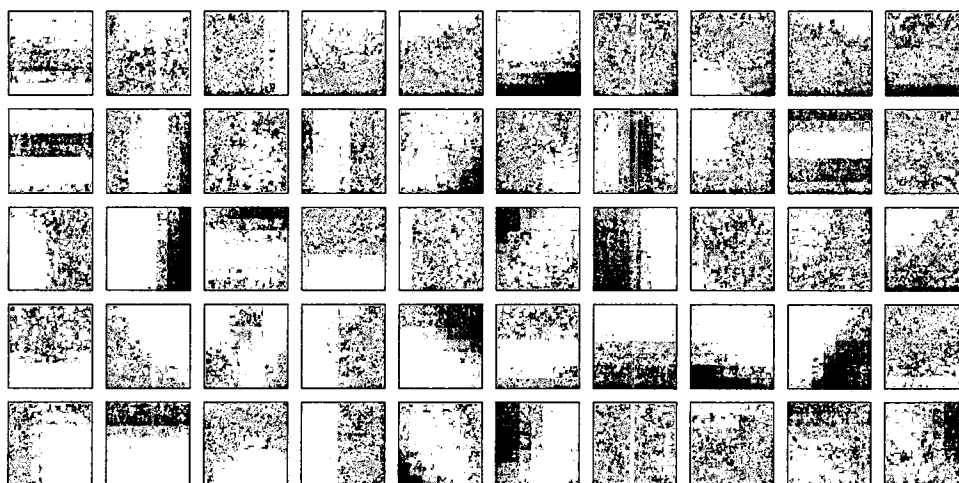


Figure 6.2: The  $M = 50$  codebook.

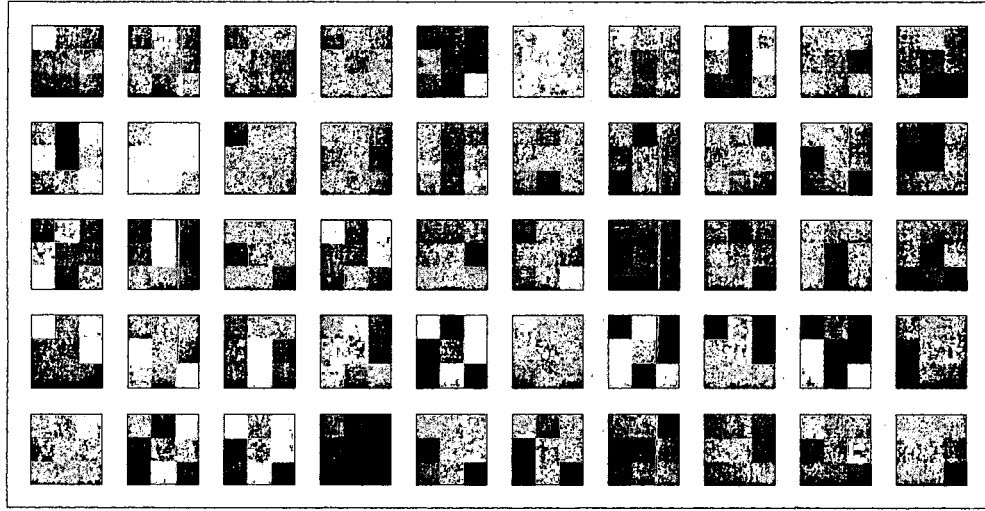


Figure 6.3: The SPWS Center method's  $M = 50$  codebook ( $N = 3 \times 3$ ).

Number selection of codewords is an important task. Depending on the complexity of the images, small partition numbers fit for less complicate images but may lead less improvements. Large partition numbers reflect complicate structure elements better, but it increases the computation complexity. Fig. 6.4 shows 10 sets of  $7 \times 7$  codewords. The observation window size is  $N = 7 \times 7$ , and the partition number  $M = 10$ . The SPWS Center method's 50 sets of  $3 \times 3$  codewords which obtained from the  $7 \times 7$  observation window are shown in Fig. 6.5.

The MSE results for the training process are shown in Table 6.1. The PWS filters MSE results show an 8.6% improvement over the Wiener filter, meanwhile the Soft-PWS filter results show a 15.2% improvement over Wiener filter results and a 7.1% improvement over PWS filters. It also shows that the parameters from PWS filters such as  $\mathbf{w}$  are not suitable to apply directly to the Soft-PWS filters. The  $\mathbf{w}$  parameters have to be updated, otherwise the performance is even worse than the

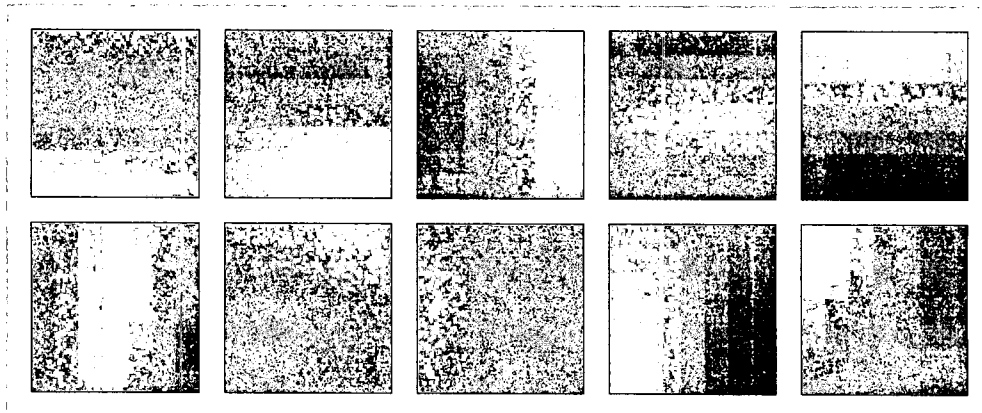


Figure 6.4: The  $M = 10$  codebook.

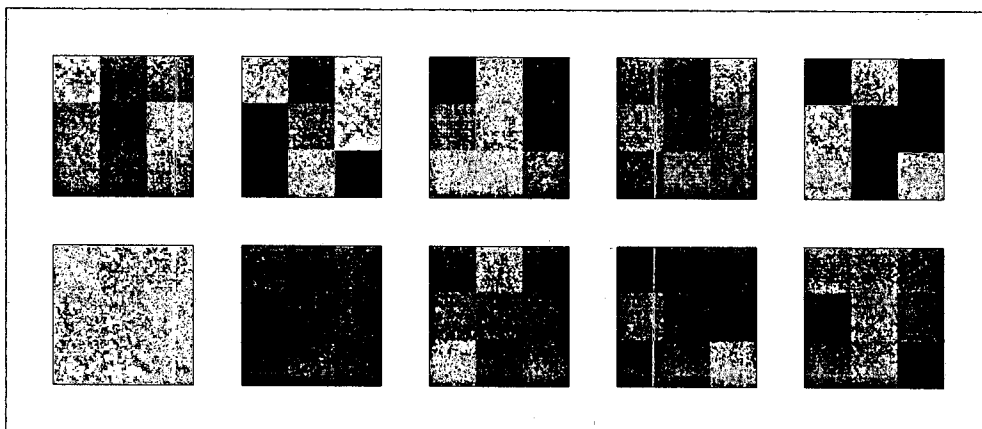


Figure 6.5: The SPWS Center method's  $M = 10$  codebook ( $N = 3 \times 3$ ).

Table 6.1: MSE Results for Training Process.

Method	MSE	Improvement over Wiener	Improvement over PWS
Wiener Filter	275.62	0.0%	-9.4%
PWS	251.82	8.6%	0.0%
Soft-PWS (using PWS's $\mathbf{w}$ , $\mathbf{c}$ and $\mathbf{s}$ )	261.99	4.9%	-4.0%
Soft-PWS (updated $\mathbf{w}$ only)	236.83	14.1%	6.0%
Soft-PWS (updated $\mathbf{w}$ and $\mathbf{c}$ only)	234.25	15.0%	7.0%
Soft-PWS (updated $\mathbf{w}$ , $\mathbf{c}$ and $\mathbf{s}$ )	234.19	15.0%	7.0%
Soft-PWS (updated $\mathbf{w}$ one more times after updated $\mathbf{w}$ , $\mathbf{c}$ and $\mathbf{s}$ )	233.84	15.2%	7.1%

generic PWS filters. Table 6.1 also shows that updating  $\mathbf{w}$  one more time will still reduce the MSE, thus for training purpose we recommend performing the updating  $\mathbf{w}$  again after updating  $\mathbf{w}$ ,  $\mathbf{c}$  and  $\mathbf{s}$ .

### 6.1.2 Aerial Test Images Results

The first 8 bit  $512 \times 512$  test aerial image is shown in Fig. 6.6(a). The noisy aerial image is shown in Fig. 6.6(b). The noisy aerial image is generated from the desired test aerial image by artificially adding white Gaussian noise. The aerial test images are used exclusively for testing purposes which include using quantitative error analysis and visually demonstrating the effectiveness of Soft-PWS filter.

The enlarged portion of the true aerial test image ( $200 \times 200$ ) is shown in Fig. 6.7(a), and the noisy test image is shown in Fig. 6.7(b). Fig. 6.7(c) shows median filter output, and the moving average filter output is shown in Fig. 6.7(d).

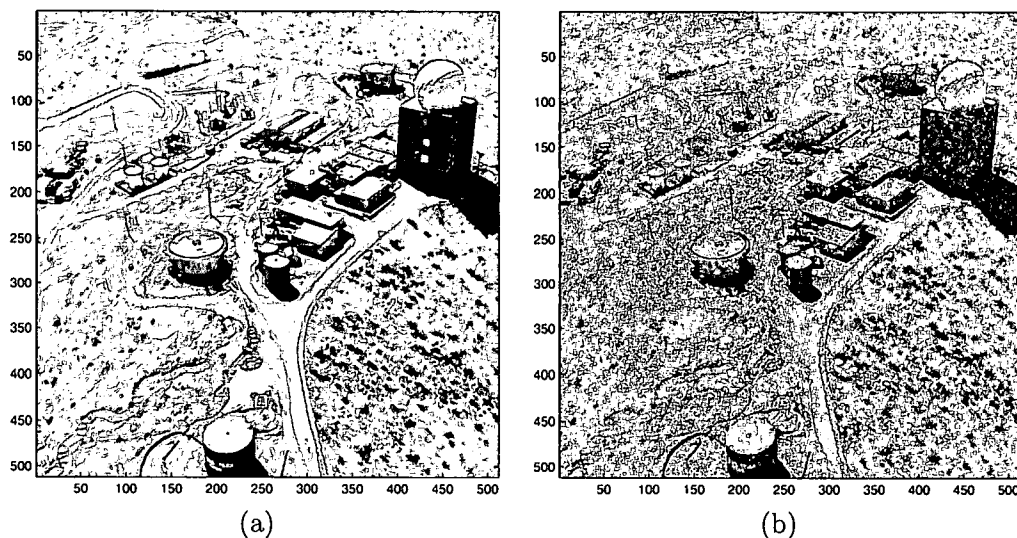


Figure 6.6: (a) The desired test aerial image; and (b) the noisy test aerial image ( $512 \times 512$ ).

Enlarged portions ( $200 \times 200$ ) of the Wiener filter output ( $\text{MSE} = 220.90$ ) is shown in Fig. 6.8(a), the PWS filter output ( $\text{MSE} = 202.79$ ) is shown in Fig. 6.8(b), the Soft-PWS filter output with updated  $\mathbf{w}$  only is shown in Fig. 6.8(c) ( $\text{MSE} = 193.83$ ), and the Soft-PWS filter output with updated  $\mathbf{w}$ ,  $\mathbf{c}$  and  $\mathbf{s}$  is shown in Fig. 6.8(d) ( $\text{MSE} = 192.69$ ). Note that in all of the above results, the observation window size is  $N = 5 \times 5$ , and the partition number is  $M = 50$ . Table 6.2 shows the MSE results for the aerial test image. Figs. 6.7, and 6.8 and Table 6.2 show that the median filters better preserve image details (ex: edge information) but do not show much improvements over the Wiener filters. Figs. 6.7, and 6.8 and Table 6.2 also show that the moving average filter smooths out both noise and details. The results show a reduction in MSE. The MSE results from the PWS filters show 8.2% improvement over the Wiener filter, meanwhile the Soft-PWS filter results show a

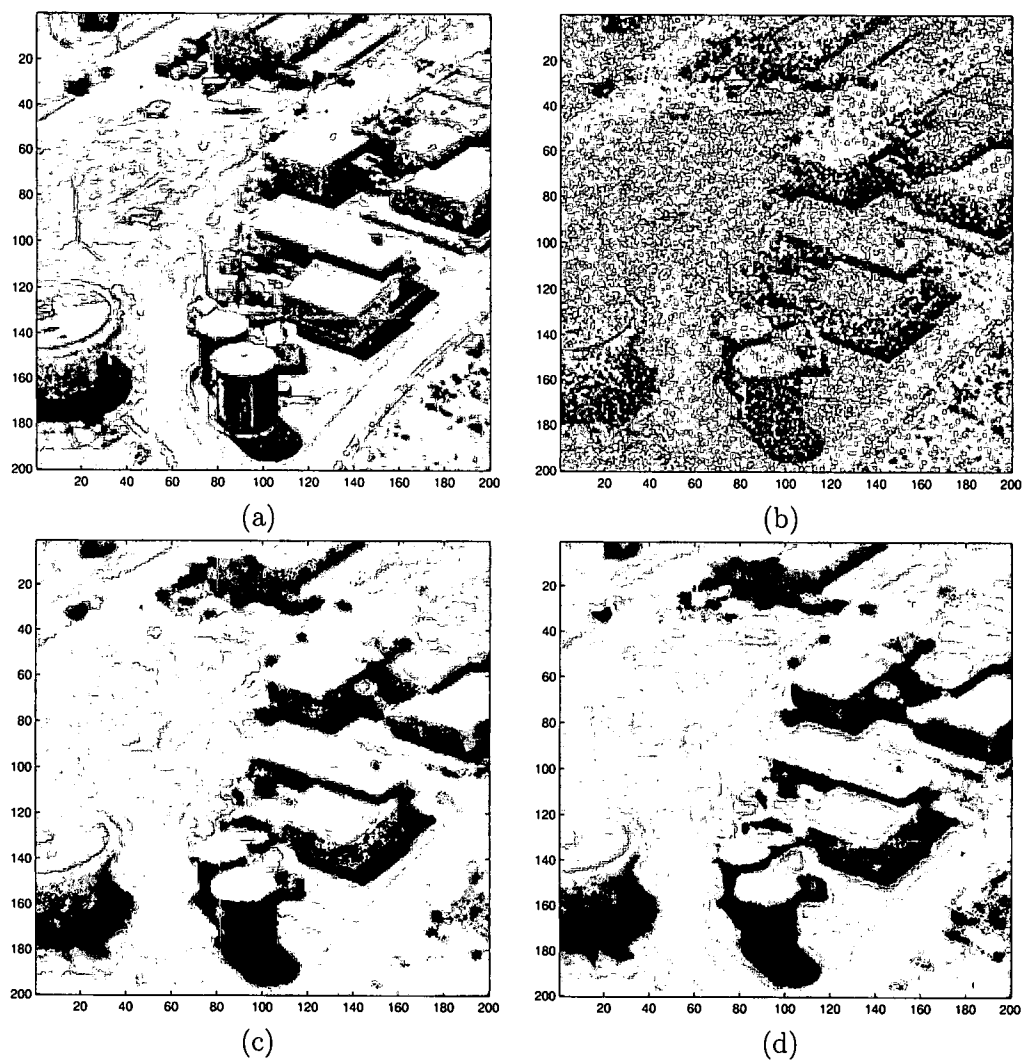


Figure 6.7: Enlarged portions ( $200 \times 200$ ) of (a) the desired test aerial image; (b) the noisy test aerial image; (c) median filter output; and (d) moving average filter output ( $N = 5 \times 5$ ).

Table 6.2: MSE Results for Test Aerial Image.

Method	MSE	Improvement over Wiener	Improvement over PWS
Wiener Filter	220.90	0.0%	-8.9%
PWS	202.79	8.2%	0.0%
Soft-PWS (updated $w$ only)	193.83	12.3%	4.4%
Soft-PWS (updated $w$ and $c$ only)	192.74	12.7%	5.0%
Soft-PWS (updated $w$ , $c$ and $s$ )	192.69	12.8%	5.0%
Soft-PWS (updated $w$ one more times after updated $w$ , $c$ and $s$ )	193.17	12.6%	4.7%

12.8% improvement over the Wiener filter results and a 5.0% improvement over the PWS filters. The experimental results also show that updating  $w$  one more time after updating  $w$ ,  $c$  and  $s$  sometimes results in tuning too much towards the training images and may not fit for other type of images. In the aerial image case, the MSE from updating one more time is 193.17 vs. 192.69 – the MSE of updating  $w$ ,  $c$  and  $s$ . Thus from testing purposes, we recommend to update only  $w$ ,  $c$  and  $s$ .

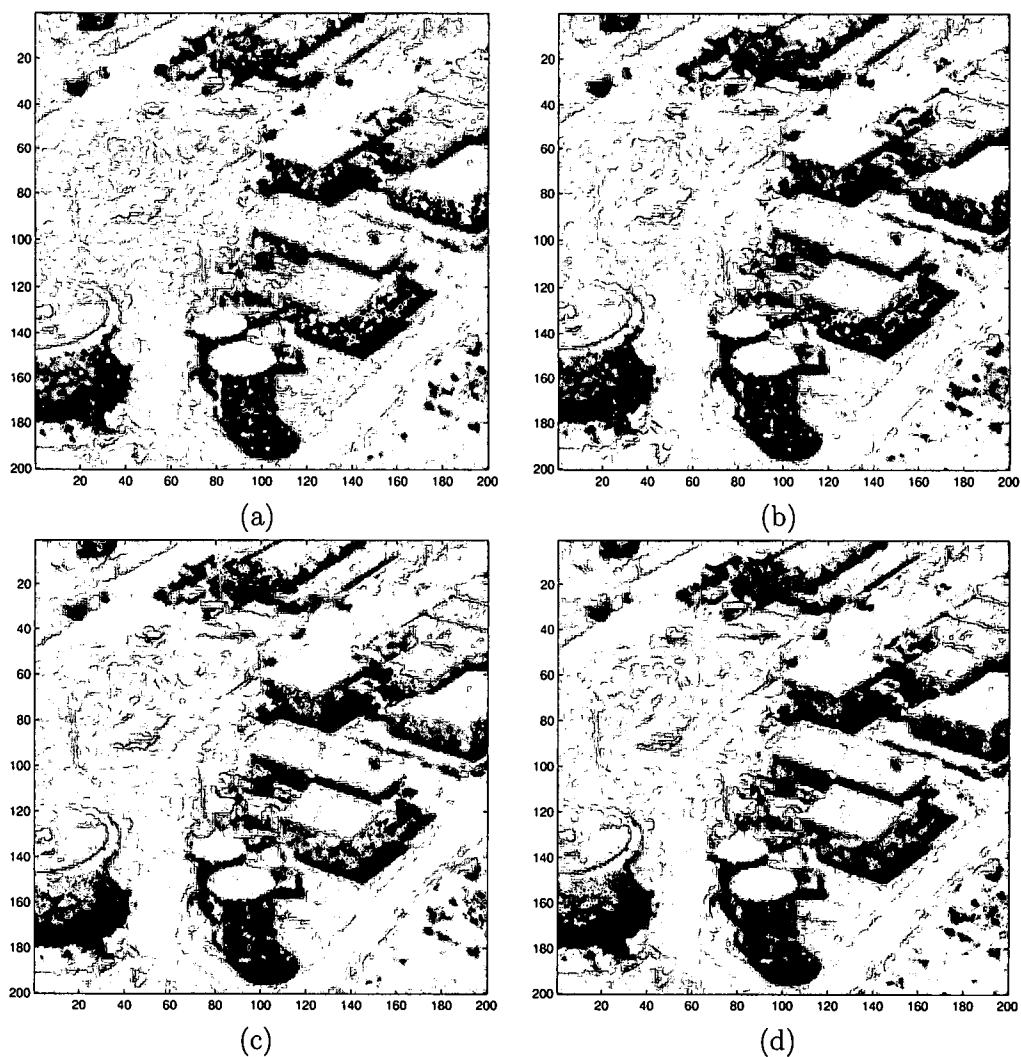


Figure 6.8: Enlarged portions (200 × 200) of (a) Wiener filter output (MSE = 220.90); (b) PWS filter output (MSE = 202.79); (c) Soft-PWS filter output with updated  $w$  only (MSE = 193.83); (d) Soft-PWS filter output with updated  $w, c$  and  $s$  (MSE = 192.69,  $N = 5 \times 5$ ,  $M = 50$ ).

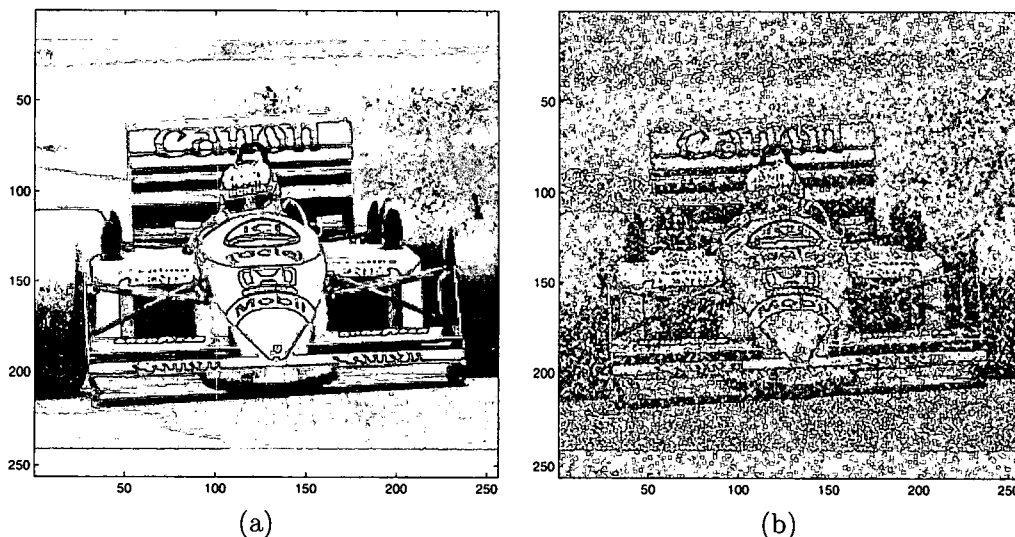


Figure 6.9: (a) The true test car image; and (c) the noisy test car image ( $256 \times 256$ ).

### 6.1.3 Car Test Image Results

The second test image, an 8 bit  $256 \times 256$  car image, is shown in Fig. 6.9(a). The noisy test car image is shown in Fig. 6.9(b). The noisy car image is generated from the desired test car image by artificially adding the white Gaussian noise. The test car images are also used exclusively for testing purposes which include using for quantitative error analysis and visual demonstration of the effectiveness of the Soft-PWS filter.

An enlarged portion of the true car test image ( $150 \times 150$ ) is shown in Fig. 6.10(a), and the noisy test image is shown in Fig. 6.10(b). Fig. 6.10(c) shows the median filter output, and the moving average filter output is shown in Fig. 6.10(d). An enlarged portion ( $150 \times 150$ ) of the Wiener Filter output ( $MSE = 234.72$ ) is shown in Fig. 6.11(a), the PWS filter output ( $MSE = 190.97$ ) is shown in Fig. 6.11(b), the



Figure 6.10: Enlarged center portion ( $150 \times 150$ ) of (a) true test car image; (b) noisy test car image; (c) median filter output; and (d) moving average filter output ( $N = 5 \times 5$ ).

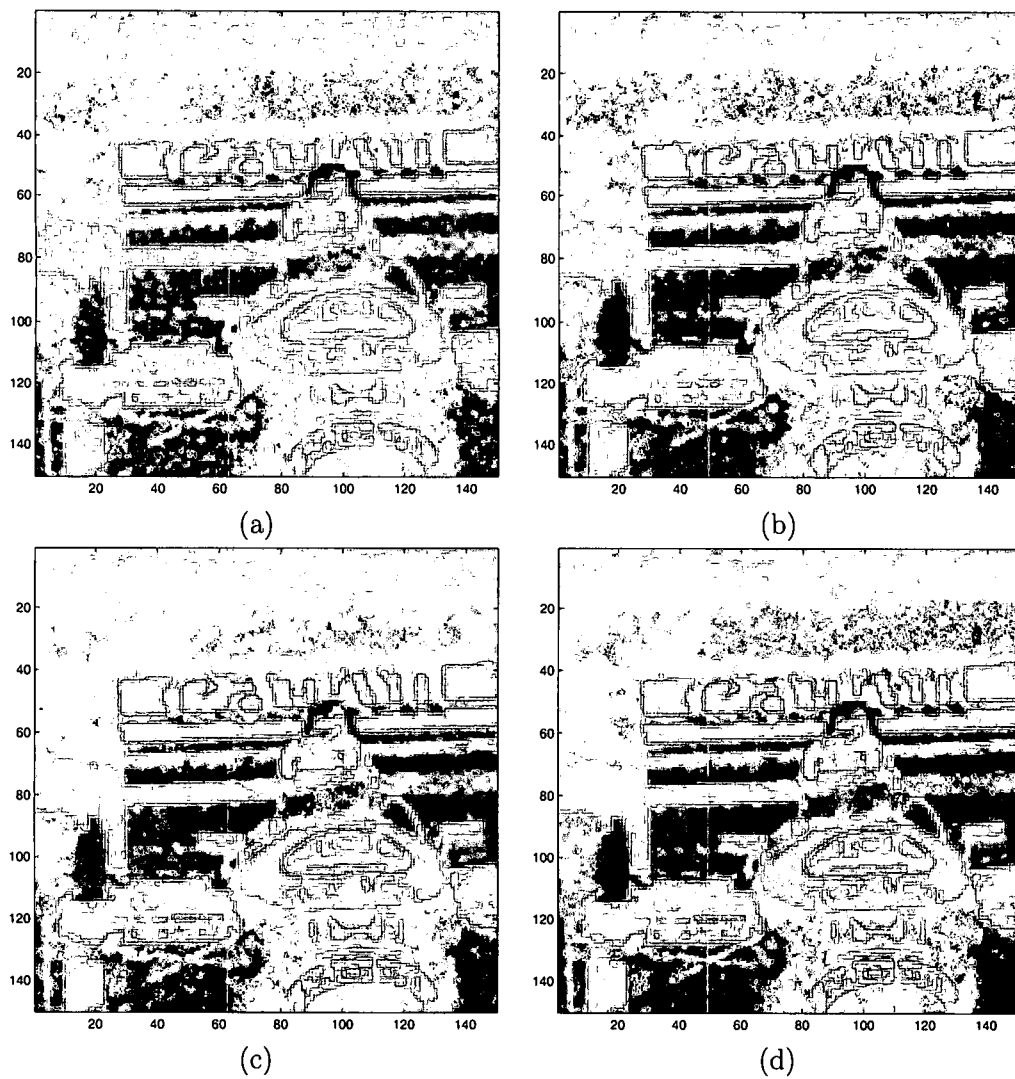


Figure 6.11: Enlarged center portion ( $150 \times 150$ ) of (a) Wiener filter output, (b) PWS filter output; (c) Soft-PWS filter output with updated  $w$  only, and (d) Soft-PWS filter output with updated  $w$ ,  $c$  and  $s$  ( $N = 5 \times 5$ ,  $M = 50$ ).

Table 6.3: MSE Results for Car Test Images.

Method	MSE	Improvement over Wiener	Improvement over PWS
Wiener Filter	234.72	0.0%	-22.9%
PWS	190.97	18.6%	0.0%
Soft-PWS (updated $\mathbf{w}$ only)	166.44	29.1%	12.8%
Soft-PWS (updated $\mathbf{w}$ and $\mathbf{c}$ only)	164.32	30.0%	14.0%
Soft-PWS (updated $\mathbf{w}$ , $\mathbf{c}$ and $\mathbf{s}$ )	163.90	30.2%	14.2%
Soft-PWS (updated $\mathbf{w}$ one more times after updated $\mathbf{w}$ , $\mathbf{c}$ and $\mathbf{s}$ )	163.44	30.4%	14.4%

Soft-PWS filter output with updated  $\mathbf{w}$  only is shown in Fig. 6.11(c) (MSE = 166.44), and the Soft-PWS filter output with updated  $\mathbf{w}$ ,  $\mathbf{c}$  and  $\mathbf{s}$  is shown in Fig. 6.11(d) (MSE = 163.90). Note that in all of the above results, the observation window size is  $N = 5 \times 5$ , and the partition number is  $M = 50$ . Table 6.3 shows the MSE results for the car test image. Figs. 6.10, and 6.11 and Table 6.3 show the MSE reduction. The generic PWS filters MSE results show an 18.6% improvement over the Wiener filter, meanwhile the Soft-PWS filter results show a 30.2% improvement over the Wiener filter results and a 14.4% improvement over the PWS filters. The experimental results also show that updating  $\mathbf{w}$ ,  $\mathbf{c}$  and  $\mathbf{s}$  did not significantly improve performance as opposed to updating  $\mathbf{w}$  only. In other words, we recommend to perform only updating  $\mathbf{w}$  to reduce computation complex by giving up a little performance.

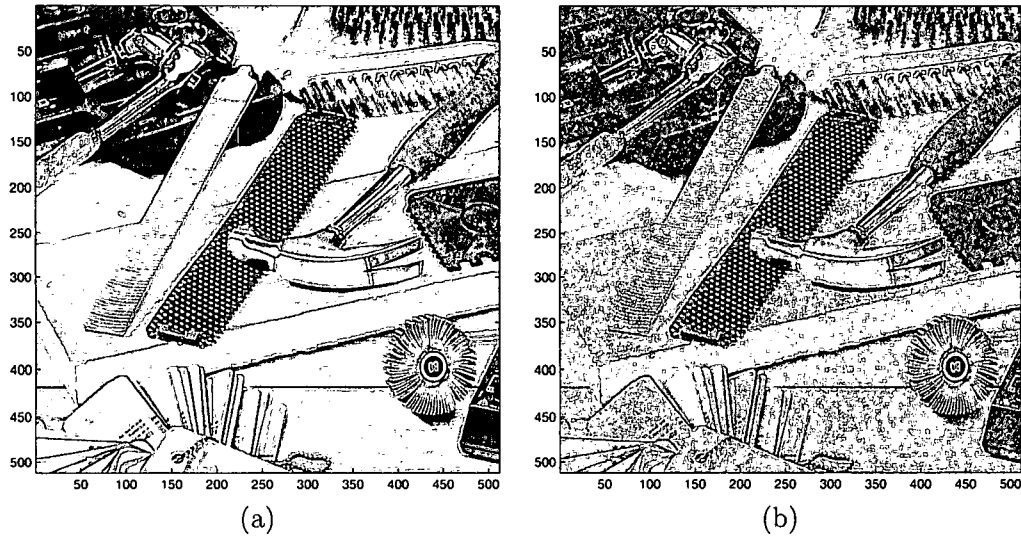


Figure 6.12: (a) The true test tool image; and (c) the noisy tool image ( $512 \times 512$ ).

#### 6.1.4 Tools Test Image Results

The third test image is an 8 bit  $512 \times 512$  tools image shown in Fig. 6.12(a). The noisy tools image is shown in Fig. 6.12(b). The noisy tools image is generated from the desired test tools image by artificially adding white Gaussian noise. The test tools image is also used exclusively for testing purposes.

The enlarged center portion of the true test tools image ( $150 \times 150$ ) is shown in Fig. 6.13(a), and the noisy test image is shown in Fig. 6.13(b). Fig. 6.13(c) shows the median filter output, and the moving average filter output is shown in Fig. 6.13(d). Enlarged center portions ( $150 \times 150$ ) of the Wiener filter output ( $\text{MSE} = 412.86$ ) is shown in Fig. 6.14(a), the generic PWS filter output ( $\text{MSE} = 328.65$ ) is shown in Fig. 6.14(b), the Soft-PWS filter output with updated  $\mathbf{w}$  only is shown in Fig. 6.14(c) ( $\text{MSE} = 271.95$ ), and the Soft-PWS filter output with updated  $\mathbf{w}$ ,  $\mathbf{c}$  and  $\mathbf{s}$  is shown

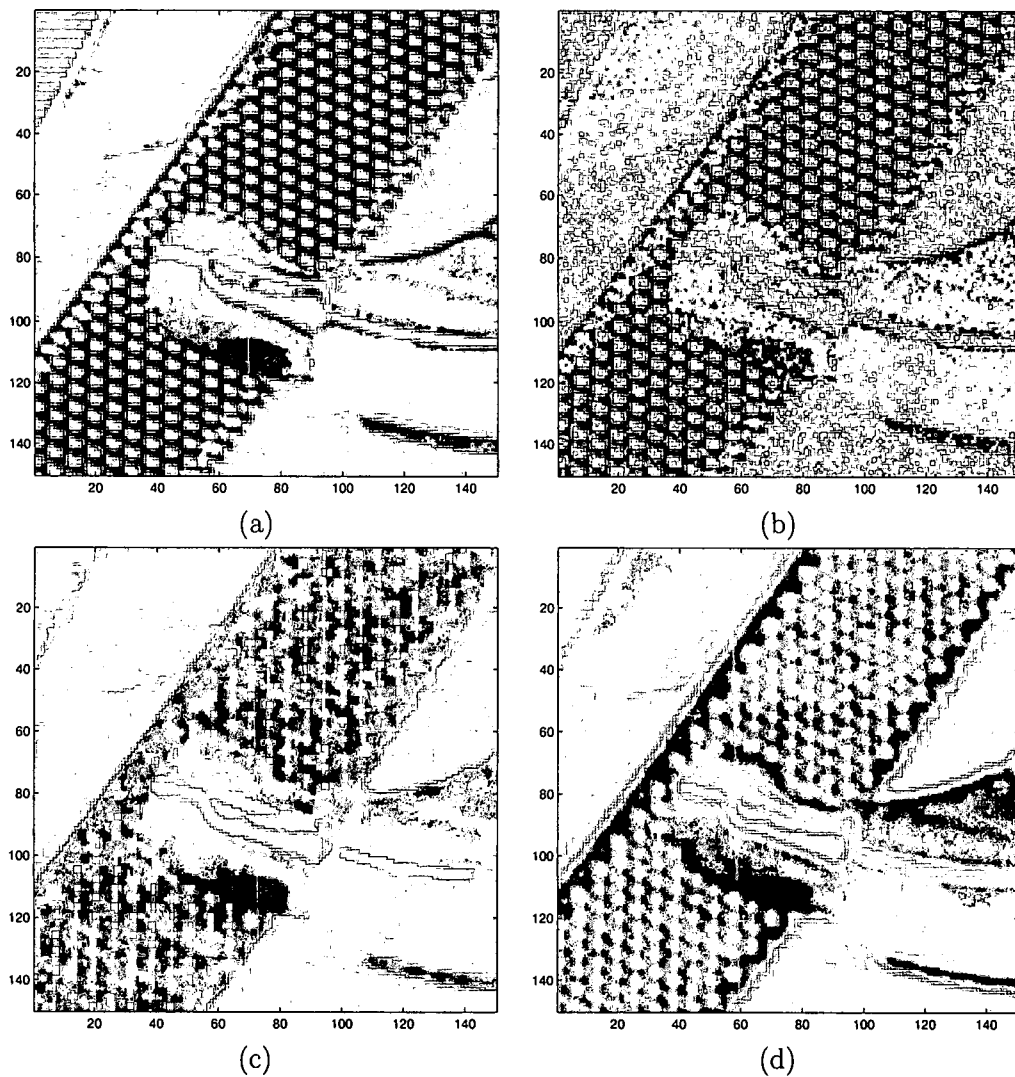


Figure 6.13: Enlarged center portion ( $150 \times 150$ ) of (a) true test tool image; (b) noisy test tool image; (c) median filter output; and (d) moving average filter output ( $N = 5 \times 5$ ).

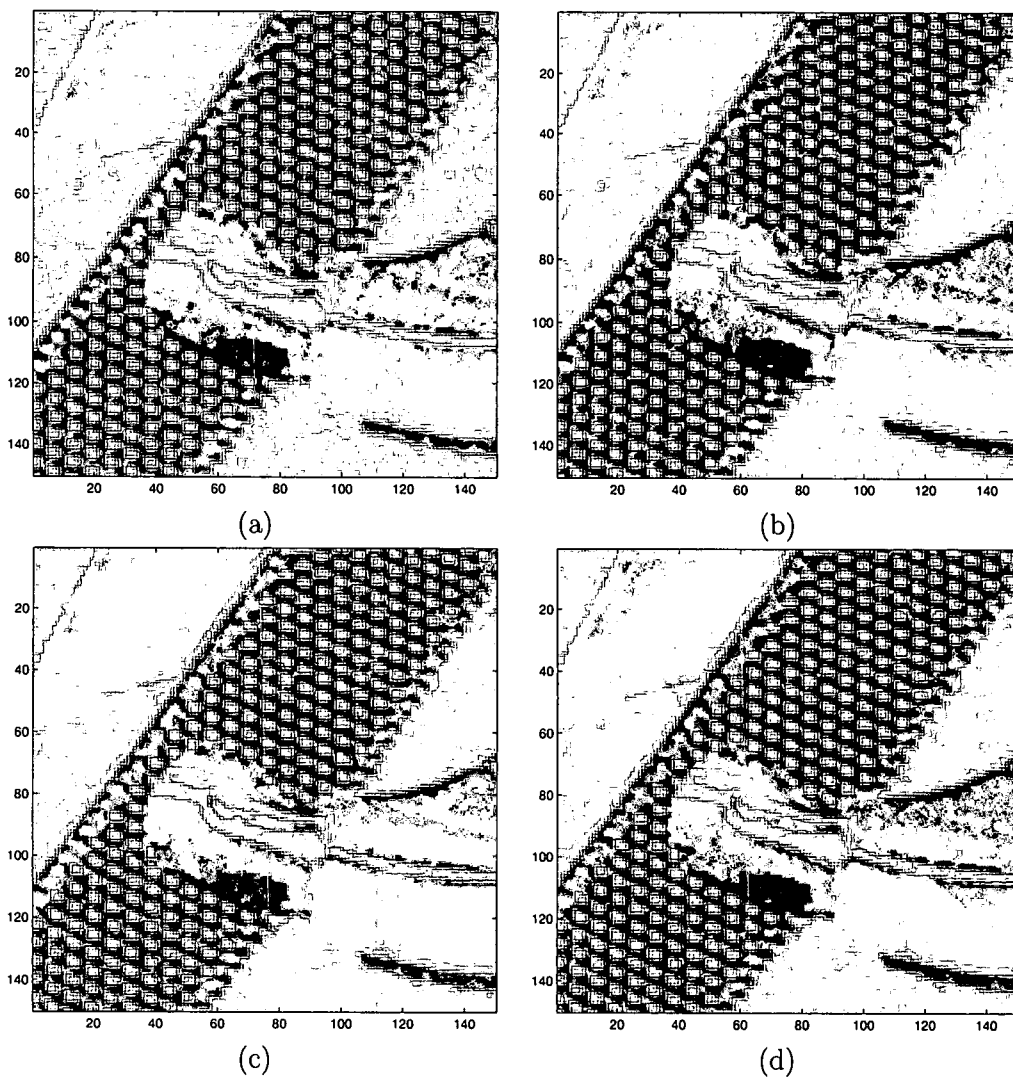


Figure 6.14: Enlarged center portion ( $150 \times 150$ ) of (a) Wiener filter output, (b) PWS filter output; (c) Soft-PWS filter output with updated  $w$  only, and (d) Soft-PWS filter output with updated  $w, c$  and  $s$  ( $N = 5 \times 5$ ,  $M = 50$ ).

Table 6.4: MSE Results for Tool Test Images.

Method	MSE	Improvement over Wiener	Improvement over PWS
Wiener Filter	412.86	0.0%	-25.6%
PWS	328.65	20.4%	0.0%
Soft-PWS (updated $\mathbf{w}$ only)	271.95	34.1%	17.3%
Soft-PWS (updated $\mathbf{w}$ and $\mathbf{c}$ only)	267.92	35.1%	18.5%
Soft-PWS (updated $\mathbf{w}$ , $\mathbf{c}$ and $\mathbf{s}$ )	267.83	35.1%	18.5%
Soft-PWS (updated $\mathbf{w}$ one more times after updated $\mathbf{w}$ , $\mathbf{c}$ and $\mathbf{s}$ )	275.62	33.2%	16.1%

in Fig. 6.14(d) ( $\text{MSE} = 267.83$ ). Note that in all of the above results, the observation window size is  $N = 5 \times 5$ , and the partition number is  $M = 50$ . Table 6.4 shows the MSE results for tool test image. Figs. 6.13, and 6.14 and Table 6.4 show the MSE reduced as promised. The generic PWS filters MSE results show 20.4% improvement over the Wiener filter, meanwhile the Soft-PWS filter results show 35.1% improvement over the Wiener filter results and 18.5% improvement over the PWS filter.

### 6.1.5 Building Test Image Results

The fourth test image is an 8 bit  $512 \times 512$  building image shown in Fig. 6.15(a). The noisy building image is shown in Fig. 6.15(b). The noisy building image is generated from the true test buildings image by artificially adding white Gaussian noise. The test building image is also used exclusively for testing purposes.

The enlarged center portion of the true buildings test image ( $150 \times 150$ ) is shown in Fig. 6.16(a), and the noisy test image is shown in Fig. 6.16(b). Fig. 6.16(c)

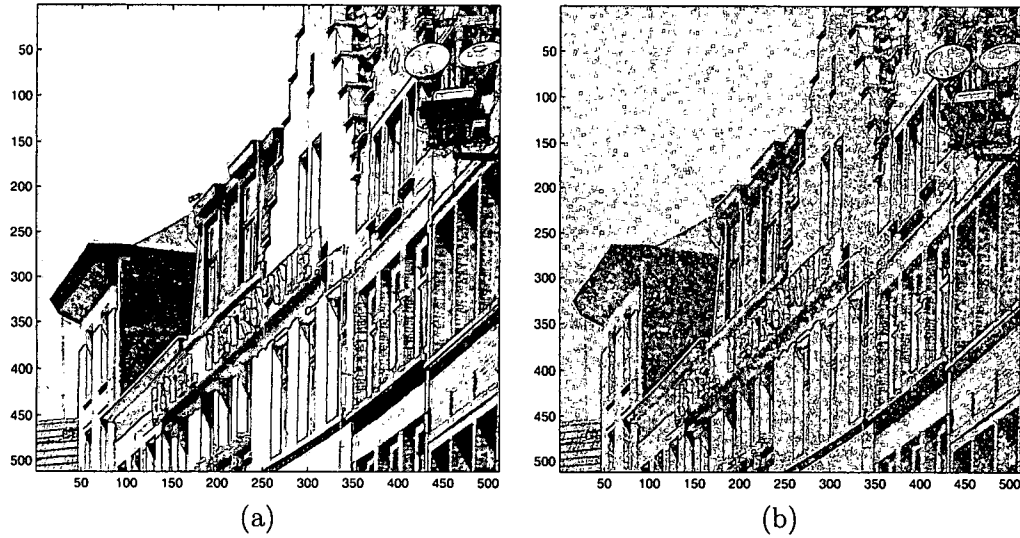


Figure 6.15: (a) The true test building image, and (b) the noisy building image ( $512 \times 512$ ).

Table 6.5: MSE Results for Building Test Images.

Method	MSE	Improvement over Wiener	Improvement over PWS
Wiener filter	326.98	0.0%	-28.5%
PWS	254.36	22.2%	0.0%
Soft-PWS (updated $w$ only)	223.08	31.8%	12.3%
Soft-PWS (updated $w$ and $c$ only)	219.34	32.9%	13.8%
Soft-PWS (updated $w$ , $c$ and $s$ )	218.95	33.0%	13.9%
Soft-PWS (updated $w$ one more times after updated $w$ , $c$ and $s$ )	221.05	32.4%	13.1%

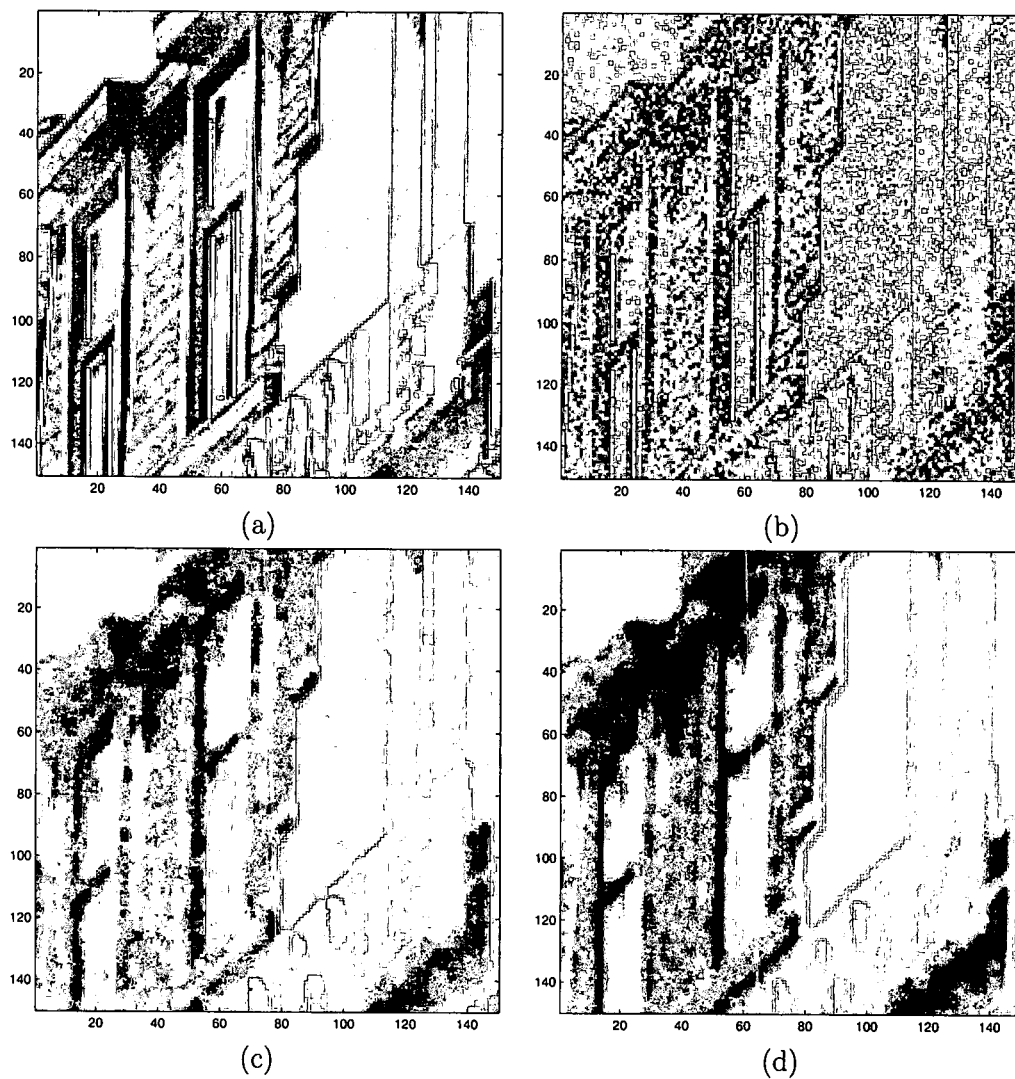


Figure 6.16: Enlarged center portion ( $150 \times 150$ ) of (a) true test building image; (b) noisy test building image; (c) median filter output; and (d) moving average filter output ( $N = 5 \times 5$ ).

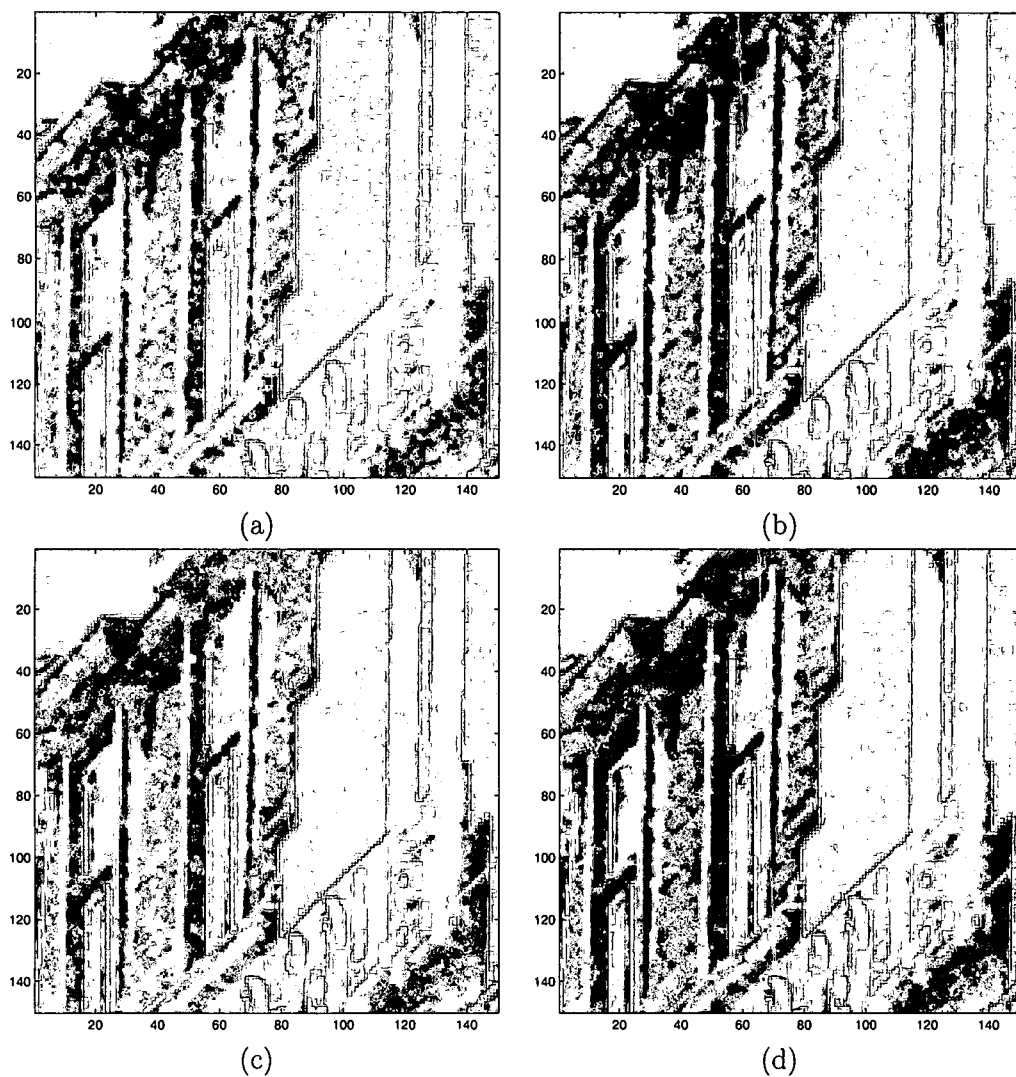


Figure 6.17: Enlarged center portion ( $150 \times 150$ ) of (a) Wiener Filter output, (b) PWS filter output; (c) Soft-PWS filter output with updated  $w$  only, and (d) Soft-PWS filter output with updated  $w$ ,  $c$  and  $s$  ( $N = 5 \times 5$ ,  $M = 50$ ).

shows the median filter output, and the moving average filter output is shown in Fig. 6.16(d). Enlarged center portions ( $150 \times 150$ ) of the Wiener filter output ( $\text{MSE} = 326.98$ ) is shown in Fig. 6.17(a), the generic PWS filter output ( $\text{MSE} = 254.36$ ) is shown in Fig. 6.17(b), the Soft-PWS filter output with updated  $\mathbf{w}$  only is shown in Fig. 6.17(c) ( $\text{MSE} = 223.08$ ), and the Soft-PWS filter output with updated  $\mathbf{w}$ ,  $\mathbf{c}$  and  $\mathbf{s}$  is shown in Fig. 6.17(d) ( $\text{MSE} = 218.95$ ). Note that in all of the above results, the observation window size is  $N = 5 \times 5$ , and the partition number is  $M = 50$ . Table 6.5 shows the MSE results for building test images. Figs. 6.16, and 6.17 and Table 6.5 show the reduction of MSE. The PWS filters MSE results show a 22.2% improvement over the Wiener filter, meanwhile the results from the Soft-PWS filter show a 33.0% improvement over the Wiener filter results and a 13.9% improvement over the PWS filters.

## 6.2 Biomedical Images Results

In this section, the performance of the Soft-PWS filter applied to the biomedical two dimensional (2D) and three dimensional (3D) CT results of image noise reduction applications is presented and compared with that of the commonly-used image noise reduction linear filters: the FIR Wiener filters, the median filter and the moving average filters, and the generic PWS non-linear filters. Automatically detecting and segmenting nodules presented in CT images is an important and timely problem. The motivation of performing noise reduction to CT images by the Soft-PWS filter is that we hope this type of processing could aid the automatic detecting and segmenting process.

The CT images used in this section are thoracic images from the iCAD, Inc (Beavercreek, Ohio) database. There are three sets of 3D CT images from different patients, and each one have lung nodules presented. Each set has 128 slices of CT images which are reconstructed at  $512 \times 512$ . The first two sets are high dose CT images. The third set are low dose CT images in which noise is present. The first set of images are used exclusively for training purposes to generate Soft-PWS filter partitions/codebook and Soft-PWS filter coefficients for each partition, and to perform optimization process. The second and the third set of images are used exclusively for testing purposes. The second set of images are used for quantitative error analysis, and the third set of images, the noisy images, are used for visual demonstration of the effectiveness of the Soft-PWS filters. One slice was selected from each set of images. The selected training CT image was artificially degraded to simulate the noise level of the test image from the third set of images. The training image from the first patient's CT image set is shown in Fig. 6.18.

### **6.2.1 Two Dimensional CT Images Results**

The 2D test image from the second patient used for quantitative error analysis is shown in Fig. 6.19(a). The MSE results are shown in Table 6.6. The MSE results show that the Soft-PWS filters perform better than median filter, moving average filter, FIR Wiener filter and traditional PWS filter. The 4.2% improvement over the Wiener filter results is preserved for the Soft-PWS filter results, and also a 1.4% improvement over the traditional PWS filter. The results demonstrated the effectiveness of the Soft-PWS filter.

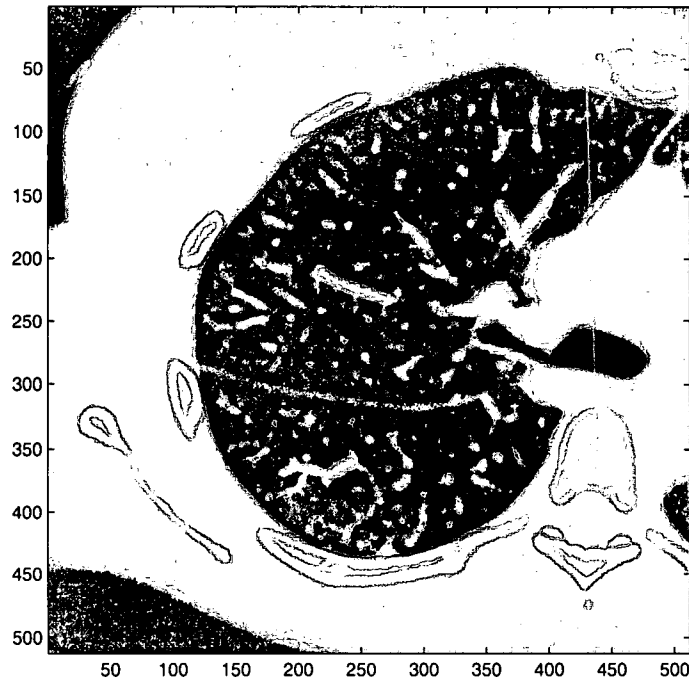


Figure 6.18: The true training CT image from the first patient CT images.

Table 6.6: MSE Results for 2D images.

Method	MSE	Improvement over Wiener	Improvement over PWS
Median Filter	5752	-34.3%	-38.2%
Moving Average Filter	4490	-4.8%	-7.9%
Wiener Filter	4283	0.0%	-2.9%
PWS	4162	2.8%	0.0%
Soft-PWS (updated $w$ only)	4124	3.7%	0.9%
Soft-PWS (updated $w$ and $c$ only)	4103	4.2%	1.4%
Soft-PWS (updated $w$ , $c$ and $s$ )	4103	4.2%	1.4%
Soft-PWS (updated $w$ one more times after updated $w$ , $c$ and $s$ )	4103	4.2%	1.4%

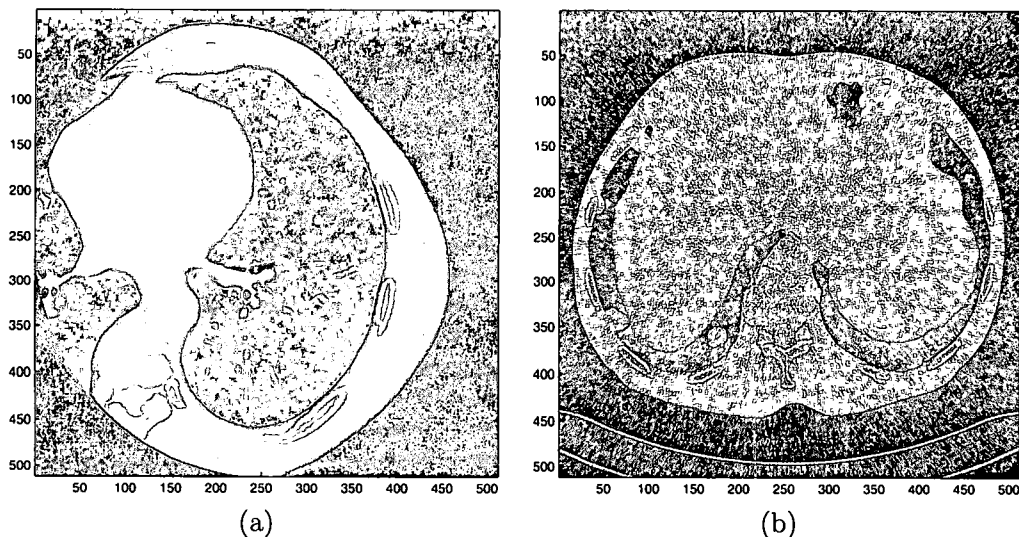


Figure 6.19: (a) The desired test CT image from the second patient; and (b) the test CT image from the third patient ( $512 \times 512$ ).

The test image from the third patient is shown in Fig. 6.19(b). The enlarged center portions of test image of the third patient ( $100 \times 100$ ) are shown in Fig. 6.20(a), and Fig. 6.20(b) shows the Wiener filter output. The median filter output is shown in Fig. 6.20(c), and the moving average filter output is shown in Fig. 6.20(d). The traditional PWS filter output is shown in Fig. 6.21(a), and the Soft-PWS filter output with updated  $w$ ,  $c$  and  $s$  is shown in Fig. 6.21(b). Note that in all of the above results, the observation window size is  $N = 5 \times 5$ , and the partition number is  $M = 50$ . A comparison of the results shows that the Soft-PWS filter output is more clear and suitable for performing further segmentations. In the other words, the results show that the Soft-PWS filter results are the most promising. This demonstrates the point we discussed in previous chapters. We also compare the Soft-PWS filter with median

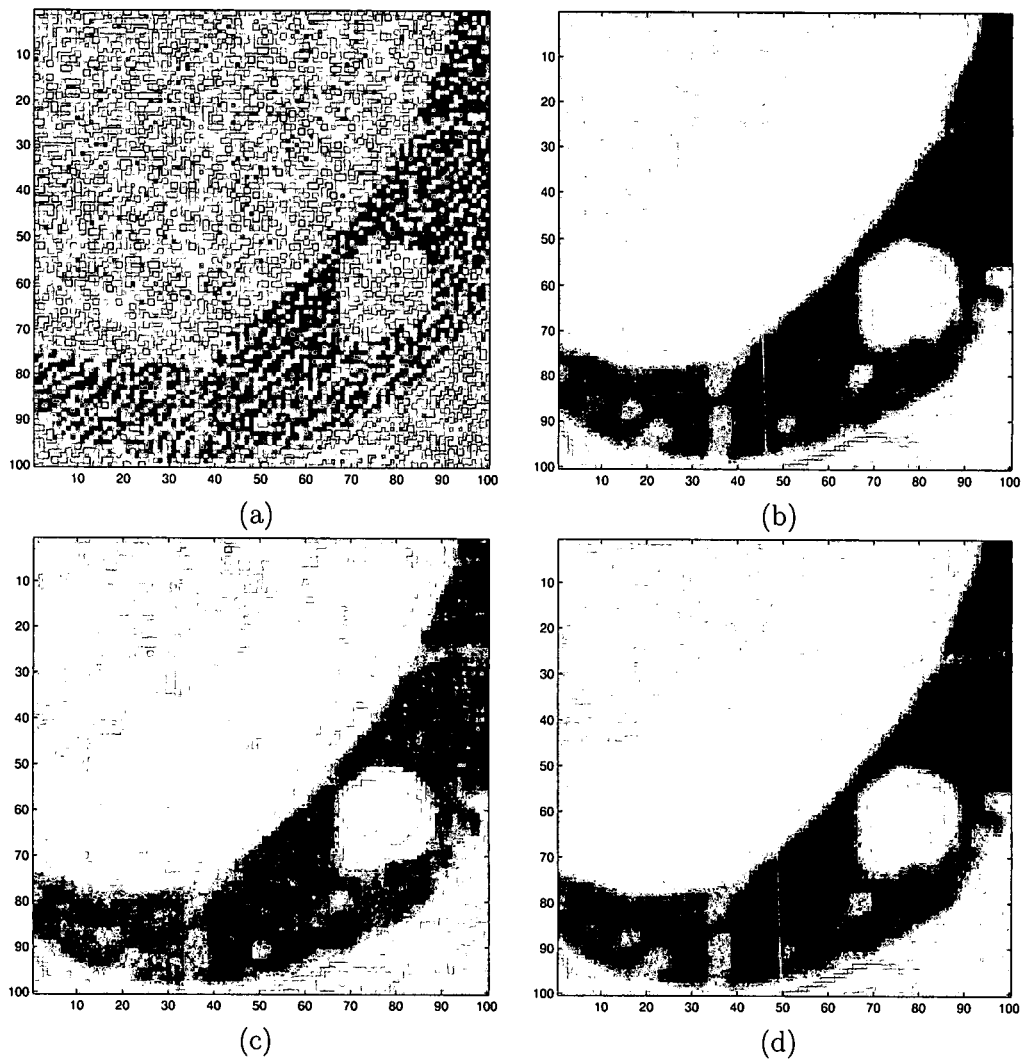


Figure 6.20: Enlarged center portions ( $100 \times 100$ ) of (a) test CT image; (b) Wiener Filter output; (c) median filter output; and (d) moving average filter output ( $N = 5 \times 5$ ).

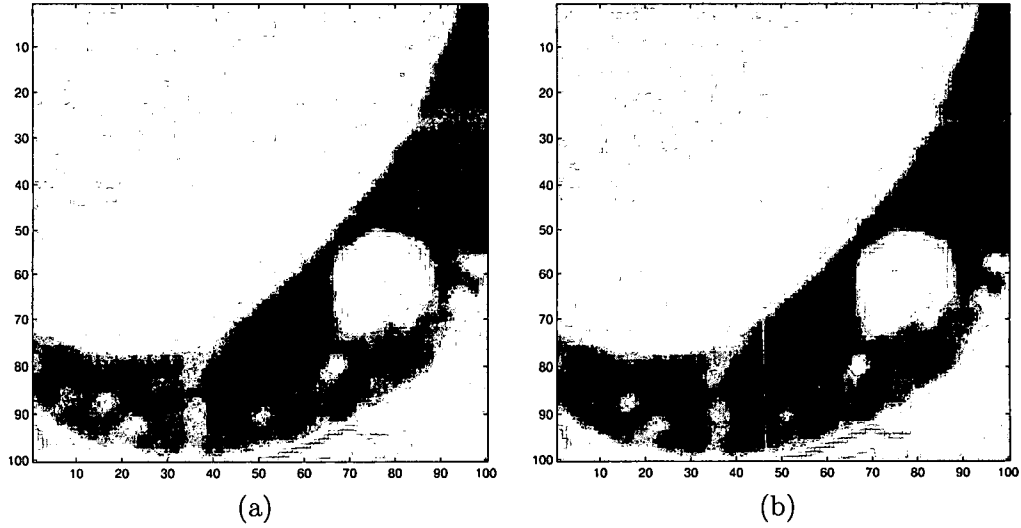


Figure 6.21: Enlarged center portions ( $100 \times 100$ ) of (a) 2D PWS filter output; and (b) 2D Soft-PWS filter output with updated  $\mathbf{w}$ ,  $\mathbf{c}$  and  $\mathbf{s}$  ( $N = 5 \times 5$ ,  $M = 50$ ).

filter and the moving average filter. The results show that the Soft-PWS filter results are much better than moving average and median filters.

From the quantitative error analysis results in Table 6.6 and results shown in Figs. 6.20 and 6.21, we have observed that updating for  $\mathbf{s}$  will not lead to significant quantitative and visual improvements. We believe the original variance -  $\mathbf{s}$  - is good enough. It may not be necessary to perform optimization of the variance. By determining this, we can reduce the computation complexity of the whole optimization process. Also we have observed that the second round of updating  $\mathbf{w}$  showed only slight improvements during training process, but not always have any improvements when applying to a new test image. In conclusion, we believe only one update for  $\mathbf{w}$  and  $\mathbf{c}$  is essential. To reduce the computational complexity even further but still keep quite close performance by updating both  $\mathbf{w}$  and  $\mathbf{c}$ , we recommend performing

Table 6.7: MSE Results for 3D Images.

Method	MSE	Improvement over Wiener	Improvement over PWS	Improvement over 2D
Wiener Filter	3148	0.0%	-10.3%	26.5%
PWS	2854	9.3%	0.0%	31.4%
Soft-PWS (updated $w$ only)	2694	14.4%	5.6%	34.7%
Soft-PWS (updated $w$ and $c$ only)	2679	14.9%	6.1%	34.7%
Soft-PWS (updated $w$ , $c$ and $s$ )	2679	14.9%	6.1%	34.7%

only closed form solution for  $w$ . Performing only close form optimization for  $w$ , can obtain very close performance by updating all three parameters. The computational expense for this would be.

### 6.2.2 Three Dimensional CT Images Results

In the case of 3D images, we selected three slices from the patient's image set associated with the 2D images selected. The second/middle slice is exactly as the same as the 2D case, the other two slices are one before and one after the 2D slice. The observation window is a 3D  $N = 5 \times 5 \times 3$  window. A  $N = 5 \times 5$  window from each slice centered at the observation pixel was selected and used to form the new 3D observation window. The observation window size now becomes  $N = 5 \times 5 \times 3$ . We still used  $M = 50$  as our partition number.

We performed 3D image noise reduction to the three slice set we selected from the second and third patients' CT images. As in the VQ partition method, we can use standard method as well as the PCA or Center method proposed elsewhere

Table 6.8: MSE Improvements of 3D Images from 2D Images.

Method	2D MSE	3D MSE	Improvement
PWS	4162	2854	31.4%
Soft-PWS (updated $w$ , $c$ and $s$ )	4103	2679	34.7%

<sup>39,40</sup> In this case, we used the center for the VQ partition process, since the Center method has characteristic high performance and less computation expenses. The middle slice's  $N = 5 \times 5$  observation window was selected as the new observation windows at the Center method to perform VQ partition and reduce the computation complexities. This process not only reduced the computational complexity, but also improved performance.<sup>39,40</sup> The true images are shown in Figs. 6.18 and 6.19.

The important image results which demonstrate the performance of the 3D Soft-PWS filters are shown in Fig. 6.22. The enlarged portions ( $100 \times 100$ ) of the 3D Wiener filter output and the 3D PWS filter output are shown in Fig. 6.22(a) and (b), and the 3D Soft-PWS filter output and 2D Soft-PWS filter output with updated  $w$ ,  $c$  and  $s$  are shown in Fig. 6.22(c) and (d). Note that in all results, the observation window size is  $N = 5 \times 5 \times 3$ ,  $L = 5 \times 5$ , and the partition number is  $M = 50$ .

The MSE results from the second patient's CT lung image set are shown in Table 6.7. The MSE results show that the Soft-PWS filter performs significantly better than the FIR Wiener filter and traditional PWS filter. The 14.9% improvement over the 3D Wiener filter results is preserved in the Soft-PWS filter results, while showing a 6.1% improvement over the traditional 3D PWS filter. The 3D improvement

results over 2D PWS and Soft-PWS filters are shown at Table 6.8. The MSE results showed an amazing 31.4% improvement compared with 2D PWS results and a 34.7% improvement over 2D Soft-PWS filters results. The results demonstrated the effectiveness of the 3D Soft-PWS filter and also demonstrated that the other two slices contained some important information used in the reconstruction process.

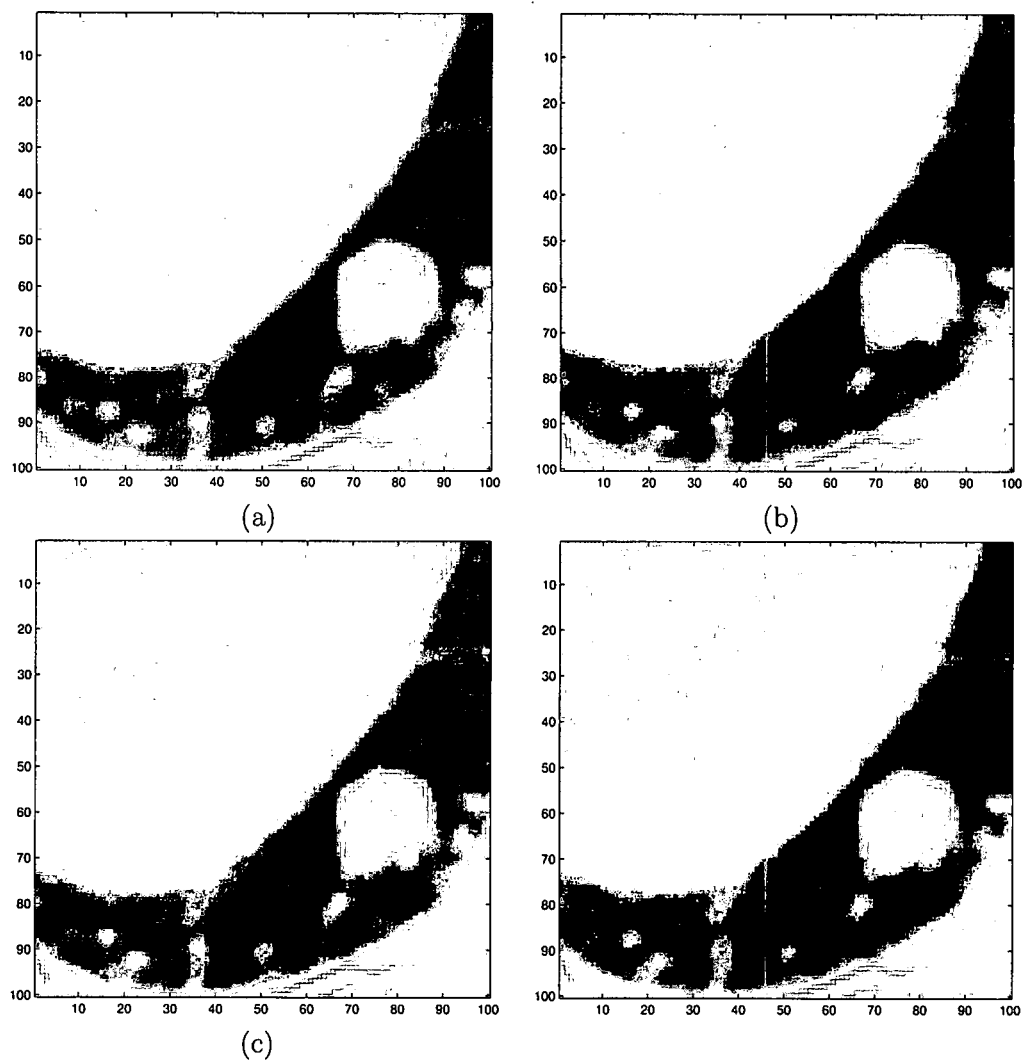


Figure 6.22: Enlarged center portion ( $100 \times 100$ ) of (a) 3D Wiener filter output, (b) 3D PWS filter output, (c) 3D Soft-PWS filter output with updated  $\mathbf{w}$ ,  $\mathbf{c}$  and  $\mathbf{s}$ , and (d) 2D Soft-PWS filter output with updated  $\mathbf{w}$ ,  $\mathbf{c}$  and  $\mathbf{s}$  ( $N = 5 \times 5 \times 3$ ,  $L = 5 \times 5$ ,  $M = 50$ ).

### 6.3 Summary

In this chapter, both natural and biomedical images are used to present the effectiveness of the optimization algorithm proposed in Chapter 4 in an image noise reduction application. The experimental results show updating  $\mathbf{w}$ ,  $\mathbf{c}$  and  $\mathbf{s}$  only once is sufficient. We also observed that the update for  $\mathbf{s}$  did not lead to significant quantitative and visual improvements. We believe the original variance –  $\mathbf{s}$  of each VQ partition from the PWS filter results is somehow good enough. It may not be necessary to perform optimization of the variance –  $\mathbf{s}$ . In conclusion, we believe only one round updates for  $\mathbf{w}$  and  $\mathbf{c}$  is essential. To reduce the computational complexity even further but still keep quite close performance with updating both  $\mathbf{w}$  and  $\mathbf{c}$ , we recommend to perform only a closed form solution for updating  $\mathbf{w}$ . By doing only closed form optimization for  $\mathbf{w}$ , you can get very close performance of updating all three parameters and its computational expense is relatively low. Also the 3D Soft-PWS filters results are significantly better than the 2D ones, it indicated that the other two slices containing some important information very helpful in reconstruction process.

The results show the Soft-PWS filters produce favorable results, thus we conclude that the Soft-PWS filters are suitable for image noise reduction applications.

## CHAPTER 7

### CONCLUSIONS

Preserving the edge and other important information in the original images is a very challenging issue for image processing, especially for filters used in image deconvolution and image denoising area. The proposed generic PWS non-linear filters and its enhancement – Soft-PWS filters have shown improvements in performances vs the linear FIR Wiener filters in image denoising applications studied by the previous work<sup>2,7</sup> and in this dissertation. The proposed subspace PWS filter aimed resolving the computational complexity issue of the PWS filters which come from the partitioning of the observation space, and significantly reduced the computational and memory demands of the partitioning process by using PCA, selecting the center of the filter window, or both. Because of these reductions/improvements, the SPWS filters allow for larger filter window sizes than would be practical with the PWS filter. This also means much better performances can be expected since using a larger observation window. The quantitative error analysis results in Chapter 5 show that not only the SPWS filters require less memory and can be implemented with significantly fewer computations, but also the performance of SPWS filters is essentially equivalent to, or slightly better than, the PWS filter of the same size.

We have observed that the performance of these partition filters is heavily dependent on the VQ codebook. The SPWS filters (Center, PCA and Center-PCA), incorporating only the selected important portion of the observation window into the partitioning scheme, not only provides computational savings, but also actually improves performance in most cases. We believe that this is due to the fact that the selected important portion of the observation window contains samples that are most correlated to the desired samples, and generally corresponds to the largest magnitude filter weights.

We have proposed an improved optimization method for the Soft-PWS filters. The method uses a closed form solution for the filter weights and a compound numerical solution for the other coefficients –  $c$  and  $s$ . The compound solution uses alternating application of the Quasi-Newton and the steepest descent methods. We have demonstrated that the Soft-PWS filter with the proposed optimization provides improved MSE performance over the PWS and Wiener filters. The bulk of the improvement can be credited to the closed form solution for the filter weights with the initial values for the other parameters. Further improvement is possible using the compound optimization method for the remaining parameters. We have also presented a radial basis function interpretation of the Soft-PWS filters that we believe provides greater insight into the nature of this filter class.

## APPENDIX A

### GRADIENT WITH RESPECT TO CODEWORDS

The gradient of (4.3) with respect to  $\mathbf{c} = [\mathbf{c}_1^T, \mathbf{c}_2^T, \dots, \mathbf{c}_M^T]^T$  and  $\mathbf{s} = [s_1, s_2, \dots, s_M]^T$  are needed for both the Quasi-Newton method and steepest descent methods. Thus, we derive them here. To begin, let us express the soft partition function as

$$\hat{p}_i(\mathbf{x}) = \frac{e^{\beta_i}}{\gamma}, \quad (\text{A.1})$$

where

$$\beta_i = -\frac{\|\mathbf{x} - \mathbf{c}_i\|^2}{s_i}, \quad (\text{A.2})$$

and

$$\gamma = \sum_{k=1}^M e^{\beta_k}. \quad (\text{A.3})$$

Now the Soft-PWS filter can be expressed as

$$F_{Soft-PWS}(\mathbf{x}) = \sum_{i=1}^M \frac{e^{\beta_i}}{\gamma} \mathbf{w}_i^T \mathbf{x}. \quad (\text{A.4})$$

From (4.3), the first derivatives of the cost function with respect to  $\mathbf{c}_i$  become

$$\begin{aligned} \nabla_{\mathbf{c}_i} J(\mathbf{w}, \mathbf{c}, \mathbf{s}) &= -2E\{d \nabla_{\mathbf{c}_i} F_{Soft-PWS}\} + 2E\{F_{Soft-PWS} \nabla_{\mathbf{c}_i} F_{Soft-PWS}\} \\ &= -2E\{(d - F_{Soft-PWS}) \nabla_{\mathbf{c}_i} F_{Soft-PWS}\}. \end{aligned} \quad (\text{A.5})$$

Using (A.4), the first derivatives of  $F_{Soft-PWS}$  with respect to  $\mathbf{c}_i$  can be expressed as

$$\begin{aligned}
\nabla_{\mathbf{c}_i} F_{Soft-PWS} &= \nabla_{\mathbf{c}_i} \left( \frac{\sum_{i=1}^M e^{\beta_i} \mathbf{w}_i^T \mathbf{x}}{\gamma} \right) \\
&= \frac{\nabla_{\mathbf{c}_i} (\sum_{i=1}^M e^{\beta_i} \mathbf{w}_i^T \mathbf{x}(\mathbf{n})) \gamma - \nabla_{\mathbf{c}_i} \gamma \sum_{i=1}^M e^{\beta_i} \mathbf{w}_i^T \mathbf{x}}{\gamma^2} \\
&= \frac{\nabla_{\mathbf{c}_i} (e^{\beta_i} \mathbf{w}_i^T \mathbf{x}) - \nabla_{\mathbf{c}_i} \gamma \sum_{i=1}^M \frac{e^{\beta_i}}{\gamma} \mathbf{w}_i^T \mathbf{x}}{\gamma} \\
&= \frac{\nabla_{\mathbf{c}_i} e^{\beta_i} \mathbf{w}_i^T \mathbf{x} - \nabla_{\mathbf{c}_i} \gamma F_{Soft-PWS}}{\gamma}.
\end{aligned} \tag{A.6}$$

The derivative of  $e^{\beta_i}$  with respect to  $\mathbf{c}_i$  is given by

$$\begin{aligned}
\nabla_{\mathbf{c}_i} e^{\beta_i} &= \nabla_{\beta_i} e^{\beta_i} \nabla_{\mathbf{c}_i} \beta_i \\
&= e^{\beta_i} \nabla_{\mathbf{c}_i} \left( -\frac{\|\mathbf{x} - \mathbf{c}_i\|^2}{s_i} \right) \\
&= e^{\beta_i} \left( -\frac{2(\mathbf{x} - \mathbf{c}_i)}{s_i} \right) \\
&= \frac{2e^{\beta_i}(\mathbf{x} - \mathbf{c}_i)}{s_i}.
\end{aligned} \tag{A.7}$$

The derivative of  $\gamma$  with respect to  $\mathbf{c}_i$  is given by

$$\begin{aligned}
\nabla_{\mathbf{c}_i} \gamma &= \nabla_{\mathbf{c}_i} \left( \sum_{k=1}^M e^{\beta_k} \right) \\
&= \nabla_{\mathbf{c}_i} e^{\beta_i}.
\end{aligned} \tag{A.8}$$

By substituting (A.7) and (A.8) into (A.6), the derivative of  $F_{Soft-PWS}$  with respect to  $\mathbf{c}_i$  then becomes

$$\begin{aligned}
\nabla_{\mathbf{c}_i} F_{Soft-PWS} &= \frac{\nabla_{\mathbf{c}_i} e^{\beta_i} \mathbf{w}_i^T \mathbf{x} - \nabla_{\mathbf{c}_i} e^{\beta_i} F_{Soft-PWS}}{\gamma} \\
&= \frac{\nabla_{\mathbf{c}_i} e^{\beta_i} (\mathbf{w}_i^T \mathbf{x} - F_{Soft-PWS})}{\gamma} \\
&= \frac{2e^{\beta_i}(\mathbf{x} - \mathbf{c}_i)(\mathbf{w}_i^T \mathbf{x} - F_{Soft-PWS})}{s_i \gamma} \\
&= \frac{2(\mathbf{x} - \mathbf{c}_i)}{s_i} (\mathbf{w}_i^T \mathbf{x} - F_{Soft-PWS}) \hat{p}_i(\mathbf{x}).
\end{aligned} \tag{A.9}$$

Then, using (A.5) and (A.9) the derivative of the cost function with respect to  $\mathbf{c}_i$  becomes

$$\begin{aligned}
\nabla_{\mathbf{c}_i} J(\mathbf{w}, \mathbf{c}, \mathbf{s}) &= -2E\left\{(d - F_{Soft-PWS}) \frac{2(\mathbf{x} - \mathbf{c}_i)}{s_i} (\mathbf{w}_i^T \mathbf{x} - F_{Soft-PWS}) \hat{p}_i(\mathbf{x})\right\} \\
&= -4E\left\{(d - F_{Soft-PWS}) \frac{(\mathbf{x} - \mathbf{c}_i)}{s_i} (\mathbf{w}_i^T \mathbf{x} - F_{Soft-PWS}) \frac{e^{-\frac{\|\mathbf{x} - \mathbf{c}_i\|^2}{s_i}}}{\sum_{k=1}^M e^{-\frac{\|\mathbf{x} - \mathbf{c}_k\|^2}{s_k}}}\right\}.
\end{aligned} \tag{A.10}$$

Finally, let  $\mathbf{g}_{\mathbf{c}_i} = \nabla_{\mathbf{c}_i} J(\mathbf{w}, \mathbf{c}, \mathbf{s})$  and  $\nabla_{\mathbf{c}} J(\mathbf{w}, \mathbf{c}, \mathbf{s}) = [\mathbf{g}_{\mathbf{c}_1}^T, \mathbf{g}_{\mathbf{c}_2}^T, \dots, \mathbf{g}_{\mathbf{c}_M}^T]^T$ .

## APPENDIX B

### GRADIENT WITH RESPECT TO RADIAL BASIS PARAMETERS

From (4.3), the first derivatives of the cost function with respect to  $\mathbf{s}$  become

$$\begin{aligned}\frac{\partial J(\mathbf{w}, \mathbf{c}, \mathbf{s})}{\partial s_i} &= -2E\left\{d \frac{\partial F_{Soft-PWS}}{\partial s_i}\right\} + 2E\left\{F_{Soft-PWS} \frac{\partial F_{Soft-PWS}}{\partial s_i}\right\} \\ &= -2E\left\{(d - F_{Soft-PWS}) \frac{\partial F_{Soft-PWS}}{\partial s_i}\right\}.\end{aligned}\quad (B.1)$$

From (A.4), the first derivatives of  $F_{Soft-PWS}$  with respect to  $s_i$  becomes

$$\begin{aligned}\frac{\partial F_{Soft-PWS}}{\partial s_i} &= \frac{\frac{\partial(\sum_{i=1}^M e^{\beta_i} \mathbf{w}_i^T \mathbf{x}) \gamma}{\partial s_i} - \frac{\partial \gamma \sum_{i=1}^M e^{\beta_i} \mathbf{w}_i^T \mathbf{x}}{\partial s_i}}{\gamma^2} \\ &= \frac{\frac{\partial(e^{\beta_i} \mathbf{w}_i^T \mathbf{x})}{\partial s_i} - \frac{\partial \gamma \sum_{i=1}^M \frac{e^{\beta_i}}{\gamma} \mathbf{w}_i^T \mathbf{x}}{\partial s_i}}{\gamma} \\ &= \frac{\frac{\partial e^{\beta_i} \mathbf{w}_i^T \mathbf{x}}{\partial s_i} - \frac{\partial \gamma F_{Soft-PWS}}{\partial s_i}}{\gamma}.\end{aligned}\quad (B.2)$$

The derivative of  $e^{\beta_i}$  with respect to  $s_i$  becomes

$$\begin{aligned}\frac{\partial e^{\beta_i}}{\partial s_i} &= \frac{\partial e^{\beta_i}}{\partial \beta_i} \frac{\partial \beta_i}{\partial s_i} \\ &= e^{\beta_i} \frac{\partial(-\frac{\|\mathbf{x} - \mathbf{c}_i\|^2}{s_i})}{\partial s_i} \\ &= e^{\beta_i} \left(-\frac{\|\mathbf{x} - \mathbf{c}_i\|^2}{s_i^2}\right) \\ &= \frac{e^{\beta_i} \|\mathbf{x} - \mathbf{c}_i\|^2}{s_i^2}.\end{aligned}\quad (B.3)$$

The derivative of  $\gamma$  with respect to  $s_i$  becomes

$$\begin{aligned}\frac{\partial \gamma}{\partial s_i} &= \frac{\partial (\sum_{k=1}^M e^{\beta_k})}{\partial s_i} \\ &= \frac{\partial e^{\beta_i}}{\partial s_i}.\end{aligned}\tag{B.4}$$

By substituting (B.3) and (B.4) into (B.2), the derivative of  $F_{Soft-PWS}$  with respect to  $s_i$  then becomes

$$\begin{aligned}\frac{\partial F_{Soft-PWS}}{\partial s_i} &= \frac{\frac{\partial e^{\beta_i} \mathbf{w}_i^T \mathbf{x}}{\partial s_i} - \frac{\partial e^{\beta_i} F_{Soft-PWS}}{\partial s_i}}{\gamma} \\ &= \frac{\frac{\partial e^{\beta_i} (\mathbf{w}_i^T \mathbf{x} - F_{Soft-PWS})}{\partial s_i}}{\gamma} \\ &= \frac{e^{\beta_i} \|\mathbf{x} - \mathbf{c}_i\|^2 (\mathbf{w}_i^T \mathbf{x} - F_{Soft-PWS})}{s_i^2 \gamma} \\ &= \frac{\|\mathbf{x} - \mathbf{c}_i\|^2}{s_i^2} (\mathbf{w}_i^T \mathbf{x} - F_{Soft-PWS}) \hat{p}_i(\mathbf{x}).\end{aligned}\tag{B.5}$$

Then, the derivative of the cost function with respect to  $s_i$ , (B.1) becomes

$$\begin{aligned}\frac{\partial J(\mathbf{w}, \mathbf{c}, \mathbf{s})}{\partial s_i} &= -2E\{(d - F_{Soft-PWS}) \frac{\|\mathbf{x} - \mathbf{c}_i\|^2}{s_i^2} (\mathbf{w}_i^T \mathbf{x} - F_{Soft-PWS}) \hat{p}_i(\mathbf{x})\} \\ &= -2E\{(d - F_{Soft-PWS}) \frac{\|\mathbf{x} - \mathbf{c}_i\|^2}{s_i^2} (\mathbf{w}_i^T \mathbf{x} - F_{Soft-PWS}) \frac{e^{-\frac{\|\mathbf{x} - \mathbf{c}_i\|^2}{s_i}}}{\sum_{k=1}^M e^{-\frac{\|\mathbf{x} - \mathbf{c}_k\|^2}{s_k}}}\}.\end{aligned}\tag{B.6}$$

Let  $\mathbf{g}_{s_i} = \frac{\partial J(\mathbf{w}, \mathbf{c}, \mathbf{s})}{\partial s_i}$ , then  $\nabla_{\mathbf{s}} J(\mathbf{w}, \mathbf{c}, \mathbf{s}) = [\mathbf{g}_{s_1}^T, \mathbf{g}_{s_2}^T, \dots, \mathbf{g}_{s_M}^T]^T$  is the first derivative function with respect to  $\mathbf{s}$  which the SDM and QNM used to derive the optimized solutions.

## APPENDIX C

### MATLAB CODES USED FOR SUBSPACE PWS FILTER

Listing C.1: Subspace PWS filter main program

```
% PWS (Partition based Weighted Sum) program Main Run program
% PCA space partitioning vs. standard LBG method
% Wiener, LBG PWS, SPWS Center, SPWS PCA, SPWS Center PCA
% Yong Lin, University of Dayton, Yong.Lin@ieee.org
% Created: Sep 2003; Last Modified: Sep 2004
%
% Calling subroutines: 1) Image_Gene; 2) Fname_Gene; 3)
    Wiener_Rout; 4) PWS_Rout.

close all; clear all; clc;

% loop for different type of blur. 0, no-blur. 1, gaussian.
    2, motion. 3, out of focus.
for NP=2:2
    % loop for different type of noise level: 1, low noise;
        2, high noise.
    for NN=1:1
```

```

% Generate Training image
load t-images.mat;
f_name = Fname_Gene(1, NP, NN);
f_name=strcat('Train_', f_name, '.mat');
N_max=600; Image_Gene(NP, NN, N_max, im_1, f_name);
% Generate Independent Test image
f_name = Fname_Gene(1, NP, NN);
f_name=strcat('Test_', f_name, '.mat');
N_max=400; Image_Gene(NP, NN, N_max, im_2, f_name);
clear im_1, im_2;
% loop for different type of reconstruction method:
    1: Wiener; 2: LBG PWS; 3: SPWS Center; 4: SPWS PCA
    ; 5: SPWS Center PCA.
f_index=[1 2 5];
for NM_i=1:1:3
    NM = f_index(NM_i);
    % Generate a file name for writing results to
    f_name = Fname_Gene(NM, NP, NN);
    f_name = strcat(f_name, '.txt')
    % Open the file to write results
    fid = fopen(f_name, 'w');
    % loop for different observation window size by
        using for loop
    for NW = 51:10:51
        fprintf(fid, 'F_size;%2i;\n', NW);
        if (NM == 1)

```

```

        Wiener_Rout(fid , NW, NP, NN);    % FIR
        wiener filter results
    else
        PWS_Rout(fid , NW, NM, NP, NN);    % (S
        )PWS filter results
    end
end
end
fclose(fid);
end
end
end

```

#### Listing C.2: Blurred image generation

```

function Image_Gene(t_p , t_n , N_max, im_a , f_name);

% PWS (Partition based Weighted Sum) program: Generate
% Blurred Images
% Yong Lin, University of Dayton, Yong.Lin@ieee.org
% Created: Sep 2003; Last Modified: Sep 2004
%
% Calling subroutines: N/A
%
% function Image_Gene(t_p , t_n , N_max, im_a , f_name);
% t_p    : type of blur. 0, no-blur. 1, gaussian. 2, motion.
%         3, out of focus.
% t_n    : type of noise level: 1, low noise; 2, high noise.

```

```

% N_max : Max pixels of the output image.
% im_a   : original image.
% f_name: file name for save generated images.

[nx, ny] = size(im_a);
im_oax=max(max(im_a));
if (im_oax == 1)
    im_k=im_a;
else
    im_oin=min(min(im_a));
    im_k=(im_a-im_oin)/(im_oax-im_oin);
end

% processing blur operations
if (t_p ~= 0)
    switch t_p
    case 1
        % Gaussian blur PSF
        P_len = 11;
        PSF = fspecial('gaussian',[P_len P_len],1.5);
    case 2
        % Motion blur PSF
        P_len = 9;
        PSF = fspecial('motion',P_len,0);
    case 3
        % Out-of-focus blur PSF
        P_rad = 3;
        PSF = fspecial('disk',P_rad);
    end
end

```

```

        im_m = imfilter(im_k,PSF,'circular','conv');
    else
        im_m = im_k;
        PSF = 0;
    end

% processing noise operations
state = 19;
randn('state', state);
noisel = randn(size(im_a));
std = 0;
switch t_n
case 1      % Low Noise
    % add Gaussian white noise of zero mean with 0.01
    % variance (1%) to the image
    std = 0.01;
case 2      % High Noise
    % add Gaussian white noise of zero mean with 0.1 variance
    % (10%) to the image
    std = 0.1;
end
im_u = im_m + noisel*std;

% Generate output images
pat_x = 1;                                pat_y = 1;
pat_xmax = pat_x+N_max-1;                  pat_ymax = pat_y+N_max-1;

```

```

im_3 (:,:) = im_a(pat_x:pat_xmax, pat_y:pat_ymax);
im_3b (:,:) = im_u(pat_x:pat_xmax, pat_y:pat_ymax);

% Save images to file
save(f_name, 'im_3', 'im_3b', 'PSF', 'N_max' );

```

#### Listing C.3: Noisy image generation

```

% SPWS (Partition based Weighted Sum) program:
% Image Denoiseing Application: Noisy Image Generation
% Yong Lin, University of Dayton, Yong.Lin@ieee.org
% Last Modified: Sep 2004

```

```

clear all; close all; clc;

load kenaerial1;
des1=double(kenaerial1);
[size_y, size_x]=size(des1);
std=30; % Set standard deviation value
obs1=des1+std*randn(size_y, size_x);
save training.mat des1 obs1;

```

#### Listing C.4: MSE PSNR generation routine

```

function [mse, mae, psnr] = MSE_PSNR(im_de, im_ob);
%
% PWS (Partition based Weighted Sum) program: Generate MSE,
% MAE and PSNR
% Yong Lin, University of Dayton, Yong.Lin@ieee.org

```

```

% Created: Sep 2003; Last Modified: Sep 2004
%
% Calling subrim_otines: N/A.
%
% function [mse, mae, psnr] = MSE_PSNR(im_de, im_ob);
% im_de      desired image
% im_ob      observed/corrupted image

[sy sx] = size(im_ob);

im_ax=max(max(im_ob));
im_in=min(min(im_ob));

mse = sqrt(sum((sum((im_de-im_ob).^2))'));
mae = mean(mean(abs(im_de-im_ob)))*255;
psnr = 10*log10((im_ax-im_in)^2*sx*sy/mse^2);

```

Listing C.5: PWS filter routine

```

function PWS_Rout(fid, filtersize, t_m, t_p, t_n);

% PWS (Partition based Weighted Sum) program: (S)PWS
% deconvolution results
% Yong Lin, University of Dayton, Yong.Lin@ieee.org
% Created: Sep 2003; Last Modified: Sep 2004
%

```

```

% Calling subroutines: 1) PCA_Gene; 2) CodeBK_Gene; 3)
    PWS_Weight_Gene; 4) SPWSs; 5) Fname_Gene
%
% function PWS_Rout(fid , filtersize , t_n , t_p);
% fid          : output file id
% filtersize: FIR inverse filter size. ex: 11 x 11 window
% t_m          : type of reconstruction method: 1: Wiener; 2:
    LBG PWS; 3: SPWS Center; 4: SPWS PCA; 5: SPWS Center PCA.
% t_p          : type of blur. 0, no-blur. 1, gaussian. 2,
    motion. 3, out of focus.
% t_n          : type of noise level: 1, low noise; 2, high
    noise.

% load training data
clear f_name
f_name = Fname_Gene(1, t_p, t_n);
f_name=strcat('Train_', f_name, '.mat');
load (f_name);

% Set for different PCA size
switch t_m
case 2
    f_f = filtersize;
    A_matr = 0;
case 3
    f_f = 5;

```

```

        A_matr = 0;
    case {4, 5}
        pca_n = 5;
        % Generate A matrix
        A_matr = PCA_Gene(im_3b, pca_n, filtersize, t_m);
        fprintf(fid, 'PCA eigen #; %2i; \n', pca_n);
        f_f = pca_n;
    end

% loop for different number of VQ codewords (ex: from 10 to
60)
for N_VQ = 5:4:5
    % load training data
    clear f_name
    f_name = Fname_Gene(1, t_p, t_n);
    f_name= strcat('Train_', f_name, '.mat');
    load (f_name);

    % Start timer for computation time
    t1 = clock; t2 = clock;

    max_code= 10*N_VQ          % codeword size: number between
        10 to 60
    fprintf(fid, 'VQ#; %2i; \n', max_code);

    codebook=zeros(f_f, max_code); % initialize codebook

```

```

[codebook,code_nobs]=CodeBK_Gene(im_3b, codebook, 5, 10,
    filtersize, A_matr, t_m); % generate codebook

% SPWS method
% generate W
W=PWS_Weight_Gene(im_3,im_3b,codebook, filtersize, t_n,
    t_p, A_matr, t_m);

% time stampe for training
fprintf(fid,'VQ train time; %6.2f; ', (etime(clock,t1)
    /60));

% load test data
clear im_3 im_3b N_Max PSF;
f_name = Fname_Gene(1, t_p, t_n);
f_name=strcat('Test_', f_name, '.mat');
load (f_name);
[sy sx] = size(im_3); NM_3=fix(filtersize/2);

% reset timer for measuring test cycle computation time
t1 = clock;
clear im_rc;

% reconstructing test image by SPWSs
im_rc = SPWSs(im_3b, codebook, W, 1, filtersize, A_matr,
    t_m);

```

```

% calculate MSEs
[mse2, mae2, psnr2] = MSE_PSNR(im_3(:,(NM_3+1):(sx-NM_3))
    , im_rc)
fprintf(fid, 'PWS MSE; %3.5f; MAE; %3.5f; PSNR; %3.5f;
    test time; %6.2f; total time; %6.2f;\n', mse2, mae2,
    psnr2, (etime(clock, t1)/60), (etime(clock, t2)/60));

% Save reconstructed image to mat format
clear f_name;
f_name = Fname_Gene(t_m, t_p, t_n);
f_name = strcat('.', data, f_name, '_', int2str(N_VQ), '_',
    int2str(filtersize), '.mat');
save(f_name, 'im_rc', 'mse2', 'psnr2', 'mae2', 'codebook',
    'W');
end

```

Listing C.6: PCA generation routine

```

function A_matr = PCA_Gene(im_x, top_n, filtersize, t_m);

%
% Calling subroutines: N/A.
%
% function A_matr = PCA_Gene(im_x, top_n, filtersize, t_m);
% A_matr      : A Matrix
% im_x        : input image x

```

```

% top_n      : PCA requirement of top selection number
% t_m        : type of reconstruction method: 1: Wiener; 2:
               LBG PWS; 3: SPWS Center; 4: SPWS PCA; 5: SPWS Center PCA.
% filtersize : FIR inverse filter size. ex: 11 x 11 window

[NM_x, NM_y] = size(im_x);

% set window size
wsx = filtersize;
wsy = 1;

% convert X
X_1=im2col(im_x,[wsy,wsx],'sliding');
X_1=X_1-ones(wsx*wsy,1)*mean(X_1);

if (t_m == 5)
    % Center PCA method
    f_f = fix(filtersize/2);
else
    % PCA method
    f_f = filtersize;
end

ws=filtersize;           % window dimension
ws_2 = ws;

```

```

w_cen=floor(ws_2/2)+1;
w_1 = w_cen - floor(f_f/2);
w_2 = w_cen + floor(f_f/2);
X_2 = X_1(w_1:w_2,:);

clear X_1;

[N_x, N_y] = size(X_2);

C = cov(X_2');

% calculate eigen value & vector
[E,V] = eigs(C, top_n);

A_matr = E';

% save file
% save results.mat V E C V1 E1 -mat

```

Listing C.7: SPWS codebook generation routine

```

function [cdbk_n, count_cod] = CodeBK_Gene(im_de, cdbk_i, nits,
    minobs, ws, A_matr1, t_m);

% SPWS (Partition based Weighted Sum) program: Generate
    codebook (PCA, Center, Center-PCA)
% Original created by Dr Russ Hardie, University of Dayton

```

```

% Enhanced by Yong Lin, University of Dayton, Yong.Lin@ieee.
    org
% Last Modified: Sep 2004
%
% Calling subroutines: N/A.
%
% function [cdbk_n,count_cod] = CodeBK_Gene(im_de,cdbk_i,nits
    ,minobs,filtersize , A_matr1);
% cdbk_n      : Output codebook
% count_cod   : codeword counts
% im_de       : desired image
% cdbk_i      : Input codebook
% nits        : number of iterations
% minobs      : min number of each codeword
% ws          : Inverse filter size. ex: 11 x 11 window
% A_matr      : A Matrix
% t_m         : type of reconstruction method: 1: Wiener; 2:
    LBG PWS; 3: SPWS Center; 4: SPWS PCA; 5: SPWS Center PCA.

[sy,sx,sz]=size(im_de);
[wss,N]=size(cdbk_i); % Window size and Number of levels (
    code book)

wss_2= floor(wss/2);
ws_2 = ws;

```

```

% Center method
switch t_m
case {2, 3}
    f_f = wss;
case 4
    f_f = ws;
case 5
    f_f = fix(ws/2);
end

w_cen=floor(ws_2/2)+1;
w_1 = w_cen - floor(f_f/2);
w_2 = w_cen + floor(f_f/2);

% pick evenly spaced vectors to initialize codeb
if sum(sum(cdbk_i))==0,
    X_1=im2col(im_de(:, :, 1) , [1, ws] , 'sliding ');
    X_1 = X_1-ones(ws*1,1)*mean(X_1);
    X_2 = X_1(w_1:w_2, :);
    if (t_m == 4 | t_m == 5)
        X_1 = A_matr1*X_2;
    else
        X_1 = X_2;
    end
    clear X_2;
    [junk, nov]=size(X_1);

```

```

        pick=round(linspace(1,nov,N));
        cdbk_i=X_1(:,pick);
end

cdbk_n=cdbk_i;
clear cdbk_i;

% loop for iterations
for mi=1:nits
    xl=sprintf('Beginning LBG clustering iteration %d',mi);
    disp(xl)

    count=0;      % Find distance from each obs vec to one
                  % code word at a time
    count_cod=zeros(1,N);
    Sum_cod=zeros(N,wss);
    % loop for 3D data
    for mj = 1:sz
        clear X_1;
        X_1=im2col(im_de(:,:,mj),[1,ws], 'sliding ');
        X_1=X_1-ones(ws*1,1)*mean(X_1);
        X_2 = X_1(w_1:w_2,:);
        if (t_m == 4 | t_m == 5)
            X_1 = A_matr1*X_2;
        else
            X_1 = X_2;

```

```

end

clear X_2;

[junk, nov]=size(X_1);

D=zeros(N, nov);

for mk=1:N % loop over codewords
    CWMX=cdbk_n(:,mk)*ones(1, nov);
    D(mk,:)=sum((X_1-CWMX).^2); % distance from each
        obs to code word mk
end

% assign each obs to codeword with min distance
[mind, minind]=min(D); % minind = set of codeword
assignment indices.

for mk=1:N % loop over codewords
    nI=find(minind==mk); % indices of all obs vecs
        assigned to mk
    if length(nI)~=0
        count_cod(mk)=count_cod(mk)+length(nI);
        Sum_cod(mk,:)=Sum_cod(mk,:)+sum(X_1(:, nI)');
    end
end

end

for ml=1:N % loop over codewords
    if count_cod(ml) < minobs
        [maxct, maxind]=max(count_cod);
        nI=find(minind==maxind);

```

```

        cdbk_n(:,ml)=X_1(:,nI(1));
        disp('split big codeword.')
    else
        AVGCW=Sum_cod(ml,:)/count_cod(ml);
        cdbk_n(:,ml)=AVGCW;
    end
end
clear D;
end

```

Listing C.8: SPWS weights generation routine

```

function [W] = PWS_Weight_Gene(im_de, im_ob, cdbk, ws, t_n,
    t_p, A_matr1, t_m);
%
% PWS (Partition based Weighted Sum) program: Generate weight
% coefficients for each PWS partition.
% Original created by Dr Russ Hardie, University of Dayton
% Enhanced by Yong Lin for PCA, Center and Center-PCA SPWS
% filter
% University of Dayton, Yong.Lin@ieee.org
% Last Modified: Sep 2004
%
% Calling subroutines: 1) Fname_Gene.
%
% function [W] = PWS_Weight_Gene(im_de,im_ob, cdbk,ws, t_n,
    t_p, A_matr1, t_m);

```

```

% W                : Weights (impulse response)
% im_de            : Desired image
% im_ob            : Observed/corrupted image
% cdbk             : Codebook
% ws               : Inverse filter size. ex: 11 x 11 window
% t_p              : type of blur. 0, no-blur. 1, gaussian. 2,
                    motion. 3, out of focus.
% t_n              : type of noise level: 1, low noise; 2, high
                    noise.
% A_matr           : A Matrix
% t_m              : type of reconstruction method: 1: Wiener; 2:
                    LBG PWS; 3: SPWS Center; 4: SPWS PCA; 5: SPWS Center PCA.

% load Wiener filter weights
clear f_name;
f_name = Fname_Gene(1, t_p, t_n); f_name=strcat('.\data\W_',
        f_name, int2str(ws), '.mat');
load (f_name, 'W_g');

[xa,ya,sz]=size(im_de);    % Get image size info
[wss,N]=size(cdbk);        % number of codewords
numwin=ws;

% Center method
switch t_m
case {2, 3}

```

```

        f_f = wss;
    case 4
        f_f = ws;
    case 5
        f_f = fix(ws/2);
    end

    w_cen=floor(numwin/2)+1;
    w_1 = w_cen - floor(f_f/2);
    w_2 = w_cen + floor(f_f/2);

    % Go over first image
    for mi = 1:sz
        X_1=im2col(im_ob(:,: ,mi) ,[1,ws] , 'sliding ');
        X_1=X_1-ones(ws*1,1)*mean(X_1);
        X_2 = X_1(w_1:w_2,:);
        if (t_m == 4 | t_m == 5)
            X_1 = A_matr1*X_2;
        else
            X_1 = X_2;
        end
        clear X_2;
        [junk ,nov]=size(X_1);

        D=zeros(N,nov);
        for n=1:N % loop over codewords

```

```

        CWMX=cdbk(:,n)*ones(1,nov);
        D(n,:)=sum((X_1-CWMX).^2); % distance from each obs
            to code word n
    end

    % assign each obs to codeword with min distance
    [mind,minind]=min(D); % minind = set of codeword
        assignment indices.
    partindex(mi,:)= minind;
end

% Compute the filter weights for each partition
W=zeros(numwin,N);
ws_2r = round(ws/2);
ws_2f = fix(ws/2);

minnumber=numwin;
cou_zero = 0 ;
for mi=1:N
    % initialize the correlation matrices and vectors
    count_1=0;
    R=zeros(numwin^2, 1);
    P=zeros(numwin, 1);
    for mk = 1:sz
        nI=find(partindex(mk,:)==mi);
        count=length(nI);

```

```

        if count ~= 0
            count_1 = count_1 + count;
            X_b=im2col(im_ob(:, :,mk),[1,ws], 'sliding ');
            X_o=im2col(im_de(:, ws_2r:(ya-ws_2f),mk),[1,1], '
                sliding ');
            for ml=1:count
                R_1= X_b(:, nI(ml))*X_b(:, nI(ml))';
                P_1= X_b(:, nI(ml))*X_o(nI(ml));
                R=R+reshape(R_1,numwin^2,1);
                P=P+P_1;
            end
        end
    end
    if count_1 > minnumber
        R=R/count_1;
        P=P/count_1;
        RR=reshape( R, [numwin numwin] );
        if det(RR) == 0
            cou_zero = cou_zero +1;
        end
        W(:,mi)= inv(RR)*P;
    else
        W(:,mi)=W_g;
    end
end
cou_zero

```

Listing C.9: SPWS filtering routine

```
function out = SPWSs(im_ob, code, W, wsy, wsx, A_matr1, t_m);
%
% Calling subroutines: 1) Fname_Gene.
%
% function [W] = PWS_Weight_Gene(im_de, im_ob, cdbk, ws, t_n,
    t_p, A_matr1, t_m);
% im_ob          : Observed/corrupted image
% code           : Codebook
% W              : Weights for each partition
% wsy            window span in vertical dimension (odd)
% wsx            window span in horizontal dimension (odd)
% A_matr         : A Matrix
% t_m            : type of reconstruction method: 1: Wiener; 2:
    LBG PWS; 3: SPWS Center; 4: SPWS PCA; 5: SPWS Center PCA.

[sy, sx] = size(im_ob); % Get image size info
ws = wsx;
% bws = (ws - 1) / 2; % get border size needed
% numwin = ws;

bwsy = (wsy - 1) / 2;
bwsx = (wsx - 1) / 2;
numwin = wsy * wsx;

[wss, N] = size(code); % number of codewords
```

```

wss_2= sqrt(wss);

% Center method
switch t_m
case {2, 3}
    f_f = wss;
case 4
    f_f = ws;
case 5
    f_f = fix(ws/2);
end

w_cen=floor(numwin/2)+1;
w_1 = w_cen - floor(f_f/2);
w_2 = w_cen + floor(f_f/2);

for i=1:sy-2*bwsy
    for j=1:sx-2*bwsx
        temp1=im_ob(i:i+wsy-1,j:j+wsx-1); % Extract window
            from observed/corrupted image
        temp1=reshape(temp1,numwin,1);
        temp2=temp1 - ones(numwin,1)*mean(temp1);
        temp3 = temp2(w_1:w_2);
        if (t_m == 4 | t_m == 5)
            temp4 = A_matr1* temp3;
        else

```

```

        temp4 = temp3;
    end
    D=zeros(N, 1);
    for n=1:N % loop over codewords
        D(n)=sum((temp4-code(:,n)).^2); % distance from
            each im_ob to code word n
    end
    [mind,minind]=min(D); % minind = set of codeword
        assignment indices.
    out(i,j)=W(:,minind) '*templ;
end
end

```

## APPENDIX D

### MATLAB CODES USED FOR SOFT-PWS FILTER

Listing D.1: Soft-PWS filter routine for generating PWS codebook and PWS weights

```
clear all; close all; clc;

% set timer
t1 = clock;

% load data
load training_aerial_10.mat
obs2 = obs1;          des2 = des1;

% settings
wsx=5;                wsy=5;
bwsy=(wsy-1)/2;       bwsx=(wsx-1)/2;
[sy ,sx]=size(des2);

% For FIR Wiener Filter
% Set initial codebook
```

```

code=zeros (wsx*wsy,1);
% Generate Wiener Weights
[W_w,Count] = wienervqt (des1,obs1,code,wsy,wsx);
% Generate Output of FIR Wiener filter
out_w = wienervqf (obs2,code,W_w,wsy,wsx);
t(1)=(etime(clock,t1)/60)

[mae(1),mse(1),psnr(1)]= MSE_PSNR(obs2((bwsy+1):(sy-bwsy),(
    bwsx+1):(sx-bwsx)),des2((bwsy+1):(sy-bwsy),(bwsx+1):(sx-
    bwsx)))
[mae(2),mse(2),psnr(2)]= MSE_PSNR(out_w((bwsy+1):(sy-bwsy),(
    bwsx+1):(sx-bwsx)),des2((bwsy+1):(sy-bwsy),(bwsx+1):(sx-
    bwsx)))
% figure(1); imstd (des2((bwsy+1):(sy-bwsy),(bwsx+1):(sx-bwsx
    ))); title ('high dose image');
% figure(2); imstd (obs2((bwsy+1):(sy-bwsy),(bwsx+1):(sx-bwsx
    ))); title (['noisy image, MSE = ', num2str(mse(1))])
% figure(3); imstd (out_w((bwsy+1):(sy-bwsy),(bwsx+1):(sx-
    bwsx))); title (['Wiener reconstructed image, MSE = ',
    num2str(mse(2))])

% For Hard PWS Filter
nits=10;                minobs=20;
M = 50;
% update codebook
code=zeros (wsx*wsy,M);

```

```

[code,nobs,err]=vqcodenew(obs1,code,nits,minobs);
% Get new weights for PWS
[W_pws,Center] = wienervqt(des1,obs1,code,wsy,wsx);
% Generate Output of PWS filter
out_pws = wienervqf(obs2,code,W_pws,wsy,wsx);
t(2)=(etime(clock,t1)/60)-t(1)

[mae(3),mse(3),psnr(3)]= MSE_PSNR(out_pws((bwsy+1):(sy-bwsy)
    ,(bwsx+1):(sx-bwsx)),des2((bwsy+1):(sy-bwsy),(bwsx+1):(sx-
    bwsx)))
% figure(4); imstd (out_pws((bwsy+1):(sy-bwsy),(bwsx+1):(sx-
    bwsx))); title (['PWS reconstructed image, MSE = ',
    num2str(mse(3))]);

save result_1.mat W_w out_w code W_pws out_pws mse mae psnr
    des1 obs1 des2 obs2 wsx wsy M t

```

Listing D.2: Soft-PWS optimization routine for updating  $w$

```

clear all; close all; clc;

% load data
load result_1.mat;

% set timer
t1 = clock;
bwsy=(wsy-1)/2;          bwsx=(wsx-1)/2;

```

```

[ sy , sx ] = size ( des2 );

% soft PWS
% Generate S
S=sigma_gen(des1 , code );

% generate soft PWS results with hard partition weights
[ out_soft_h , mse_junk ] = filt_gspws_mse( obs1 , des1 , W_pws ,
    code , S );
% out_soft_h = soft_pws_filt_im2col( obs1 , code , W_pws , S , wsy , wsx
    );
t(3) = ( etime( clock , t1 ) / 60 )
[ mae(4) , mse(4) , psnr(4) ] = MSE_PSNR( des1 ( ( bwsy + 1 ) : ( sy - bwsy ) , (
    bwsx + 1 ) : ( sx - bwsx ) ) , out_soft_h )

% New soft weight optimization by Dr Russ Method
% W_soft = soft_pws_train_im2col( ken_aerial1 ( 1 : 128 , 1 : 128 ) , obs1
    ( 1 : 128 , 1 : 128 ) , A1 , S , wsy , wsx );
W_soft = soft_pws_train_im2col_divide( des1 , obs1 , code , S , wsy ,
    wsx , 64 );
% generate SPWS output
[ out_soft_w , mse_junk ] = filt_gspws_mse( obs1 , des1 , W_soft ,
    code , S );
% out_soft_w = soft_pws_filt_im2col( obs1 , code , W_soft , S , wsy ,
    wsx );
t(4) = ( etime( clock , t1 ) / 60 ) - t(3)

```

```

[mae(5),mse(5),psnr(5)]= MSE_PSNR(des1((bwsy+1):(sy-bwsy)),(
    bwsx+1):(sx-bwsx)),out_soft_w)

save result_2.mat out_soft_w W_soft out_soft_h mse mae psnr S
    t;

```

Listing D.3: Soft-PWS optimization routine for updating  $\mathbf{c}$

```

clear all; close all; clc;

% load data
load result_1.mat;      load result_2.mat;
% load result_3.mat C_nq;

t1 = clock;
bwsy=(wsy-1)/2;          bwsx=(wsx-1)/2;
[sy,sx]=size(des2);
niter_nq=2;
niter_gd=10;

C_tp = code;
% C_tp = C_nq;

% Soft PWS with new C algorithm
% New soft weight optimization
for ni = 1:10
    C_tp = rbf_pws_C_train(des1,obs1,W_soft,C_tp,S,niter_nq);

```

```

        C_tp = rbf_pws_C_gd_train(des1, obs1, W_soft, C_tp, S,
                                niter_gd);
    end

    C_nq=C_tp;
    % generate SPWS output
    [out_soft_c_nq, mse_junk] = filt_gspws_mse(obs1, des1, W_soft
        , C_nq, S);
    % out_soft_c_nq = soft_pws_filt_im2col(obs1, C_nq, W_soft, S, wsy
        , wsx);
    t(5) = (etime(clock, t1)/60)
    [mae(6), mse(6), psnr(6)] = MSE_PSNR(des1((bwsy+1):(sy-bwsy)), (
        bwsx+1):(sx-bwsx)), out_soft_c_nq)

    % W_soft = soft_pws_train_im2col(kenaaerial1(1:128, 1:128), obs1
        (1:128, 1:128), A1, S, wsy, wsx);
    save result_3.mat out_soft_c_nq C_nq mse mae psnr t;

```

Listing D.4: Quasi-Newton optimization routine for updating  $c$

```

function C_out = rbf_pws_C_train(des, obs, W, C, S, niter);

% Soft-PWS program: Quasi-Newton method for updating C
% Created by Yong Lin, University of Dayton, Yong.Lin@ieee.
    org
% Last Modified: Mar 2005

```

```

% function C_out = rbf_pws_C_train(des,obs,W,C,S,niter);
% obs — observation vectors x
% des — desired signal, located at the center of the
      observation window
% W — initial weight matrix
% C — initial codebook
% S — initial variance S
% niter — total number of iterations

[wss,M]=size(C);
[sy,sx]=size(obs); % Get image size info
wsy = sqrt(wss); wsx = sqrt(wss);
bwsy=(wsy-1)/2; bwsx=(wsx-1)/2;
numwin=wss; wss2=floor(wss/2)+1;
nm_M = numwin*M; nm_M2 = nm_M * nm_M;
% set Monte Carlo parameters
mc_sam = 250; mc_ite = 100; mc_sum = mc_sam*mc_ite;
% initial H and g
H = eye(wss*M); g = zeros(wss*M,1);

X_dt=im2col(des((bwsy+1):(sy-bwsy),(bwsx+1):(sx-bwsx)),[wsy,
      wsx], 'sliding ');
% [junk,nov]=size(X_dt);
X_d = X_dt(wss2,:); clear X_dt;
X_o=im2col(obs((bwsy+1):(sy-bwsy),(bwsx+1):(sx-bwsx)),[wsy,
      wsx], 'sliding ');

```

```

X_onm=X_o-repmat( mean(X_o), [wss, 1] );

% upgate g by using Monte Carlo method
g = g-gene(X_d, X_o, X_onm,W,C,S,mc_sum);
% a = 0; b = 10;
g_r = 0.618;

% loop for Quasi-Newton iterations
for i_acc=1:niter
    t2 = clock;
    s_kl = -H*g;
    s_k = im2col(s_kl, [wss 1], 'distinct ');
    % line search by golden ratio method for best possible
    parameter of alpha
    a = 0; b = 1000;
    for nz =1:100
        xy = a + g_r*(b-a);
        xx = b - g_r*(b-a);
        C_y = C + xy*s_k;
        C_x = C + xx*s_k;
        [junk, Jy] = filt_gspws_mse(obs, des, W, C_y, S);
        [out, Jx] = filt_gspws_mse(obs, des, W, C_x, S);
        if Jy > Jx
            b = xy;
        elseif Jy == Jx
            b= xy; a=xx;

```

```

        else
            a = xx;
        end
        if abs(b-a) < 0.0001
            break
        end
    end
end
% nz
xx
% xy
delt_k = xx*s_kl;
C = C + xx*s_k;
b = xx;
g_k = g;
% upgate g by using Monte Carlo method
g = g-gene(X_d, X_o, X_onm,W,C,S,mc_sum);
% upgate gama_k
gama_k = g - g_k;
% H(:, :, nj)=delt_k(:, nj)\gama_k(:, nj);
H_k=H+((delt_k-H*gama_k)*(delt_k-H*gama_k)')/((delt_k-H*
    gama_k)'*gama_k);
H= H_k;
% H = delt_k\gama_k;
% [out, mse] = filt_gspws_mse(obs, des, W, C, S);
out_y(:, :, i_acc) = out;
mse_y(i_acc) = Jx;

```

```

        alpha_y(i_acc) = xx;
        iter_y(i_acc) = nz;
        mse_y
        if Jx < 0.0001
            break
        end
        etime(clock,t2)/60
end;
C_out=C;

save result_C_nq_mc.mat mse_y alpha_y iter_y;

```

Listing D.5: Gradient descent optimization routine for updating  $c$

```

function C_out = rbf_pws_C_gd_train(des,obs,W,C,S,niter);

% Soft-PWS program: Gradient Descent method for updating C
% Created by Yong Lin, University of Dayton, Yong.Lin@ieee.
%   org
% Last Modified: Mar 2005

% function C_out = rbf_pws_C_gd_train(des,obs,W,C,S,niter);
% obs — observation vectors x
% des — desired signal, located at the center of the
%       observation window
% W — initial weight matrix
% C — initial codebook

```

```

% S — initial variance S
% niter — total number of iterations

[wss,M]=size(C);
[sy,sx]=size(obs); % Get image size info
wsy = sqrt(wss); wsx = sqrt(wss);
bwsy=(wsy-1)/2; bwsx=(wsx-1)/2;
numwin=wss; wss2=floor(wss/2)+1;
nm_M = numwin*M; nm_M2 = nm_M * nm_M;
% set Monte Carlo parameters
mc_sam = 250; mc_ite = 100; mc_sum = mc_sam*mc_ite;
% initial H and g
H = eye(wss*M); g = zeros(wss*M,1);

X_dt=im2col(des((bwsy+1):(sy-bwsy),(bwsx+1):(sx-bwsx)), [wsy,
    wsx], 'sliding ');
% [junk,nov]=size(X_dt);
X_d = X_dt(wss2,:); clear X_dt;
X_o=im2col(obs((bwsy+1):(sy-bwsy),(bwsx+1):(sx-bwsx)), [wsy,
    wsx], 'sliding ');
X_onm=X_o-repmat( mean(X_o), [wss, 1] );

% upgate g by using Monte Carlo method
g = g_gene(X_d, X_o, X_onm,W,C,S,mc_sum);

% loop for GD iterations

```

```

for i_acc=1:niter
    t2 = clock;
    s_kl = -g;
    s_k = im2col(s_kl, [wss 1], 'distinct');
    % line search by goldern ratio method for best possible
    parameter of alpha
    if (mod(i_acc, 10) == 1) % search every 10 iterations
        for saving time
            a = 0; b = 10000; g_r = 0.618;
            for nz =1:100
                xy = a + g_r*(b-a);
                xx = b - g_r*(b-a);
                C_y = C + xy*s_k;
                C_x = C + xx*s_k;
                [junk, Jy] = filt_gspws_mse(obs, des, W, C_y, S);
                [out, Jx] = filt_gspws_mse(obs, des, W, C_x, S);
                if Jy > Jx
                    b = xy;
                elseif Jy == Jx
                    b= xy; a=xx;
                else
                    a = xx;
                end
                if abs(b-a)<0.0001
                    break
                end
            end
        end
    end
end

```

```

        end
        xx
        % xy
    end
    % update C
    C = C + xx*s_k;
    % upgate g by using Monte Carlo method
    g = g-gene(X_d, X_o, X_onm,W,C,S,mc_sum);
    % generate SPWS filter output
    [out, mse] = filt_gspws_mse(obs, des, W, C, S);
    out_y(:, :, i_acc) = out;
    mse_y(i_acc) = mse;
%     mse_y(i_acc) = Jx;
    mse_y
    if Jx < 0.001
        break
    end
    etime(clock, t2)/60
end;
C_out=C;

save result_C_gd_gm_mc.mat mse_y;

```

Listing D.6: Soft-PWS optimization routine for updating s

```
clear all; close all; clc;
```

```

% load data
load result_1.mat;
load result_2.mat;
load result_3.mat;

t1 = clock;
bwsy=(wsy-1)/2;          bwsx=(wsx-1)/2;
[sy,sx]=size(des2);
niter_nq=2;
niter_gd=10;

% Soft PWS with new S algorithm

S_tp = S;

% New soft weight optimization
for ni = 1:10
    S_tp = rbf_pws_S_train(des1,obs1,W_soft,C_nq,S_tp,
        niter_nq);
    S_tp = rbf_pws_S_gd_train(des1,obs1,W_soft,C_nq,S_tp,
        niter_gd);
end

S_nq=S_tp;
% S_soft = soft_pws_S_train(des1,obs1,W_soft,C_nq,S,mu_s,thr,
    niter);

```

```

% out_soft_s_nq = soft_pws_filt_im2col(obs1,C_nq,W_soft,S_nq,
    wsy,wsx);
[out_soft_s_nq, mse_junk] = filt_gspws_mse(obs1, des1, W_soft
    , C_nq, S_nq);
t(5) = (etime(clock,t1)/60)
[mae(7),mse(7),psnr(7)]= MSEPSNR(des1((bwsy+1):(sy-bwsy)),(
    bwsx+1):(sx-bwsx)),out_soft_s_nq)

% W_soft = soft_pws_train_im2col(kenaaerial1(1:128,1:128),obs1
    (1:128,1:128),A1,S,wsy,wsx);
save result_4.mat out_soft_s_nq S_nq mse mae psnr t;

```

Listing D.7: Quasi-Newton optimization routine for updating  $s$

```

function S_out = rbf_pws_S_train(des,obs,W,C,S,niter);

% Soft-PWS program: Quasi-Newton method for updating S
% Created by Yong Lin, University of Dayton, Yong.Lin@ieee.
    org
% Last Modified: Mar 2005

% function S_out = rbf_pws_S_train(des,obs,W,C,S,niter);
% obs — observation vectors x
% des — desired signal, located at the center of the
    observation window
% W — initial weight matrix
% C — initial codebook

```

```

% S — initial variance S
% niter — total number of iterations

[wss,M]=size(C);
[sy,sx]=size(obs); % Get image size info
wsy = sqrt(wss); wsx = sqrt(wss);
bwsy=(wsy-1)/2; bwsx=(wsx-1)/2;
numwin=wss; wss2=floor(wss/2)+1;
nm_M = numwin*M; nm_M2 = nm_M * nm_M;
% set Monte Carlo parameters
mc_sam = 250; mc_ite = 100; mc_sum = mc_sam*mc_ite;
% initial H and g
wss1 = 1;
H = eye(wss1*M); g = zeros(wss1*M,1);

X_dt=im2col(des((bwsy+1):(sy-bwsy),(bwsx+1):(sx-bwsx)), [wsy,
    wsx], 'sliding ');
% [junk,nov]=size(X_dt);
X_d = X_dt(wss2,:); clear X_dt;
X_o=im2col(obs((bwsy+1):(sy-bwsy),(bwsx+1):(sx-bwsx)), [wsy,
    wsx], 'sliding ');
X_onm=X_o-repmat( mean(X_o), [wss, 1] );

% upgate g by using Monte Carlo method
g = g_gene_s(X_d, X_o, X_onm,W,C,S,mc_sum);
% a = 0; b = 10;

```

```

g-r = 0.618;

% loop for Quasi-Newton iterations
for i_acc=1:niter
    t2 = clock;
    s_kl = -H*g;
    s_k = im2col(s_kl, [wssl 1], 'distinct');
    % line search by golden ratio method for best possible
    parameter of alpha
    a = 0; b = 1000;
    for nz =1:100
        xy = a + g-r*(b-a);
        xx = b - g-r*(b-a);
        S_y = S + xy*s_k;
        S_x = S + xx*s_k;
        [junk, Jy] = filt_gspws_mse(obs, des, W, C, S_y);
        [out, Jx] = filt_gspws_mse(obs, des, W, C, S_x);
        if Jy > Jx
            b = xy;
        elseif Jy == Jx
            b= xy; a=xx;
        else
            a = xx;
        end
        if abs(b-a)<0.0001
            break
        end
    end
end

```

```

        end
    end
    % nz
    xx
    % xy
    delt_k = xx*s_kl;
    S = S + xx*s_k;
    b = xx;
    g_k = g;
    % upgate g by using Monte Carlo method
    g = g-gene_s(X_d, X_o, X_onm,W,C,S,mc_sum);
    % upgate gama_k
    gama_k = g - g_k;
    % H(:, :, nj)=delt_k(:, nj)\gama_k(:, nj);
    H_k=H+((delt_k-H*gama_k)*(delt_k-H*gama_k)')/((delt_k-H*
        gama_k)'*gama_k);
    H= H_k;
    % H = delt_k\gama_k;
    % [out, mse] = filt_gspws_mse(obs, des, W, C, S);
    out_y(:, :, i_acc) = out;
    mse_y(i_acc) = Jx;
    alpha_y(i_acc) = xx;
    iter_y(i_acc) = nz;
    mse_y
    if Jx < 0.0001
        break
    end
end

```

```

        end
        etime(clock,t2)/60
    end;
    S_out=S;

    save result_S_nq_mc.mat mse_y alpha_y iter_y;

```

Listing D.8: Gradient descent optimization routine for updated  $s$

```

function S_out = rbf_pws_S_gd_train(des,obs,W,C,S,niter);

% Soft-PWS program: Gradient Descent method for updating S
% Created by Yong Lin, University of Dayton, Yong.Lin@ieee.
%   org
% Last Modified: Mar 2005

% function S_out = rbf_pws_S_gd_train(des,obs,W,C,S,niter);
% obs — observation vectors x
% des — desired signal, located at the center of the
%       observation window
% W — initial weight matrix
% C — initial codebook
% S — initial variance S
% niter — total number of iterations

[wss,M]=size(C);
[sy,sx]=size(obs); % Get image size info

```

```

wsy = sqrt(wss); wsx = sqrt(wss);
bwsy=(wsy-1)/2; bwsx=(wsx-1)/2;
numwin=wss; wss2=floor(wss/2)+1;
nm_M = numwin*M; nm_M2 = nm_M * nm_M;
% set Monte Carlo parameters
mc_sam = 250; mc_ite = 100; mc_sum = mc_sam*mc_ite;
% initial H and g
wss1 = 1;
H = eye(wss1*M); g = zeros(wss1*M,1);

X_dt=im2col(des((bwsy+1):(sy-bwsy),(bwsx+1):(sx-bwsx)), [wsy,
    wsx], 'sliding ');
% [junk, nov]=size(X_dt);
X_d = X_dt(wss2,:); clear X_dt;
X_o=im2col(obs((bwsy+1):(sy-bwsy),(bwsx+1):(sx-bwsx)), [wsy,
    wsx], 'sliding ');
X_onm=X_o-repmat( mean(X_o), [wss, 1] );

% upgate g by using Monte Carlo method
g = g_gene_s(X_d, X_o, X_onm, W, C, S, mc_sum);

% loop for GD iterations
for i_acc=1:niter
    t2 = clock;
    s_kl = -g;
    s_k = im2col(s_kl, [wss1 1], 'distinct ');

```

```

% line search by goldern ratio method for best possible
parameter of alpha
if (mod(i_acc , 10) == 1) % search every 10 iterations
    for saving time
        a = 0; b = 10000; g_r = 0.618;
        for nz =1:100
            xy = a + g_r*(b-a);
            xx = b - g_r*(b-a);
            S_y = S + xy*s_k;
            S_x = S + xx*s_k;
            [junk, Jy] = filt_gspws_mse(obs, des, W, C, S_y);
            [out, Jx] = filt_gspws_mse(obs, des, W, C, S_x);
            if Jy > Jx
                b = xy;
            elseif Jy == Jx
                b= xy; a=xx;
            else
                a = xx;
            end
            if abs(b-a)<0.0001
                break
            end
        end
    end
    xx
    % xy
end

```

```

% update C
S = S + xx*s_k;
% upgate g by using Monte Carlo method
g = g_gene_s(X_d, X_o, X_onm,W,C,S,mc_sum);
% generate SPWS filter output
[out, mse] = filt_gspws_mse(obs, des, W, C, S);
out_y(:, :, i_acc) = out;
mse_y(i_acc) = mse;
%     mse_y(i_acc) = Jx;
mse_y
if Jx < 0.001
    break
end
etime(clock, t2)/60
end;
S_out=S;

save result_S-gd-gm-mc.mat mse_y;

```

#### Listing D.9: Soft-PWS filtering routine

```

clear all; close all; clc;

% load data
load result_1.mat;
load result_2.mat;
load result_3.mat;

```

```

load result_4.mat;
% load result_2a.mat W_soft2;
load test_car10.mat obs2 des2
% C_nq = code;
clear mse mae psnr t out* obs1 des1;

t1 = clock;
bwsy=(wsy-1)/2;          bwsx=(wsx-1)/2;
[ sy , sx]=size (des2);

% generate SPWS output
out_w = wienvqvqf(obs2 , zeros (wsx*wsy , 1) , W_w , wsy , wsx );
t(1)=(etime (clock , t1)/60)
[mae(1) , mse(1) , psnr (1)]= MSE_PSNR(obs2 (( bwsy+1):( sy-bwsy) ,(
    bwsx+1):( sx-bwsx)) , des2 (( bwsy+1):( sy-bwsy) ,( bwsx+1):( sx-
    bwsx)))
[mae(2) , mse(2) , psnr (2)]= MSE_PSNR(out_w (( bwsy+1):( sy-bwsy) ,(
    bwsx+1):( sx-bwsx)) , des2 (( bwsy+1):( sy-bwsy) ,( bwsx+1):( sx-
    bwsx)))
out_pws = wienvqvqf(obs2 , code , W_pws , wsy , wsx );
t(2)=(etime (clock , t1)/60)-t(1)
[mae(3) , mse(3) , psnr (3)]= MSE_PSNR(out_pws (( bwsy+1):( sy-bwsy)
    ,( bwsx+1):( sx-bwsx)) , des2 (( bwsy+1):( sy-bwsy) ,( bwsx+1):( sx-
    bwsx)))
figure (1); imstd (des2 (( bwsy+1):( sy-bwsy) ,( bwsx+1):( sx-bwsx))
    );

```

```

% title ('Training image');
figure(2); imstd (obs2((bwsy+1):(sy-bwsy),(bwsx+1):(sx-bwsx))
);
% title ('Test image');
% title (['noisy image, MSE = ', num2str(mse(1))])
figure(3); imstd (out_w((bwsy+1):(sy-bwsy),(bwsx+1):(sx-bwsx)
));
% title ('Wiener result image');
% title (['Wiener reconstructed image, MSE = ', num2str(mse
(2))])
figure(4); imstd (out_pws((bwsy+1):(sy-bwsy),(bwsx+1):(sx-
bwsx)));
% title ('PWS result image');
% title (['PWS reconstructed image, MSE = ', num2str(mse(3))
]);

[out_soft_w, mse_junk] = filt_gspws_mse(obs2, des2, W_soft,
code, S);
% out_soft_w = soft_pws_filt_im2col(obs2,code,W_soft,S,wsy,
wsx);
t(3) = (etime(clock,t1)/60)-t(2)
[mae(4),mse(4),psnr(4)]= MSE_PSNR(des2((bwsy+1):(sy-bwsy),(
bwsx+1):(sx-bwsx)),out_soft_w)
figure(5); imstd (out_soft_w);

```

```
save result_5_car.mat obs2 des2 out_w out_pws out_soft_w mse  
    mae psnr t;
```

## BIBLIOGRAPHY

- [1] S. V. Vaseghi, *Advanced Digital Signal Processing and Noise Reduction*. John Wiley & Sons, 2000.
- [2] K. E. Barner, A. M. Sarhan, and R. C. Hardie, "Partition-based weighted sum filters for image restoration," *IEEE Trans. Image Processing*, vol. 8, pp. 740–745, May 1999.
- [3] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantization," *IEEE Trans. Commun. Theory*, vol. COMM-28, pp. 84–95, Jan 1980.
- [4] M. Shao, *Partition-based Weighted Sum Filtering Theory with Applications to Image Processing and Biomedical Engineering*. PhD thesis, University of Delaware, 2004.
- [5] M. Shao and K. E. Barner, "Optimization of partition based weighted sum filters," *IEEE EURASIP Nonlinear Signal and Image Processing (NSIP) Workshop, (Baltimore, MD)*, June 2001.
- [6] M. Shao and K. E. Barner, "Optimization of partition-based weighted sum filters and their application to image denoising," *IEEE Trans. Image Processing*, October 2004. Accepted for publication.
- [7] M. Shao, K. E. Barner, and R. C. Hardie, "Partition-based interpolation for image demosaicking and super-resolution reconstruction," *Optical Engineering*. Accepted for publication.
- [8] A. Cichocki and S. ichi Amari, *Adaptive Blind Signal and Image Processing*. John Wiley & Sons, 2002.
- [9] T. F. Chan and C. Wong, "Total variation blind deconvolution," *IEEE Trans. Image Processing*, vol. 7, pp. 370–375, March 1998.
- [10] P. Blomgren and T. F. Chan, "Color tv: Total variation methods for restoration of vector valued images," *IEEE Trans. Image Processing*, vol. 7, pp. 304–309, March 1998.

- [11] R. H. Chan, T. F. Chan, and C.-K. Wong, "Cosine transform based preconditioners of total variation deblurring," *IEEE Trans. Image Processing*, vol. 8, pp. 1472–1478, October 1999.
- [12] T. F. Chan, S. Osher, and J. Shen, "The digital tv filter and nonlinear denoising," *IEEE Trans. Image Processing*, vol. 10, pp. 231–241, February 2001.
- [13] R. Nakagaki and A. K. Katsaggelos, "A vq-based blind image restoration algorithm," *IEEE Trans. Image Processing*, vol. 12, pp. 1044–1053, Sep 2003.
- [14] J. C. Bezdek, *Pattern Recognition With Fuzzy Objective Function Algorithms (Advanced applications in pattern recognition)*. New York: Plenum, 1981.
- [15] K. E. Barner, G. R. Arce, and J.-H. Lin, "On the performance of stack filters and vector detection in image restoration," *Circuits, Systems, and Signal Processing*, vol. 11, pp. 153–169, Jan 1992.
- [16] R. C. Hardie and K. E. Barner, "Rank conditioned rank selection filters for signal restoration," vol. 3, pp. 192–206, Mar. 1994.
- [17] E. Abreu, M. Lightstone, S. Mitra, and K. Arakawa, "A new efficient approach for the removal of impulsive noise from highly corrupted images," *IEEE Trans. Image Processing*, vol. 5, pp. 1012–1025, June 1996.
- [18] D. G. Sheppard, K. Panchapakesan, A. Bilgin, B. R. Hunt, and M. W. Marcellin, "Lapped nonlinear interpolative vector quantization and image super-resolution," *IEEE Trans. Image Processing*, vol. 9, pp. 295–298, Feb 2000.
- [19] T. Chen and H. R. Wu, "Application of partition-based median type filters for suppressing noise in images," *IEEE Trans. Image Processing*, vol. 10, pp. 829–836, June 2001.
- [20] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression (The Kluwer International Series in Engineering and Computer Science)*. Norwell, MA: Kluwer, 1992.
- [21] L. Ungar, T. Johnson, and R. de Veaux, "Radial basis function neural networks for process control," *CIMPRO Proceedings*, pp. 357–364, 1994.
- [22] H. Peng, T. Ozaki, V. Haggan-Ozaki, and Y. Toyoda, "A parameter optimization method for radial basis function type models," *IEEE Trans. Neural Networks*, vol. 14, pp. 432–438, March 2003.
- [23] R. Fletcher, *Practical Methods of Optimization*. John Wiley & Sons, 1987.

- [24] T. H. Sidebotham, *The A to Z of Mathematics, A Basic Guide*. John Wiley & Sons, 2002.
- [25] L. D. Berkovitz, *Convexity and Optimization in  $\mathbb{R}^n$* . John Wiley & Sons, 2002.
- [26] C. J. Zarowski, *An Introduction To Numerical Analysis For Electrical And Computer Engineers*. John Wiley & Sons, 2004.
- [27] M. S. Grewal and A. P. Andrews, *Kalman Filtering: Theory and Practice Using Matlab*. John Wiley & Sons, 2001.
- [28] R. L. Haupt and S. E. Haupt, *Practical Genetic Algorithms*. John Wiley & Sons, 2004.
- [29] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.
- [30] K. R. Castleman, *Digital Image Processing*. Prentice Hall, 1995.
- [31] R. M. Howard, *Principles of Random Signal Analysis and Low Noise Design: The Power Spectral Density and its Applications*. John Wiley & Sons, 2002.
- [32] W. K. Pratt, *Digital Image Processing*. John Wiley & Sons, 2001.
- [33] M. Petrou and P. Bosdogianni, *Image Processing: The Fundamentals*. John Wiley & Sons, 1999.
- [34] J. C. Russ, *The Image Processing Handbook*. CRC Press, 2002.
- [35] T. Bose, *Digital Signal and Image Processing*. John Wiley & Sons, 2004.
- [36] J. C. Goswami and A. K. Chan, *Fundamentals of Wavelet Theory, Algorithms and Applications*. John Wiley & Sons, 1999.
- [37] A. C. Kak and M. Slaney, *Principles of Computerized Tomographic Imaging*. The Institute of Electrical and Electronics Engineers, Inc., 1999.
- [38] V. Castelli and L. D. Bergman, *Image Databases: Search and Retrieval of Digital Imagery*. John Wiley & Sons, 2002.
- [39] Y. Lin, R. C. Hardie, and K. E. Barner, "Subspace partition weighted sum filters for image deconvolution," *2005 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2005)*, (Philadelphia, PA), March 2005.
- [40] Y. Lin, R. C. Hardie, and K. E. Barner, "Subspace partition weighted sum filters for image restoration," *IEEE Signal Processing Letter*, September 2005. To be appeared.

R702031873

## VITA

Yong Lin received the B.S.E.E. degree from Northern JiaoTong University, Beijing, China, in 1989. Yong received the M.S.E.E. degrees from Hokkaido Institute of Technology, Sapporo, Japan, in 1992. Since 2002, Yong has been a graduate student with Department of Electrical and Computer Engineering, University of Dayton, Dayton, OH. From 1992 to 1998, Yong was a Senior Consultant/Engineer at NCR (former AT&T Global Information Solutions). From 1998 to 2001, Yong was a Product Manager at Genesys Telecommunication Laboratories (an Alcatel company). His research interests include signal and image processing, communication, robust signal processing, and non-linear systems. He now lives in Mason, Ohio, with his wife Tao and his son Astin.